UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS
FACULTAD 3

SISTEMA INFORMÁTICO PARA EL CONTROL DEL PROCESO DE ENTREGA Y RECIBO DE ACTIVOS FIJOS

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE
INGENIERO EN CIENCIAS INFORMÁTICAS

AUTORES: Alejandro Miguel Saborit Figueroa
José Javier Alemán Álvarez

TUTORES: **Ing**. Annia Verdecia Boza
Ing. Leodanny Wuilber Polanco Garay

ASESOR: **Ing.** Zénel Reyes Pérez

LA HABANA, 2015

DECLARACIÓN DE AUTORÍA

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

Para que así conste, firmamos la presente a los __ días del mes de ____ del año ____.

_____

Firma del Autor

Alejandro M. Saborit Figueroa

_____

Firma del Autor

José Javier Alemán Álvarez

_____

Firma del Tutor

Ing. Annia Verdecia Boza

_____

Firma del Tutor

Ing. Leodanny Wuilber Polanco Garay

José Javier:

… A los tres grandes pilares de mi vida, mi mamá Yamila, mi papá José Octavio y a mi abuela Hortensia que en el cielo está.

Alejandro:

… A mi mama, que siempre me dio fuerzas para seguir adelante, para vencer todos los obstáculos que se pusieron en mi camino, por tu infinita paciencia y todo tu sacrificio, este resultado también es tuyo. A mi papa por todos sus consejos, por inculcar en mí todos esos valores humanos que me han hecho la persona que soy. A mi hermana, por su confianza, por toda tu ayuda, por tu amor y por regalarme todo ese apoyo incondicional. A mi novia por haber transitado junto a mí a lo largo de este año, por muchas razones no ha sido fácil y contar con tu apoyo ha sido mi sustento.

José Javier:

Tengo el honor de mencionar y agradecer a las personas que sin ellos no hubiera sido posible este gran reto. Primeramente nombrar a mis padres Yamila Álvarez y José Octavio Alemán, por creer en mí, así como en mis decisiones y metas, por apoyarme y guiarme por el buen camino, por ayudarme a formarme más que en un ingeniero, en un hombre de bien, por el sacrificio asumido en estos cinco años teniéndome fuera del calor del hogar, por ser los padres más excepcionales del mundo, por ser fieles a mi credo, por ser siempre las dos luces que alumbraron mi camino en los buenos y malos momentos, por los sabios consejos dados, a ustedes: mami y papi, mis más grandes y fieles amigos … *GRACIAS* …

Quiero agradecer de forma especial a mi novia Aymée Perdomo, quien desde su posición supo ayudarme y darme la tranquilidad necesaria en los momentos críticos del desarrollo de esta tesis. Agradezco además junto a ella, a mis compañeros de estudios y andanzas, Roberto Alfonso (Robe), Rigoberto Peña (Rigo), Rey Manuel Lazo (mariquita), Ivaniet (el tío), Yoandry (Yoa), Juan Darién Macías, Danny, Dariel Rojas, Pablo Antonio Moreno García (Chenco) y Alejandro Miguel Saborit (Sabo); sin la ayuda de ninguno de los antes mencionados esta tesis estuviera a la altura de lo que es hoy, señalando en especial a estos dos últimos: a Chenco y a Sabo; uno por volverme loco al extremo y al otro por ser mi magnífico compañero de tesis, sin el CERAF no hubiera sido CERAF. A ambos por ser más que mis amigos y por haberse convertido sin que se los pidiera en mis hermanos aunque no sean de sangre, GRACIAS, siempre tendrán un lugar especial en mi corazón…

Para las personas con las que alguna vez fui parte de su familia, Reglita, Rolando e Iyemay (Iye), quienes llegué a querer como mi propia familia, GRACIAS…

A mis tutores Annia y Polanco por la guía y apoyo brindado en todo este largo y estresante proceso, a ellos, GRACIAS. Además quiero mencionar a toda esa gran brigada de la que fui parte a lo largo de estos ya esfumados cinco años: El 3107, de ahí señalar especialmente a Laurita, Leydis, Indra, Yeisel, Alexei, Efraín, Lian, Michel, El Kuty (Reinel), El Yerkill (Yuri) y Luiso. Agradecer además a las muchachas más lindas de la Facultad 2, a Melisco (Melissa), Sole (Solangel) y Zurelol (Zurelis) por compartir con nosotros lindos e históricos momentos. En fin a todos mil gracias por haber sido parte de la historia de mi etapa tan linda que fue la universidad…

Muchas gracias también a los que la mente no me dio la posibilidad de recordar en el momento de esta redacción…

Alejandro:

A mi mama, por ser el motor impulsor de cada proyecto trazado en mi vida, por nunca decirme que no, por ese apoyo incondicional que me has mostrado siempre, por hacerme el centro de tu universo. Gracias por tu sacrificio y tu inagotable cariño, por tu confianza, por hacer de mí el brillo de tus ojos. A mi papa por todos sus consejos, por ser mí meta, por guiarme siempre por un camino de bien y apoyarme siempre que lo he necesitado. A ustedes, una y mil veces. Muchas Gracias.

A mis cuatro abuelos, en especial a mi abuela Dulce, que a pesar de su delicada salud, nunca he estado ni un segundo lejos de sus pensamientos; gracias “abue” por todo el tiempo que me has regalado y por todas tus enseñanzas. Agradezco a mi hermana pues a pesar de los pocos años de diferencia, te has convertido en una madre para mí, gracias por tus consejos y tu infinito cariño, te quiero mucho.

A mi novia, por tu infinito amor, gracias por haber llegado a mi vida, por comprenderme y apoyarme en estos meses tan difíciles, por ser la constante de mi vida y hacerme ver que aunque las cosas se pongan feas, si estamos juntos, todo estará bien.

A José Javier, primeramente, gracias por todas las madrugadas de trabajo que hicieron posible que este investigación saliera adelante, gracias por convertirte en estos cinco años en un hermano para mí, gracias por tus consejos y por haber compartido conmigo los mejores momentos que viví en la UCI.

A mis tutores, por su apoyo, dedicación y colaboración en el desarrollo de este trabajo. A todos los buenos amigos y amigas de estos 5 años, a Laurita, Yeisel, José Javier, Roberto, Pablo, Kuty, Rey y Yoandry, a todos ustedes gracias por todos los momentos de alegría, por permitirme contar con su amistad por el resto de mi vida.

A todos mis compañeros de grupo, de todos me llevo lo mejor, gracias por dejarme compartir junto a ustedes esta etapa inolvidable.

A todos los que han contribuido a mi desarrollo profesional y al resultado de este trabajo. En general, a todas las personas que me preguntaron alguna vez: “Como va la tesis”, que de una forma u otra me dieron fuerza para seguir adelante, sin ustedes no podría haberlo conseguido.

A todos, muchas gracias.

RESUMEN

El Vicedecanato de Administración de la Facultad 3 y el Departamento de Tecnología de la Universidad de las Ciencias Informáticas, tienen la responsabilidad de controlar físicamente todos los activos fijos con que cuentan las distintas áreas. El presente trabajo tiene como objetivo la realización de una aplicación web que permita informatizar el proceso de entrega y recibo de activos fijos de la Facultad 3 que hoy se realizan de manera manual. La solución permitirá establecer las bases para lograr un mayor control y seguridad sobre el proceso mencionado, logrando así reducir gastos y disminuir el tiempo en que ocurren las actividades realizadas. Permitiendo a los directivos de esta área tener un control elevado sobre todos los medios, dando la posibilidad además, de conocer el estado de las incidencias y de los reportes generados para cada activo. Para la investigación se emplearon los métodos empíricos de *entrevista, método experimental y método de observación científi*ca, por otra parte, fueron utilizados los métodos teóricos: *inductivo-deductivo, analítico-sintético y la modelación*.

Palabras claves: activos fijos, control de entrega y recibo, incidencias, reportes, seguridad.

ÍNDICE

Resumen v
INTRODUCCIÓN 1
CAPÍTULO 1: Enfoques investigativos sobre el proceso de entrega y recibo de activos fijos 5
1.1. Estudio del estado del arte 5
1.2. Metodología de desarrollo de software 10
1.3. Lenguaje y herramienta de modelado 12
1.4. Arquitectura de software 13
1.5. Ambiente de desarrollo 16
CAPÍTULO 2: Propuesta de solución 19
2.1. Requisitos 19
Modelo Conceptual 19
Técnicas para la captura de requisitos funcionales 20
Definición de los requisitos funcionales 21
Historias de Usuario 23
Descripción de los requisitos no funcionales 25
Validación de los requisitos funcionales 27
2.2. Análisis y diseño 28
Descripción de la arquitectura 28
Diagrama de paquetes 30
Diagrama de clases del diseño 31
Utilización de los patrones de diseño 32
Validación del diseño 37
CAPÍTULO 3: Implementación y Validación de la Propuesta de solución 44
3.1. Implementación 44
Diagrama de componente 44
Diagrama de despliegue 45
Estándar de codificación 47
3.2. Pruebas Internas 50
Pruebas unitarias 50

Pruebas de caja negra 51
3.3. Pruebas de Aceptación con el cliente 55
3.4. Control sobre los AF y fácil manejo y tratamiento de la información manipulada. 56
Conclusiones 62
Recomendaciones 63
Referencias Bibliográficas 64
Glosario de términos 67
Anexos 69
Anexo 1: Entrevista para la captura de requisitos 69
Anexo 2: Historias de Usuario 69
Anexo 3: Diagramas de Clases del diseño 75
Anexo 4: Diagramas de Casos de Pruebas (DCP) 79
Anexo 5: Carta de Aceptación del Cliente 88

ÍNDICE DE ILUSTRACIONES

Ilustración 1. Representación de la arquitectura MVC en Symfony2. Fuente: (15). 14
Ilustración 2. Modelo conceptual del negocio. Fuente: elaboración propia. 20
Ilustración 3. Prototipo para la captura del requisito Administrar área. Fuente: elaboración propia. ... 27
Ilustración 4. El patrón MVC en Symfony2. Fuente: elaboración propia. 29
Ilustración 5. Modelo de funcionamiento de Symfony2 (Capa controladora). Fuente: (29). 29
Ilustración 6. Flujo de trabajo de Symfony. Fuente: (29). 30
Ilustración 7. Diagrama de Paquete del sistema a desarrollar. Fuente: elaboración propia. 31
Ilustración 8. Diagrama de clase del diseño del escenario IER. Fuente: elaboración propia. 32
Ilustración 9. Uso del patrón experto en la clase ActaCierreController. Fuente: elaboración propia.... 33
Ilustración 10. Uso del patrón creador en la clase ActivoFijoController. Fuente: elaboración propia... 33
Ilustración 11. Uso del patrón controlador en la clase TipoAFController. Fuente: elaboración propia. 34
Ilustración 12. Uso del patrón Alta Cohesión en la clase UsuarioController. Fuente: elaboración propia. 34
Ilustración 13. Uso del patrón decorador. Fuente: elaboración propia. 35
Ilustración 14. Uso del patrón Inyección de dependencia. Fuente: elaboración propia. 35
Ilustración 15. Modelo de datos de la solución propuesta. Fuente: elaboración propia. 36
Ilustración 16. Atributos de calidad para las pruebas TOC. Fuente: elaboración propia. 39

Ilustración 17. Comportamiento de los atributos de calidad de la métrica TOC. Fuente: elaboración propia. 39
Ilustración 18. Umbrales utilizados por la métrica RC. Fuente: elaboración propia. 41
Ilustración 19. Comportamiento de los atributos de calidad de la métrica RC. Fuente: elaboración propia. 42
Ilustración 20. Diagrama de componentes del sistema CERAF. Fuente: elaboración propia. 45
Ilustración 21. Diagrama de despliegue del sistema CERAF. Fuente: elaboración propia. 46
Ilustración 22. Nomenclatura de comentario en las clases del sistema. Fuente: elaboración propia... 48
Ilustración 23. Nomenclatura de comentario en las funciones del sistema. Fuente: elaboración propia. 48
Ilustración 24. Mensaje de error en el formulario IER. Fuente: elaboración propia. 49
Ilustración 25. Mensaje de confirmación del sistema. Ejemplo: Modificación del IER. Fuente: elaboración propia. 49
Ilustración 26. Mensaje de información del sistema. Fuente: elaboración propia. 49
Ilustración 27. Resultado de las pruebas internas realizadas al código. Fuente: elaboración propia. . 51
Ilustración 28. Escenario de prueba Adicionar IER introduciendo datos válidos. Fuente: elaboración propia. 54
Ilustración 29. Escenario de prueba Adicionar IER dejando al menos un campo vacío. Fuente: elaboración propia. 54
Ilustración 30. Resultado de las pruebas funcionales. Fuente: elaboración propia. 55
Ilustración 31. Cantidad de no conformidades encontradas por iteración. Fuente: elaboración propia. 56
Ilustración 32. Inventario de AF. Fuente: elaboración propia. 57
Ilustración 33. Listado de IER creados en el día. Fuente: elaboración propia. 57
Ilustración 34. Detalle del IER creado. Fuente: elaboración propia. 58
Ilustración 35. Listado de IER creados por local. Fuente: elaboración propia. 59
Ilustración 36. Listado de IER por fecha de creación. Fuente: elaboración propia. 59
Ilustración 37. Listado de Incidencias. Fuente: elaboración propia. 60
Ilustración 38. DCD Acta de Responsabilidad. Fuente: elaboración propia. 75
Ilustración 39. DCD: Gestionar AF. Fuente: elaboración propia. 76
Ilustración 40. DCD Gestionar Área. Fuente: elaboración propia. 77
Ilustración 41. DCD Gestionar Local. Fuente: elaboración propia. 78
Ilustración 42. DCD Gestionar Tipo AF. Fuente: elaboración propia. 79

ÍNDICE DE TABLA

Tabla 1. Comparación entre los sistemas estudiados. Fuente: elaboración propia. 9
Tabla 3. Disciplinas de la metodología AUP-UCI. Fuente: (11). 11
Tabla 4. Requisitos funcionales del software. Fuente: elaboración propia. 22
Tabla 5. Historia de Usuario de los escenarios: Adicionar AF y Listar Reportes de IER por local. Fuente: elaboración propia. 24
Tabla 6. Requisitos no funcionales del software. Fuente: elaboración propia. 25
Tabla 7. Cantidad de clases según los atributos de calidad: Responsabilidad, Complejidad de implementación y Reutilización. Fuente: elaboración propia. 38
Tabla 8. Atributos que posibilita medir la métrica TOC. Fuente: elaboración propia. 39
Tabla 9. Cantidad de clases según los atributos de calidad: Acoplamiento, Complejidad de mantenimiento, Reutilización y Cantidad de pruebas propuestos por la métrica RC. Fuente: elaboración propia. 40
Tabla 10. Modo en que se afectan los atributos de calidad. Fuente: elaboración propia. 41
Tabla 11. Caso de Prueba: Crear IER. Fuente: elaboración propia. 52
Tabla 12. Descripción de las variables utilizadas en el DCP: Crear IER. Fuente: elaboración propia. 53
Tabla 13. Historia de Usuario: Adicionar Área. Fuente: elaboración propia. 69
Tabla 14. Historia de Usuario: Adicionar Elemento. Fuente: elaboración propia. 70
Tabla 15. Historia de Usuario: Adicionar Área. Fuente: elaboración propia. 71
Tabla 16. Historia de Usuario: Asignar Medios a Áreas. Fuente: elaboración propia. 71
Tabla 17. Historia de Usuario: IER. Fuente: elaboración propia. 72
Tabla 18. Historia de Usuario: Inventario diario. Fuente: elaboración propia. 73
Tabla 24. DCP: Adicionar Acta de Cierre. Fuente: elaboración propia. 80
Tabla 25. Descripción de las variables del DCP: Adicionar Acta de cierre. Fuente: elaboración propia. 80
Tabla 26. DCP: Adicionar AF. Fuente: elaboración propia. 81
Tabla 27. Descripción de las variables del DCP: Adicionar AF. Fuente: elaboración propia. 81
Tabla 28. DCP: Editar AF. Fuente: elaboración propia. 82
Tabla 29. Descripción de las variables del DCP: Editar AF. Fuente: elaboración propia. 82
Tabla 30. DCP: Adicionar Local. Fuente: elaboración propia. 83
Tabla 31. Descripción de las variables del DCP: Adicionar Local. Fuente: elaboración propia. 83
Tabla 32. DCP: Crear Incidencia. Fuente: elaboración propia. 84
Tabla 33. Descripción de las variables del DCP: Crear Incidencia. Fuente: elaboración propia. 84

Tabla 34. DCP: Adicionar Modelo. Fuente: elaboración propia. 85
Tabla 35. Descripción de las variables del DCP: Adicionar Modelo. Fuente: elaboración propia. 85
Tabla 36. DCP: Adicionar Usuario. Fuente: elaboración propia. 85
Tabla 37. Descripción de las variables del DCP: Adicionar Usuario. Fuente: elaboración propia. 86
Tabla 38. DCP: Adicionar Reporte Tipo AF. Fuente: elaboración propia. 87
Tabla 39. Descripción de las variables del DCP: Adicionar Reporte Tipo AF. Fuente: elaboración propia. 87

INTRODUCCIÓN

Las tecnologías de la información y las comunicaciones se han ido insertando en gran parte de las actividades del quehacer humano, modificando la forma de pensar y actuar de las personas. Su uso se torna casi imprescindible para estar acorde al acelerado desarrollo en todos los sectores de la sociedad actual, en la cual la cubana no es un ente aislado en esta rápida evolución, abogando por la informatización de varias instituciones. Uno de los principales actores de este desarrollo en el país es la Universidad de las Ciencias Informáticas (UCI), la misma, prioriza la creación de proyectos de diversas índoles, trayendo consigo el desarrollo de herramientas y soluciones informáticas para gestionar algunos procesos internos. Basado en esta filosofía, el área de Administración de la Facultad 3 ha planteado la necesidad de informatizar el control de entrega y recibo de sus activos fijos (AF).

Actualmente, las empresas inmersas en su trabajo rutinario, le proporcionan importancia a controlar físicamente sus principales activos. En la mayoría de estas, pequeñas y medianas, la responsabilidad de resguardar y controlar los activos, en el mejor de los casos, ha recaído en el área de Contabilidad pues generalmente no existe una única persona que lleve un control adecuado y se responsabilice de éstos. En las grandes empresas, incluso puede existir un área específica de control de activos e inversiones, sin embargo siempre dependen de la dirección de Finanzas y Administración (1).

Uno de los grandes problemas que presentan los AF son los movimientos. Mantener actualizadas estas acciones resulta complejo para las empresas, debido a que las personas que deben ejecutarlos no les dan la importancia que requiere. Los encargados de realizar estos movimientos, se ocupan de otras funciones que no permiten dedicarle el tiempo necesario a dicha actividad.

En la universidad la estructura se encuentra sujeta a constantes cambios, creando, modificando y desapareciendo áreas, por tanto es muy complejo seguir la trazabilidad de los activos y su ubicación exacta. Además, al no realizarse la actividad de control del inventario periódicamente es mayor el tiempo que hay que emplear cuando se decide hacerla. En la Facultad 3, como parte de la universidad, existen los siguientes problemas:

- El proceso de control de entrega y recibo de los AF, se realiza de manera manual utilizando papeles impresos que incrementan el consumo de recursos materiales y disminuye la productividad de los trabajadores.
- La gestión de la información de los activos de meses anteriores se hace tediosa por la búsqueda entre varios papeles, lo cual provoca un gasto mayor del tiempo que se emplea para el tratamiento de dicha información.

- El control sobre una observación determinada de un activo puede pasar desapercibida varios días después de realizada.
- El reporte de incidencias sobre los activos para un determinado período de tiempo y cierto número de áreas resulta trabajoso por la cantidad de reportes a revisar; debido a que estos pueden no contener todos los datos que se requieren.
- Generalmente, la entrega y recibo de un área en un período determinado debe realizarla el jefe del área como responsable de los medios del local, pero no se puede asegurar que el proceso haya sido realizado por dicho responsable aunque el reporte tenga su firma. Por lo que no se puede garantizar la autenticidad ni el no repudio de los reportes generados en el proceso.

Por todo lo antes mencionado se identificó como *problema de la investigación*: ¿cómo mejorar el proceso de control de entrega y recibo de AF en el área de Administración de la Facultad 3?

El *objeto de estudio* se enmarca en el proceso de entrega y recibo de AF, delimitando como *campo de acción* los sistemas informáticos encargados del control de AF.

Siguiendo como *objetivo general* desarrollar un sistema informático para el área de Administración de la Facultad 3 que permita gestionar el proceso de control de entrega y recibo de AF, además de garantizar la autenticidad y no repudio de los reportes de guardias generados.

Para dar solución al problema descrito se tiene como *idea a defender*: “con el desarrollo de una solución informática para el proceso de control de entrega y recibo de AF, dirigido al área de Administración de la Facultad 3, se contribuirá a mejorar el control, procesamiento de la información de los activos y garantizará la autenticidad y no repudio de los reportes generados por las personas autorizadas en el proceso”.
De acuerdo a lo planteado anteriormente se definen como *objetivos específicos*:

1. Elaborar la fundamentación teórica de la investigación a partir del estudio del proceso de control de entrega y recibo de AF para conocer su funcionamiento dentro del área implicada.
2. Desarrollar los artefactos de la disciplina de requisitos.
3. Confeccionar los artefactos de la disciplina de análisis y diseño de la solución.
4. Implementar la solución propuesta.
5. Ejecutar las pruebas de funcionalidad, pruebas unitarias y aceptación para la aplicación.

Para darle cumplimento a los objetivos se trazan las siguientes *tareas de investigación*:

- Estudio de sistemas y procesos para el control de entrega y recibo de activos.
- Estudio del proceso de gestión y control de entrega y recibo de los activos en la Facultad 3.
- Elaboración de la fundamentación teórica para definir los principales conceptos asociados al campo de acción y al objetivo de la investigación.
- Caracterización de las tecnologías y herramientas a utilizar en el desarrollo del sistema.
- Identificación de los requisitos funcionales a partir de las técnicas para la captura de requisitos.
- Desarrollo del artefacto historias de usuario.
- Validación de los requisitos funcionales.
- Desarrollo del artefacto diagrama de clases del diseño.
- Desarrollo del artefacto modelo de datos.
- Desarrollo del artefacto diagrama de componentes.
- Validación del diseño.
- Implementación de los requisitos descritos.
- Desarrollo del artefacto diseño de casos de prueba.
- Desarrollo de las pruebas internas y de aceptación.

Los métodos de investigación científica utilizados en esta investigación son:

Métodos empíricos:

- El criterio de expertos: para consultar varios expertos que permitan validar la propuesta de solución, basada en conocimientos, investigaciones, experiencias y estudios bibliográficos, que brinden una opinión profesional sobre el funcionamiento del control de los AF.
- La entrevista: se realiza a las personas encargadas de gestionar la entrega y recibo de los activos con el objetivo de precisar el problema a resolver e identificar el modo de funcionamiento de la misma. Inicialmente se realizó una entrevista no estructurada permitiendo profundizar en el tema, para conocer el objeto de la investigación desde un punto de vista externo, sin que se requiera aún la profundización en la esencia del fenómeno. En un segundo momento se realizó una entrevista estructurada a partir de un cuestionario pues la información que se obtiene es mucho más fácil de procesar brindando un entendimiento más general de lo que se quiere lograr.
- El método experimental: para comprobar la funcionalidad de los resultados obtenidos a partir de la puesta en práctica de la aplicación.
- El método de la observación científica: para conocer la realidad mediante la percepción directa de los procesos descritos.

Métodos teóricos:

- El método inductivo-deductivo: para la identificación de la problemática, así como sus variantes de solución.
- El analítico-sintético: para la recolección bibliográfica sobre el proceso de control de AF con el objetivo de realizar un estudio y valoración de los mismos determinando los aspectos esenciales para arribar a conclusiones prácticas y teóricas.
- La modelación: para crear todos los modelos y diagramas que permitan un mejor entendimiento de los procesos del sistema. Destacando: el modelo conceptual, la especificación de requisitos, los diagramas de clases del diseño y el modelo de datos.

Para un mejor entendimiento de este documento, a continuación se describen brevemente cada uno de los capítulos en que se encuentra estructurado el mismo.

Capítulo 1. Enfoques investigativos sobre el proceso de entrega y recibo de activos fijos: describe los principales conceptos asociados al dominio del problema, así como las relaciones que se establecen entre ellos. Expone un análisis de las soluciones existentes en el mercado, resaltando las tendencias y tecnologías más comunes en su desarrollo. Finalmente, se argumentan las bases del desarrollo del sistema propuesto.

Capítulo 2. Propuesta de solución: refleja cada uno de los pilares del desarrollo del sistema propuesto, la especificación de sus funcionalidades, la arquitectura, los modelos de diseño y de datos, así como el uso de patrones usados para la implementación de esta investigación.

Capítulo 3. Implementación y validación de la propuesta de solución: describe la nomenclatura usada en la implementación de la solución y refiere la estrategia de pruebas definida a partir de la metodología de desarrollo, con el objetivo de garantizar la validación de la solución propuesta. Se analizan los niveles de pruebas ejecutados y los resultados alcanzados. Finalmente, se presentan evidencias de cómo la solución propuesta puede contribuir a mejorar el control del proceso de entrega y recibo de AF.

CAPÍTULO 1: ENFOQUES INVESTIGATIVOS SOBRE EL PROCESO DE ENTREGA Y RECIBO DE ACTIVOS FIJOS

Introducción

El presente capítulo brinda información sobre el estudio del arte realizado a los procesos de control de los AF, así como la descripción de soluciones existentes tanto a nivel nacional e internacional, mencionándose sus características, beneficios y soluciones asociadas al dominio del problema. Además de describir las tecnologías, herramientas y metodología a utilizar para el desarrollo de la solución.

1.1. Estudio del estado del arte

Conceptos asociados:

Activos: los activos son los bienes, derechos y otros recursos controlados económicamente por la empresa, resultantes de sucesos pasados de los que se espera obtener beneficios o rendimientos económicos en el futuro. Un activo es un bien que la empresa posee y que pueden convertirse en dinero u otros medios líquidos equivalentes (2).

Clasificación de los activos: los activos que una empresa posee se clasifican dependiendo de su liquidez, es decir, la facilidad con la que ese activo puede convertirse en dinero (2).

Por ello se dividen en (2):

- Activo fijo: son los activos utilizados en el negocio y no adquiridos con fines de venta, como maquinarias y bienes inmuebles.
- Activo circulante: son activos que se esperan que sean utilizados en un período inferior al año, como clientes o existencias.

Activo fijos: **"***Un activo fijo es un bien tangible o intangible que posee una empresa o persona natural. Por extensión, se denomina también activo al haber de una empresa. El activo forma parte de las cuentas reales o de balance"* (3).

Activo es un sistema construido con bienes y servicios, con capacidades funcionales y operativas que se mantienen durante el desarrollo de cada actividad socio-económica específica (3).

Son ejemplos de activos: bienes inmuebles, maquinaria, material de oficina, etc. También se incluyen dentro del AF las inversiones en acciones, bonos y valores emitidos por empresas afiliadas (4).

Para la investigación el concepto de activo se refiere a los AF asignados a la facultad que se consideran tangibles.

Estudio de sistemas similares:

En gran parte del mundo, casi todos los procesos económicos y de control interno se encuentran automatizados para ofrecer una mayor claridad y seguridad, así como una mejor gestión sobre dichos procesos. Muchos de estos programas se encargan de gestionar el control de los AF, por esta razón se hace necesario realizar un análisis sobre algunos de estos sistemas que poseen características similares a las requeridas en nuestro problema, con el objetivo de comprobar si es posible adaptarlo a los requerimientos de la problemática planteada.

Sistema de control de activos usados en el mundo

Sistema de Control y Administración de Activos Fijos (SCAF): su objetivo es mantener administrado de forma integrada y detallada los AF de la organización, controlando la ubicación física, valores históricos y depreciación acumuladas. De igual forma permite proporcionar información para mejorar el costo de mantenimiento de los activos. El sistema permite (5):

- Consultar con rapidez todos los datos.
- Aplicación automática y por centros de costo, del gasto de depreciación.
- Aplicación automática de la revaluación y su depreciación (también por centros de costo).
- Avisos automáticos del sistema.
- Control automático de acciones a tomar para el mantenimiento preventivo.
- Avisos automáticos sobre término de la vida útil estimada de los activos.

SISTEMA ACTIVE-AF: es un software avanzado para la administración y control de los AF. Puede ser la solución al reto de mantener bien identificados cada uno de los bienes e incluso de conocer su ubicación real y por quién es administrado dentro de la organización. Realizado en México por SWARZ Asociados S.A de C.V (6). El sistema permite**:**

- Control físico de activos:
  - Definición de múltiples atributos en cada activo para su posterior clasificación.
  - Reportes que permiten seleccionar y agrupar la información antes de imprimir.
  - Búsquedas múltiples en la base de datos.
  - Conciliación inventario físico-libros a través de etiquetado de código de barras.
- Exportación de los reportes a múltiples plataformas (Excel, Lotus, WinWord, texto, HTML)
- Depreciación por meses o días.
- Importación de catálogos desde plataformas externas (ASCII, DBASE, etc.).
- Proyecciones en valores y depreciaciones sin límite de tiempo y consultas de cálculos pasados.

SAP R/3: es un sistema informático que gestiona de manera integrada "en línea" todas las áreas funcionales de la empresa. Está organizado en un conjunto de módulos de software en el cual se incluye la gestión de los activos de la organización. El sistema permite (7)*:*

- Jerarquía de la información.
- Integración con aplicaciones y sistemas externos.
- Reduce el coste y la complejidad de las actualizaciones y el mantenimiento.
- Según consideraciones de usuarios del producto SAP R/3, para la gestión contable de las empresas, se considera como el mejor de su tipo ya que presenta como principales ventajas sobre otros sistemas mejor seguridad, apertura, reputación. Aunque reconocen que aun presenta como principal desventaja el alto precio del producto así como la complejidad que trae consigo su explotación.

Sistemas de control de activos usados en Cuba

*Versat Sarasola***:** sistema cubano de contabilidad confiable, permite enviar información eficaz, de forma inmediata, desde lugares apartados, a la vez que ofrece mayor organización, control y disciplina en cada gestión. Diseñado para su empleo en cualquier entidad empresarial o presupuestada. Permite llevar el control y registro contable individual de todos los hechos económicos que se originan en las estructuras internas de las entidades, así como exponer el estado financiero y toda la información económica y contable en este universo. Este sistema integrado cuenta con un conjunto de 12 módulos los cuales son: *Configuración y Seguridad, Contabilidad General y de Gastos, Costos y Procesos, Análisis Económico Empresarial, Control de Activos Fijos, Finanzas y Cajas, Planificación y Presupuestos, Control de Inventarios, Pago de Salario, Paquete de Gestión, Contratación, Facturación* (8).

Entre las operaciones que realiza el módulo de control de los activos se pueden citar, las de dar altas, bajas y modificación de los medios. Además de poseer un asistente para la configuración del subsistema. Este sistema permite (8):

- Realizar inventarios a partir de diferentes selecciones, determinar las posibles diferencias y contabilizar las mismas.
- Permite definir diferentes ciclos de depreciación de los AF teniendo en cuenta las características de la entidad.
- Permite el control de los activos en diferentes monedas.
- Ofrece variedad de reportes sobre las existencias y movimientos de los activos.

Rodas XXI: sistema integral económico-administrativo creado por CITMATEL[1] para la automatización de la gestión empresarial. Cuenta con seis módulos: *Contabilidad, Inventario, Activos Fijos, Nóminas, Finanzas y Facturación*. El sistema permite (9):

- Control detallado de los medios de la entidad, realizando en el mismo momento que se registra un movimiento su contabilización.
- Realiza todo tipo de operaciones de AF con facilidad en el momento que se desee, generando el documento asociado al movimiento de que se trate de forma automática.
- Cuenta con una gran variedad de opciones de informes, entre ellos el submayor de AF, listados y localización de los medios de transporte de la entidad, la depreciación mensual y acumulada de uno o de varios AF.
- Permite visualizar información correspondiente a períodos anteriores ya cerrados.

Sistema de control de activos utilizado en la UCI

Sistema de Gestión Integral ASSETS: este es un sistema comercializado por la firma panameña D´MARCO S.A. y distribuido en Cuba por INFOMASTER, entidad informática perteneciente a la Empresa Nacional de Producción y Servicios a la Educación Superior del MES[2] (10).

ASSETS, es un sistema de gestión integral estándar y parametrizado que permite controlar, realizar y contabilizar todas las transacciones comunes relacionadas con el proceso de compra-venta, e incluye así mismo los procedimientos necesarios para registrar los movimientos de los AF y de los útiles y herramientas. El sistema permite (10):

- Controlar, por centros de costos, los medios básicos. Permite definir además, la ubicación física de los mismos.
- Controlar los procesos de compras, alquiler, altas, bajas, préstamo y traspasos hacia otras áreas dentro y fuera de la entidad.
- Realiza reparaciones, realizar ajustes y controlar la depreciación acumulada de cada activo. Permite además realizar revalorizaciones de los activos y emitir, al cierre del mes, el comprobante de operaciones por la depreciación de los activos, cada movimiento genera automáticamente el asiento contable correspondiente.

[1] Empresa de Tecnologías de la información y servicios telemáticos.
[2] Ministerio de Educación Superior.

Después de realizar un estudio de los sistemas de gestión mencionados anteriormente, los cuales incluyen dentro de sus funcionalidades el control de AF; describiéndose sus principales características. Se concluyó que los mismos a pesar de contar con elementos positivos para su implementación, no satisfacen las necesidades del proceso que se desea desarrollar, pues estas aplicaciones tienen una concepción económica para la gestión de los activos y la situación presente está objetivamente enfocada a gestionar el proceso del control de entrega y recibo de los AF y del personal que realiza la entrega.

Por otra parte, todos ellos son software propietarios, lo que constituye un aspecto negativo para el país y para la UCI, pues estos se encuentran inmersos en un proceso de independencia tecnológica y traerían consigo gastos innecesarios por la obtención de las licencias. Además, no todos permiten la adaptación de los procesos a las características de la entidad en la cual se centra la investigación y no cuentan con mecanismo de integración con otras aplicaciones. Para una mejor comprensión, se muestra una comparación de estos sistemas en la siguiente tabla.

Tabla 1. Comparación entre los sistemas estudiados. Fuente: elaboración propia.

Sistemas usados en los distintos ámbitos		**Base de datos compatibles**	**Sistemas Operativos compatibles**	**Tipo de aplicación**	**Licencia comercial**
Internacionales	Sistema de Control y Administración de Activos Fijos (SCAF)	Microsoft SQL Server	Microsoft Windows	Cliente-servidor	Privativa
	ACTIVE-AF	Microsoft SQL Server	Windows 95, 98, NT, 2000 o XP	Cliente-servidor	Privativa
	SAP R/3	Informix, Oracle, Adabas, Sybase ASE, IBM DB/2, Microsoft SQL Server.	UNIX, Open MS, Microsoft Windows Server	Cliente-servidor	Privativa
Nacionales	Versat_Sarasola	Microsoft SQL Server 2000	Microsoft Windows	Aplicación de escritorio	Privativa

	Rodas XXI	Microsoft SQL Server 2000	Microsoft Windows XP, 2000 o 2003	Cliente-servidor	Privativa
UCI	Sistema de Gestión Integral ASSETS	Microsoft SQL Server 2000	Microsoft Windows'98 o superior	Cliente-servidor	Privativa

La UCI propone para el desarrollo de sus proyectos de software la metodología AUP-UCI, para el presente trabajo se hará uso de esta en su versión 1.2.

1.2. Metodología de desarrollo de software

El Proceso Unificado Ágil de Scott Ambler o Agile Unified Process (AUP por sus siglas en inglés), en unión con el modelo CMMI-DEV v1.3 la cual es una versión simplificada del Proceso Unificado de Desarrollo (RUP por sus siglas en inglés) (11).

Esta metodología es una manera simple y fácil de entender la forma de desarrollar aplicaciones de software de negocio usando técnicas ágiles y conceptos que aún se mantienen válidos en RUP. El AUP tiene entre sus principales características que está dirigida por pruebas, cuenta con un modelado ágil, incluye la gestión de cambios ágiles y la refactorización de base de datos para mejorar la productividad (11).

Las fases con que cuenta el AUP son (11):

1. Inicio: el objetivo de esta fase es obtener una comprensión común cliente-equipo de desarrollo del alcance del nuevo sistema y definir una o varias arquitecturas candidatas para el mismo.
2. Elaboración: el objetivo es que el equipo de desarrollo profundice en la comprensión de los requisitos del sistema y en validar la arquitectura.
3. Construcción: durante la fase de construcción el sistema es desarrollado y probado al completo en el ambiente de desarrollo.
4. Transición: el sistema se lleva a los entornos de pre-producción donde se somete a pruebas de validación, aceptación y finalmente se despliega en los sistemas de producción.

Variación AUP 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." (11).

La cual estará apoyada 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 (11).

Descripción de las Fases AUP-UCI

"De las 4 fases que propone AUP (Inicio, Elaboración, Construcción, Transición) se decide para el ciclo de vida de los proyectos de la UCI mantener la fase de Inicio, pero modificando el objetivo de la misma, se unifican las restantes 3 fases de AUP en una sola, a la que llamaremos Ejecución y se agrega una fase de Cierre." (11).

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 8 disciplinas, pero a un nivel más atómico que el definido en AUP. Para una mejor compresión de las disciplinas se muestra la siguiente Tabla (11).

Tabla 2. Disciplinas de la metodología AUP-UCI. Fuente: (11).

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

Disciplinas Variación AUP-UCI	Objetivos Disciplinas (Variación AUP-UCI)
	diseños de casos de prueba, listas de chequeo y de ser posibles componentes de prueba ejecutables para automatizar las pruebas.
Pruebas de liberación	Pruebas diseñadas y ejecutadas por una entidad certificadora de la calidad externa, a todos los entregables de los proyectos antes de ser entregados al cliente para su aceptación.
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.
Despliegue (Opcional)	Constituye la instalación, configuración, adecuación, puesta en marcha de soluciones informáticas y entrenamiento al personal del cliente.

Escenarios para la disciplina de requisitos:

La metodología propone cuatro escenarios para la disciplina de requisitos. El presente trabajo estará regido por el *escenario No 4,* su selección está basada en que dentro de sus características cumple: que no es un proyecto extenso y el cliente estará siempre acompañando al equipo de desarrollo para definir en conjunto los detalles de los requisitos y así poder implementarlos, probarlos y validarlos.

Al ser identificada la metodología a utilizar, así como las fases y disciplinas que rigen la misma, se hace necesario describir las herramientas y las principales tecnologías a utilizar en el proceso de desarrollo del software.

1.3. Lenguaje y herramienta de modelado

Lenguaje unificado de modelado 2.1 (Unified Modeling Language, UML 2.1): para el modelado de los proceso de negocio de la presente investigación fue aplicado el lenguaje de modelado de sistemas de software *UML*. Ya que constituye un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema de software. *UML* ofrece un estándar para describir un plano del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocios, funciones del sistema y aspectos concretos (12).

Herramienta de modelado: para el modelado de los procesos de esta investigación se hace uso del *Visual Paradigm-UML 8.0*. Esta herramienta fue creada para utilizarse tanto en el entorno de Windows como en el entorno de Linux, facilita la realización de los diagramas de modelado como los diagramas de clases y código inverso además de que soporta el ciclo de vida completo del desarrollo de software (13).

1.4. Arquitectura de software

La arquitectura de software permite comprender y mejorar la estructura de las aplicaciones, además, permite la corrección de estas y su grado de cumplimiento respecto a los requisitos inciales (14). Haciendo uso para ellos de patrones, los cuales serán descritos a continuación.

Patrones de arquitectura: expresan un esquema fundamental de organización estructural para sistemas de software. Proveen subsistemas predefinidos, especificando sus responsabilidades, e incluyen reglas y guías para organizar las relaciones entre ellos (12). A continuación se describe el patrón arquitectónico utilizado en esta investigación.

Arquitectura Modelo Vista Controlador (MVC): este patrón arquitectónico separa en tres capas diferentes los datos de aplicación, la interfaz de usuario, y la lógica de control.

- El modelo administra el comportamiento y la información del negocio, de la misma forma que responde sobre acciones que se decidan efectuar sobre este.
- La vista, es la encargada de administrar la presentación de la información.
- El controlador es el componente intermediario entre el modelo y la vista para lograr un desacoplamiento total entre ambos. Se utiliza en la implementación de los módulos del sistema a desarrollar.

El marco de trabajo Symfony2 respeta este principio, creando sobre el mismo algunas variaciones, las cuales están ligadas a criterios de seguridad. El principio básico relacional entre los tres elementos (Modelo, Vista, Controlador) sigue siendo el mismo, de esta manera nunca existirá una interacción directa entre el modelo y la vista, sin antes haber pasado por el controlador, el cual gestiona toda la lógica de negocio (15).

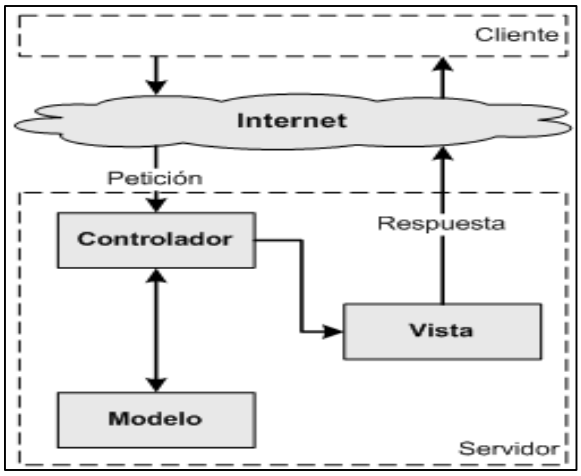


Ilustración 1. Representación de la arquitectura MVC en Symfony2. Fuente: (15).

Patrones de diseño: un patrón de diseño define un esquema de refinamiento de los subsistemas o componentes dentro de un sistema, o las relaciones entre estos. Este describe una estructura común y recurrente de componentes interrelacionados, que resuelve un problema general de diseño dentro de un contexto particular (14); dividiéndose en dos grandes grupos: patrones GRASP y GoF.

Patrones GRASP: son patrones generales de software para asignar responsabilidades a objetos (12). Los utilizados son:

- *Controlador*: consiste en asignar la responsabilidad a una clase de manejar los mensajes correspondientes a eventos de un sistema. El controlador es un intermediario entre la interfaz de usuario y el núcleo donde reside la lógica de la aplicación. Este patrón sugiere que la lógica de negocios debe estar separada de la capa de presentación para aumentar la reutilización del código (16).
- *Experto*: este patrón consiste en asignar una responsabilidad al experto en información, es decir, a la clase que cuenta con la información necesaria para cumplir con la responsabilidad. Lo que permite reforzar el encapsulamiento y favorecer el bajo acoplamiento (16).
- *Creador***:** es el responsable de la creación o instanciación de nuevos objetos o clases (16). Éste patrón refleja que la nueva instancia podrá ser creada por una clase si (16):
    - Clase B contiene o agrega la clase A.
    - Clase B es una agregación o composición de la clase A.
    - Almacena la instancia en algún sitio (por ejemplo una base de datos).
    - Tiene la información necesaria para realizar la creación del objeto (es 'Experta').

- Usa directamente las instancias creadas del objeto.

- *Alta cohesión*: su objetivo es asignar una responsabilidad, de modo que la cohesión siga siendo alta. Las clases con alta cohesión se caracterizan por tener responsabilidades estrechamente relacionadas y no realizar un trabajo enorme. Una clase con alta cohesión es útil porque es bastante fácil darle mantenimiento, entenderla y reutilizarla. La utilización de este patrón se evidencia en las responsabilidades asignadas a las clases de tal manera que cada una se encargará solamente de realizar las funciones que estén en correspondencia con la responsabilidad que posea (16).
- *Bajo acoplamiento*: pretende asignar una responsabilidad para mantener el bajo acoplamiento, es decir, el diseño de clases más independientes que no se relacionen con muchas otras y reduzcan el impacto de los cambios, siendo más reutilizables y acrecienten la oportunidad de una mayor productividad (16).

Patrones GoF: a principios de los años 90 con la publicación del libro *Design Patterns*, se establecen veintitrés patrones de diseño GoF. Los patrones GoF se descubren como una forma indispensable de enfrentarse a la programación a raíz del libro “Design Patterns-Elements of Reusable Software” de Erich Gamma, Richard Helm, Ralph Jonson y John Vlissides, a partir de entonces estos patrones son conocidos como los patrones de la pandilla de los cuatro (GoF, gang of four) (17).
Los patrones GOF según su propósito se clasifican en tres grupos. Los creacionales tratan la creación de instancias. Los estructurales tratan la relación entre clases, la combinación clases y la formación de estructuras de mayor complejidad. Mientras que los de comportamiento tratan la interacción y cooperación entre clases. Los patrones a aplicar serán:

- *Decorador (Decorator):* este patrón permite modificar, retirar o agregar responsabilidades a un objeto dinámicamente, o sea, permite agregarle funcionalidades a los objetos durante su ejecución. Esto trae consigo que no sea necesario la creación de clases compleja con mucho código, pues podemos usar distintas combinaciones de decoraciones para generar distintos comportamientos o resultados (18).
- *Inyección de dependencia (Dependency injection):* es un patrón que sugiere "inyectar" objetos a una clase, en vez de que la clase cree el objeto por sí solo. Esto permite que la clase que utilice el objeto no requiera especificar como crearlo, sino que otra clase se ocupará en crear este objeto según sus especificaciones y simplemente será agregado en la clase en cuestión (18).

1.5. Ambiente de desarrollo

Lenguajes de programación: el lenguaje del lado del servidor a usar en el desarrollo web es *PHP 5.5 (Hypertext Pre-processor).* PHP es un lenguaje gratuito e independiente de plataforma, rápido, con una gran librería de funciones y mucha documentación, multiplataforma, sumamente flexible. Utiliza el paradigma orientado a objeto y en la actualidad es muy utilizado, ya que permite el desarrollo de programas seguros y de alto rendimiento (19).

En cuanto a los lenguajes de programación usados del lado del cliente para el desarrollo de la investigación, se encuentran: *JavaScript,* porque es compatible con la mayoría de los navegadores modernos, permite crear efectos especiales en las páginas web y definir interactividades con el usuario. Es un lenguaje bastante sencillo, rápido y altamente modular. Por otro lado es utilizado *HTML 5 (Hyper Text Markup Language),* por ser un lenguaje sencillo que permite definir y ubicar los distintos elementos que componen una página web. Puede integrarse con lenguajes como JavaScript y PHP. *Las Hojas de Estilo en Cascada o CSS (Cascading Style Sheet)* constituyen un estándar que permite establecer el estilo de documentos; permitiendo definir el aspecto visual de todas las páginas de la solución (20).

Luego de definir los lenguajes a utilizar se seleccionan los marcos de trabajo los cuales definen una serie de buenas prácticas aplicables a un proceso de codificación.

Marco de trabajo (Framework): un marco de trabajo en el desarrollo de software es una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado. Los que a continuación se definen son los propuestos para el desarrollo de la solución.

- *Symfony2:* es un proyecto *PHP* de software libre que permite crear aplicaciones y sitios web rápidos y seguros de forma profesional, su código y el de todos los componentes y librerías que incluye se publican bajo la licencia *MIT*[3] de software libre (16).

  Este marco de trabajo permite exprimir al límite todas las nuevas características de PHP 5 y por eso es uno de los frameworks PHP con mejor rendimiento. Su arquitectura interna está completamente desacoplada, posibilitando reemplazar o eliminar fácilmente aquellas partes que no encajan en el desarrollo de un proyecto determinado. Siendo este independiente del sistema gestor de bases de datos, fácil de instalar y configurar en la mayoría de plataformas (16). Para el desarrollo del sistema se utilizará la versión Symfony 2.6, la cual es compatible con la versión de *PHP 5.5*.

[3] *Massachusetts* Institute of Technology.

- *Bootstrap:* permite crear interfaces web con *CSS* y *JavaScript* que adaptan la interfaz dependiendo del tamaño del dispositivo en el que se visualice de forma nativa, es decir, automáticamente se adapta al tamaño de un ordenador o de una tablet sin que el usuario tenga que hacer nada. Incluye varios estilos predefinidos fáciles de configurar: botones, menús desplegables, entre otros (21). Por lo antes expuesto se propone utilziar la versión *Bootstrap 3.1*, la cual permite el uso de interfaces capaces de adaptarse automaticamente al tamaño de cualquier dispositivo.

Herramientas de desarrollo propuestas para la confección de la solución:

- *NetBeans IDE 8.0*: IDE de código abierto y multiplataforma, emplea plugins para proporcionar todas sus funcionalidades. La arquitectura de plugins permite integrar diversos lenguajes, introducir otras aplicaciones accesorias que pueden resultar útiles durante el proceso de desarrollo (22).
- *PostgresSQL 9.3:* es un gestor de base de datos de código abierto, siendo uno de los primeros en implementar un motor de base de datos relacional. Dicho gestor maneja una serie de características y funcionalidades que permiten un trabajo flexible sobre los datos, como son la alta concurrencia, claves ajenas y disparadores (23).
- *Gitlab1.6:* sistema distribuido de control de versiones basado en software libre en el cual cada copia es un repositorio completo que no depende de un repositorio central para su funcionamiento. Gitlab es una herramienta rápida y eficiente, que permite llevar cuenta de los cambios de un conjunto de archivos digitales en el tiempo. Es muy utilizado para la gestión de código fuente permitiendo a un grupo de programadores gestionar el código de un mismo proyecto. Además, se especializa generalmente en archivos de texto (clases, documentos, etc.) (24).
- *Apache 2.2.22:* servidor web de código abierto, para plataformas Unix, Microsoft Windows, Macintosh y otras. Apache es altamente configurables, tiene bases de datos de autenticación y negociado de contenido. Entre las ventajas que presenta el mismo se pueden citar que es modular, de código abierto, multi-plataforma y extensible. Es usado principalmente para lanzar páginas web estáticas y dinámica a la World Wide Web (Gran Telaraña Mundial) (25).

Todas las herramientas descritas con anterioridad son propuestas para su utilización por cumplir las condiciones de ser software de código libre y contar con las versiones más actualizadas en relación al momento en que se desarrolla esta investigación.

Conclusiones del capítulo

- A partir de los estudios realizados en este capítulo se obtuvo el diseño teórico de la investigación quedando claramente expuestos los principales conceptos relacionados con el tema, así como las herramientas y tecnologías propuestas para utilizar.
- Las aplicaciones nacionales e internacionales estudiadas para el control de los AF, no satisfacen las principales necesidades de la institución (control del proceso de entrega y recibo de AF, control sobre los reportes e incidencias generados, control sobre la personal encargado de realizar la entrega), por lo que se decide implementar una nueva aplicación.
- Durante el análisis del estudio de las herramientas y tecnologías para el desarrollo de la investigación se concluyó que las más convenientes a utilizar en la investigación son: la metodología *AUP-UCI v1.2* para guiar todo el proceso de desarrollo de software, el *Visual Paradigm 8.0*, para modelar todos los diagramas y artefactos propuestos por la metodología, el marco de trabajo *Symfony2* por su alta modularidad, el *NetBeans IDE* en su versión 8.0 para la implementación del código, y el gestor de base de datos *PostgresSQL 9.3*.

CAPÍTULO 2: PROPUESTA DE SOLUCIÓN

Introducción

En este capítulo se describen los artefactos de trabajo propuestos por AUP-UCI para el desarrollo de la solución propuesta en las fases de: requisitos, análisis y diseño. En la etapa de requisitos se identifican y describen los requisitos funcionales, no funcionales e historias de usuario. En el análisis y el diseño, se detallan los principales artefactos de dicha disciplina, destacando el modelo de diseño, los diagramas de clases, así como el modelo de datos. Igualmente, se especifica la utilización de los patrones de arquitectura y de diseño para la concepción de la solución.

2.1. Requisitos

El propósito de esta disciplina es hacer que los requerimientos alcancen un estado óptimo antes de desarrollar la fase de diseño en el proyecto, definiendo, las características de un sistema que satisfaga las necesidades de negocio de clientes y que se integre con éxito en el entorno en el que se explote. Además de gestionar las líneas base y las peticiones de cambios que se produzcan en la especificación de requisitos, manteniendo la trazabilidad entre los requisitos y otros productos del desarrollo (17).
Para un mejor entendimiento de como se encuentran estructurados los procesos dentro de la insitución, a continuación se describe la realización del modelo conceptual.

Modelo Conceptual

Un modelo conceptual explica los conceptos significativos en un dominio del problema, para entender los principales términos asociados a los procesos internos de una entidad, la siguiente imagen muestra los conceptos fundamentales relacionados con el desarrollo de esta investigación.

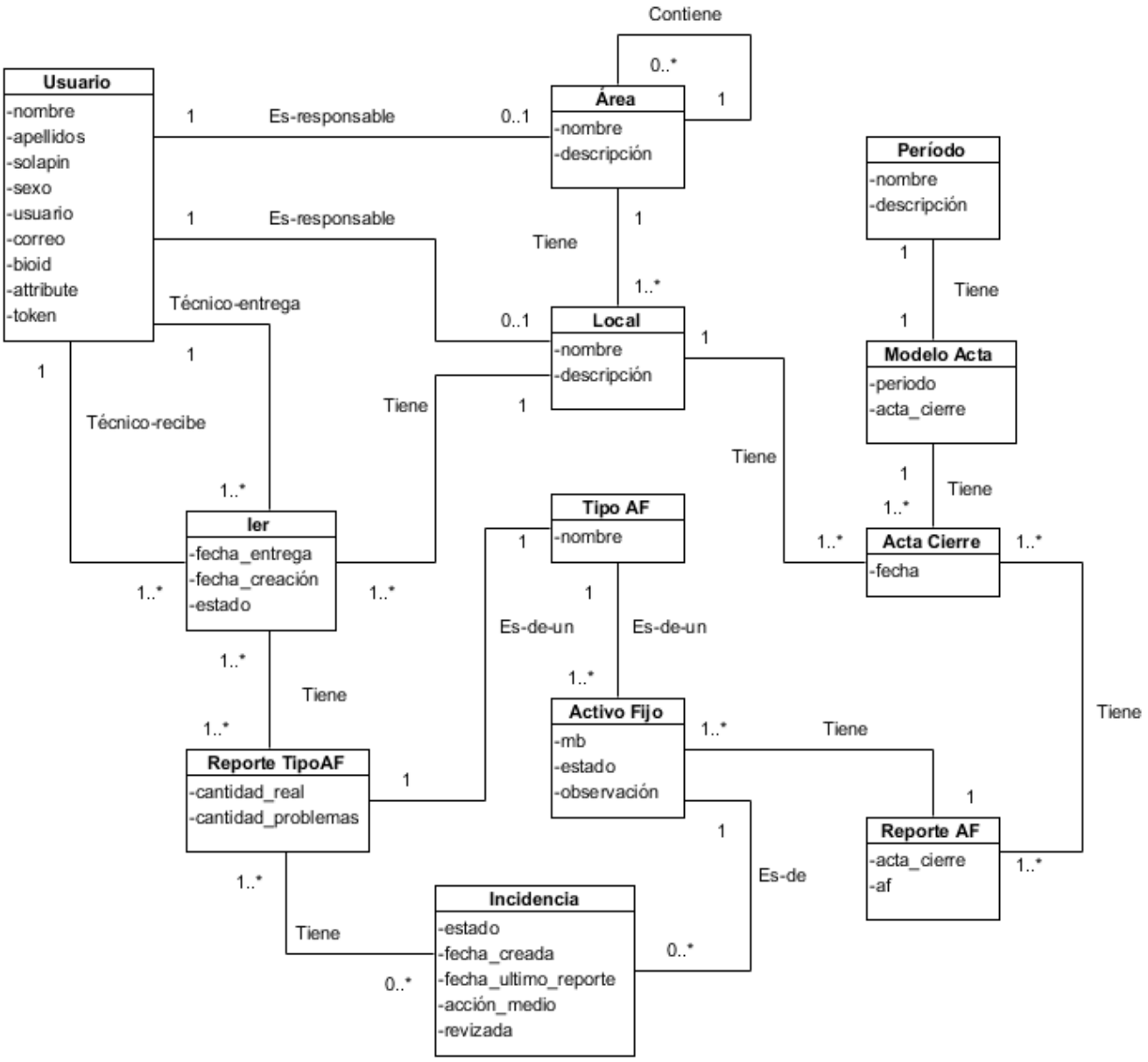


Ilustración 2. Modelo conceptual del negocio. Fuente: elaboración propia.

Técnicas para la captura de requisitos funcionales

La captura de requisitos es la actividad mediante la cual, el equipo de desarrollo de un sistema de software, extrae de cualquier fuente de información las necesidades que debe cubrir dicho sistema. Las técnicas utilizadas fueron:

- *Entrevista*: la misma se realizó a las personas encargadas de controlar la entrega y recibo de los activos con el objetivo de precisar el problema a resolver e identificar su modo de funcionamiento. Esta entrevista se le realizó al Ingeniero Zénel Pérez Reyes, jefe del

Departamento de Tecnología y a la máster Ana Marys García Rodríguez, vicedecana de Administración de la Facultad 3 (Consultar Anexo 1), a partir de un conjunto de preguntas, pues la información que se obtiene es mucho más fácil de procesar brindando un entendimiento más general de lo que se quiere lograr.

Las entrevistas realizadas arrojaron información valiosa acerca de las necesidades y perspectivas de utilización de la solución, resaltando la importancia de desarrollar una aplicación sencilla, amigable y fácil de utilizar. El uso de esta técnica permitió determinar: la necesidad de restringir el acceso a la información, brindar información detallada sobre todas las incidencias reportadas en los distintos locales, permitiendo contar con un historial que posibilite acceder de una manera rápida a toda la información registrada en el sistema para los distintos locales que cuentan con AF. Además, los especialistas plantearon la necesidad de que la solución contara con nuevas funcionalidades, entre ellas:

- un mecanismo de autenticación que permita garantizar la presencia física del responsable en el área y el no repudio de los informes generados.
- un historial de las incidencias reportadas conociendo su estado en todo momento.
- un listado de los medios que más incidencias reportadas han tenido, así como los locales con más incidencias pendientes.

- *Análisis de soluciones existentes:* esta técnica consiste en analizar diferentes sistemas ya desarrollados que posean características semejantes con el software que se desea construir. Los diferentes sistemas estudiados para el proceso de control de entrega y recibo de AF proponen funcionalidades las cuales fueron parte de la propuesta de solución como por ejemplo:
  - controlar físicamente los activos por local.
  - obtener reportes sobre los activos.
  - realizar operaciones sobre los medios (alta, baja y modificación).

  Los resultados obtenidos se describen en el epígrafe *1.1 Estudio del estado del arte,* en el cual se analizan los distintos sistemas estudiados.

A continuación se definen los requisitos funcionales identificados luego de aplicar las técnicas para la captura de los mismos.

Definición de los requisitos funcionales

Los requisitos funcionales de un sistema describen lo que el sistema debe hacer. Estos requerimientos dependen del tipo de software que se desarrolle, de los posibles usuarios del software y del enfoque

general tomado por la organización al redactarlos. Son declaraciones de los servicios que debe proporcionar el sistema, de la manera en que este debe reaccionar a entradas particulares y de como este se debe comportar en situaciones particulares (26).

La siguiente tabla muestra el listado de los requisitos funcionales identificados agrupados mediante la utilización de los patrones CRUD[4] total y parcial, ya que existen requisitos agrupados según las acciones del patrón y en otros casos solamente agrupados por algunas de ellas.

Tabla 3. Requisitos funcionales del software. Fuente: elaboración propia.

No	RF	Requisito funcional	Complejidad
	RF 01	**Gestionar AF**	
1	RF 1.1	Adicionar AF	Alta
2	RF 1.2	Eliminar AF	Media
3	RF 1.3	Buscar AF	Alta
4	RF 1.4	Modificar AF	Media
	RF 02	**Gestionar Local**	
5	RF 2.1	Adicionar local	Media
6	RF 2.2	Eliminar local	Media
7	RF 2.3	Buscar local	Alta
8	RF 2.4	Modificar local	Media
	RF 03	**Gestionar Área**	
9	RF 3.1	Adicionar área	Media
10	RF 3.2	Eliminar área	Media
11	RF 3.3	Buscar área	Alta
12	RF 3.4	Modificar área	Media
	RF 04	**Gestionar Tipo AF**	
13	RF 4.1	Adicionar tipo AF	Media
14	RF 4.2	Eliminar tipo AF	Media
15	RF 4.3	Buscar tipo AF	Media
16	RF 4.4	Modificar tipo AF	Alta
	RF 05	**Gestionar Usuario**	
17	RF 5.1	Registrar usuario	Bajo
18	RF 5.2	Eliminar usuario	Bajo
19	RF 5.3	Buscar usuario	Media
20	RF 5.4	Autenticar usuario	Alta
21	**RF 06**	Generar token	Alta
22	**RF 07**	Asignar token a usuario	Media
23	**RF 08**	Asignar rol a usuario	Baja
24	**RF 09**	Registrar vector	Alta

[4] Create-Read-Update-Delete.

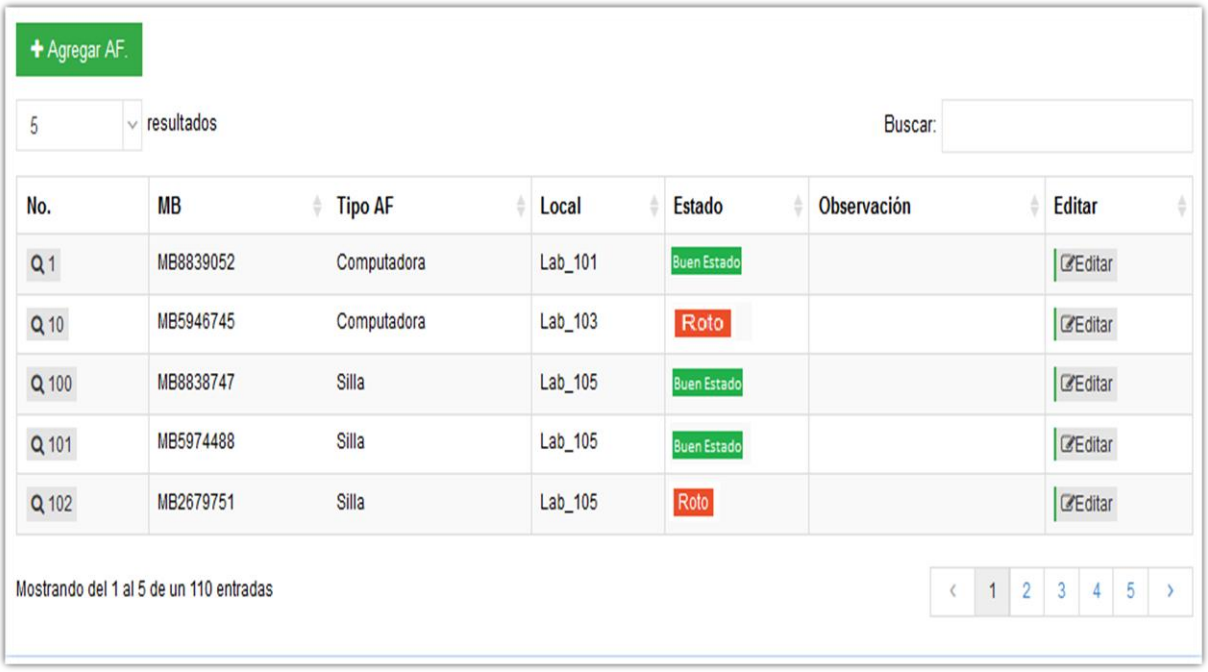
No	RF	Requisito funcional	Complejidad
25	**RF 10**	Asignar vector a usuario	Alta
	RF 11	**Gestionar Reporte Tipo AF**	
26	RF 11.1	Adicionar reporte tipo AF	Media
27	RF 11.2	Eliminar reporte tipo AF	Media
28	RF 11.3	Buscar reporte tipo AF	Baja
29	RF 11.4	Modificar reporte tipo AF	Alta
30	RF 11.5	Listar reportes de tipo de AF	Alta
	RF 12	**Gestionar Incidencia**	
31	RF 12.1	Adicionar incidencia	Media
32	RF 12.3	Buscar incidencia	Baja
33	RF 12.4	Modificar incidencia	Alta
	RF 13	**Gestionar Modelo Acta**	
34	RF 13.1	Adicionar modelo acta	Alta
35	RF 13.3	Buscar modelo acta	Baja
36	RF 13.4	Modificar modelo acta	Media
	RF 14	**Gestionar Acta Cierre**	
37	RF 14.1	Adicionar acta cierre	Alta
38	RF 14.3	Buscar acta cierre	Baja
39	RF 14.4	Modificar acta cierre	Baja
40	**RF 15**	Listar los reporte de incidencia	Alta
41	**RF 16**	Listar los reportes de IER por local	Alta
42	**RF17**	Listar reportes de IER por fecha	Alta
43	**RF 18**	Exportar el IER a PDF	Alta
	RF 19	**Gestionar Informe de Entrega y Recibo (IER)**	
44	RF 19.1	Adicionar IER	Alta
45	RF 19.4	Modificar IER	Alta
46	RF 19.3	Buscar IER	Media
47	**RF 20**	Asignar local a área	Baja
48	**RF 21**	Asignar AF a local	Media
49	**RF 22**	Listar reportes de AF por área	Alta
50	**RF 23**	Listar reportes de AF por local	Alta
51	**RF 24**	Comprobar token de seguridad	Alta
52	**RF 25**	Listar AF con más incidencias reportadas	Alta
53	**RF 26**	Listar locales con más incidencias pendientes	Alta
54	**RF 27**	Listar todas las incidencias pendientes	Media
55	**RF 28**	Activar mecanismo de seguridad	Alta

Historias de Usuario

Las Historias de Usuario son un enfoque de requerimientos ágiles que se focaliza en establecer conversaciones acerca de las necesidades de los clientes. Son descripciones cortas y simples de las funcionalidades del sistema, narradas desde la perspectiva de la persona que desea dicha

funcionalidad, usualmente un usuario (27). A continuación se presentan dos de las historias de usuario del sistema, el resto se puede encontrar en el Anexo 2.

Tabla 4. Historia de Usuario de los escenarios: Adicionar AF y Listar Reportes de IER por local. Fuente: elaboración propia.

<table>
<tr><th>No.</th><th>Nombre</th><th>Descripción</th><th>Complejidad</th></tr>
<tr><td rowspan="2">RF 1.1</td><td>Adicionar AF</td><td>Luego de autenticarse en el sistema, un usuario con los privilegios necesario para acceder a esta funcionalidad, podrá registrar un nuevo AF. El usuario inicialmente debe especificar el número del medio básico (MB), luego selecciona el tipo de AF al que pertenece dicho medio, seleccionando también el estado y el local al cual va hacer asignado el AF. Además el requisito permite realizar una búsqueda por MB sobre la tabla.</td><td>Alta</td></tr>
<tr><td colspan="3">Prototipo de interfaz
 
</td></tr>
<tr><th>No.</th><th>Nombre</th><th>Descripción</th><th>Complejidad</th></tr>
<tr><td rowspan="2">RF 16</td><td>Listar Reportes de IER por local</td><td>Luego de autenticarse en el sistema, el administrador podrá consultar todos los IER creados para un local determinado. Conociendo detalladamente el listado de todos los reportes generados sobre los medios presentes en el local, permitiendo también realizar una búsqueda filtrando por fecha de realización del IER.</td><td>Alta</td></tr>
<tr><td colspan="3">Prototipo de interfaz</td></tr>
</table>

No.	Nombre	Descripción	Complejidad
			

local	Fecha Creada	Estado informe de guardia	fecha revision	tecnico entrega	tecnico recibe	responsable	lista de reportes
Lab 304	12/02/2015	aprobado	12/02/2015	amsaborit	jjaleman	zenel	Ver Reportes
Lab 304	13/02/2015	aprobado	13/02/2015	jjaleman	pgrillo	zenel	Ver Reportes
Lab 304	14/02/2015	aprobado	14/02/2015	pgrillo	amsaborit	pamoreno	Ver Reportes
Lab 304	15/02/2015	erroneo	16/02/2015	amsaborit	jjaleman	trola	Ver Reportes
Lab 304	17/02/2015	pendiente	17/02/2015	jjaleman	psantiagovalera	trolans	Ver Reportes
Lab 304	22/02/2015	erroneo	26/02/2015	amsaborit	jjaleman	trola	Ver Reportes

Descripción de los requisitos no funcionales

Los requisitos no funcionales (RNF) describen aspectos del sistema que son visibles por el usuario que no incluyen una relación directa con el comportamiento funcional del sistema; incluyen restricciones como el tiempo de respuesta, la precisión, recursos consumidos y seguridad (26).

A continuación se muestran los requisitos no funcionales identificados basados en la norma ISO 9126:

Tabla 5. Requisitos no funcionales del software. Fuente: elaboración propia.

Requisitos no funcionales	
RNF	**Usabilidad**
1	Amigable al usuario: el sistema debe presentar una interfaz amigable que permita la fácil interacción entre el usuario y el sistema, permitiéndole acceder de manera rápida y entendible a todas las funcionalidades presentes, además de posibilitar una fácil adaptación a usuarios sin experiencias.
2	Conformidad: el sistema debe lograr un estado positivo en todos los actores que interactúen con las funcionalidades del sistema.
3	Instructibilidad: el sistema debe contar con una descripción mostrando un ejemplo en los campos que debe llenar el usuario.
4	Comprensibilidad: el sistema debe mostrar facilidad para interactuar y entender las actividades que realiza el usuario.

Requisitos no funcionales	
5	Utilizabilidad: el sistema debe contar con un menú que le permite acceder a todas las funcionalidades que realiza el sistema.
6	El sistema debe visualizar el nombre del usuario que está autenticado.
Portabilidad	
7	Instalabilidad: el sistema debe instalarse fácilmente, solo bastará con contar con un servidor web donde montar la aplicación y luego los usuarios pueden acceder desde cualquier lugar que se encuentre visible dicho servidor.
8	El sistema se desarrollará con tecnología PHP 5.5.
9	Se empleará como Gestor de Base de Datos, PostgreSQL 9.3.
10	Se empleará como Servidor de Aplicaciones Web, Apache en su versión 2.2.22.
11	La aplicación se desarrolla en la plataforma Linux, en su versión Linux Mint 17.
12	Para el acceso de los clientes solo basta tener una computadora con navegador instalado (Firefox, Internet Explorer, etc.)
13	Portabilidad: el sistema debe ser capaz de funcionar en el entorno donde sea desplegado.
14	Reemplazabilidad: el sistema debe permitir adaptar nuevos módulos o modificar los existentes según los requisitos del cliente.
Eficiencia	
15	Tiempo de respuesta: las consultas a la base de datos no deberán exceder de 1 segundo como tiempo de respuesta.
Funcionabilidad	
16	Idoneidad: el sistema está enfocado para la gestión del proceso de control de entrega y recibo de AF en la facultad 3, así como facilitar la gestión de la información de los reportes generados y la autenticidad y no repudio de los informes generados.
17	Interoperabilidad: el sistema debe permitir la integración con otras aplicaciones mediante la utilización de servicios, como son la autenticación mediante LDAP y el servicio de autenticación biométrica.
Confiabilidad	
18	Fiabilidad: el tiempo máximo de inactividad del sistema es de 15 minutos, al cabo de ese tiempo el usuario deberá autenticarse nuevamente.
19	Fiabilidad: el sistema debe contar con campos obligatorios para garantizar un manejo adecuado de la información introducida por el usuario.
20	Fiabilidad: el sistema no permite la entrada de datos incorrectos.
Mantenibilidad	
21	Analizabilidad: el sistema debe ser fácil a la hora de analizar el código fuente pues el mismo esta comentado y la documentación están redactada en un lenguaje fácil de entender.
22	Cambiabilidad: el sistema permite cambios en sus módulos sin verse afectado su funcionamiento.
Requerimientos de Software y Hardware	
23	Requerimientos del servidor: servidor web Apache v2.2.22. PHP v5.5 o superior, PostgreSQL v9.3 o superior. Procesador Core 2 Duo 2.0 GHZ o superior. RAM 256 MB (512 MB recomendada).
24	Requerimiento de las PC cliente: navegador Web, Internet Explorer 8 o Firefox v32 o superior.

Requisitos no funcionales	
Seguridad	
25	El sistema mostrará las funcionalidades de acuerdo a quien esté autenticado en el mismo.
26	El sistema debe cumplir con los principios básicos de seguridad de cualquier sistema informático, manteniendo la integridad, confidencialidad y disponibilidad de la información.

Validación de los requisitos funcionales

Esta actividad se realizó con el objetivo de garantizar que los requisitos fueran correctos y cumplieran con las necesidades del cliente. Obteniendo en todo momento la conformidad del mismo, la misma se realizó mediante las técnicas:

Validación de requisitos mediante prototipos de interfaz de usuario: se presentaron los prototipos elaborados a los responsables, para corroborar que responden a las necesidades y aspiraciones identificadas en la obtención de los requisitos. Para ello, se desarrollaron varios escenarios posibles con el auxilio de juegos de datos, de forma tal que se visualizaron las diferentes funcionalidades que tendría el sistema. Teniendo como resultado que todos los requisitos identificados fueron aceptados. A continuación se muestra un ejemplo de los prototipos utilizados.

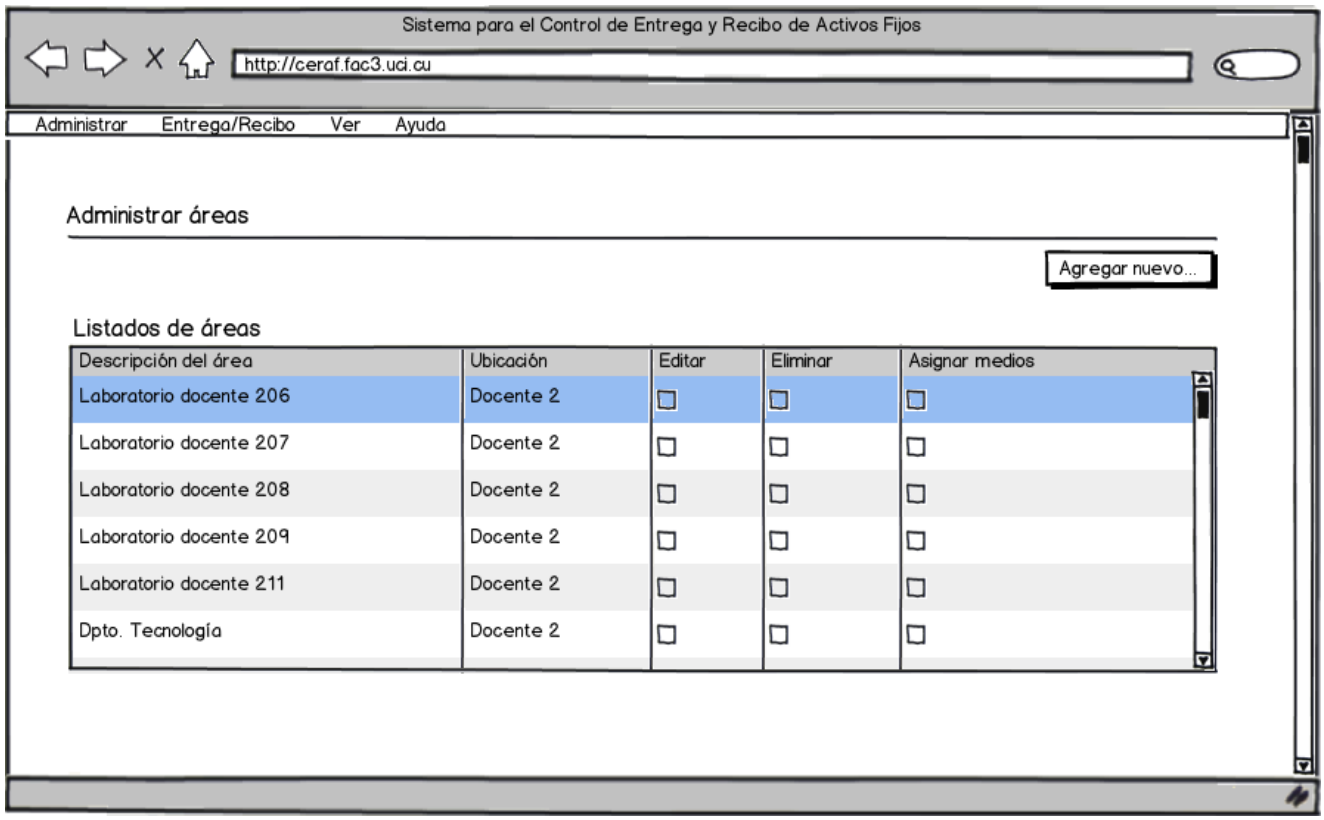


Ilustración 3. Prototipo para la captura del requisito Administrar área. Fuente: elaboración propia.

Validación de requisitos mediante caso de pruebas: se crearon un conjunto de casos de prueba para determinar si los requisitos identificados son completamente satisfactorios, su ejecución permite obtener problemas con que pueden contar los requisitos identificados. Esta técnica se puede apreciar en el epígrafe 3.2 que pertenece a la disciplina *pruebas internas* descritas en el capítulo tres de la presente investigación.

Para que el desarrollo de un proyecto de software concluya con éxito, es importante que antes de comenzar a codificar lo que constituirá la solución, se tenga una completa y plena comprensión de los procesos relacionados con el software y un diseño correctamente elaborado para el entendimiento de los desarrolladores. En el siguiente epígrafe se describen los artefactos generados para la fase de análisis y diseño que propone la metodología, así como la evidencia del uso de los patrones de arquitectura y diseño utilizado.

2.2. Análisis y diseño

El análisis permite definir en detalle el ámbito del software y se crean modelos de los requisitos de datos, flujo de información, flujo de control y del comportamiento operativo (26). Además la etapa de diseño permite describir como el sistema va a satisfacer los requisitos. El resultado obtenido de la etapa de diseño facilita enormemente la implementación posterior de la solución, pues proporciona la estructura básica y cómo los diferentes componentes actúan y se relacionan entre sí (28).

Descripción de la arquitectura

Symfony está basado en un patrón clásico del diseño web conocido como arquitectura MVC. El sistema a desarrollar respeta dicha arquitectura, siendo el controlador el encargado de gestionar el flujo de la petición realizada a través de la clase CerafController.php, pidiendo al modelo aquello que el usuario solicita (ejemplo ActivofijoEntity.php), y devuelve como respuesta una representación del modelo a través de la vista (ejemplo GestionarAF.html.twig). La siguiente imagen ilustra la cooperación entre estos tres objetos dentro de la solución propuesta.

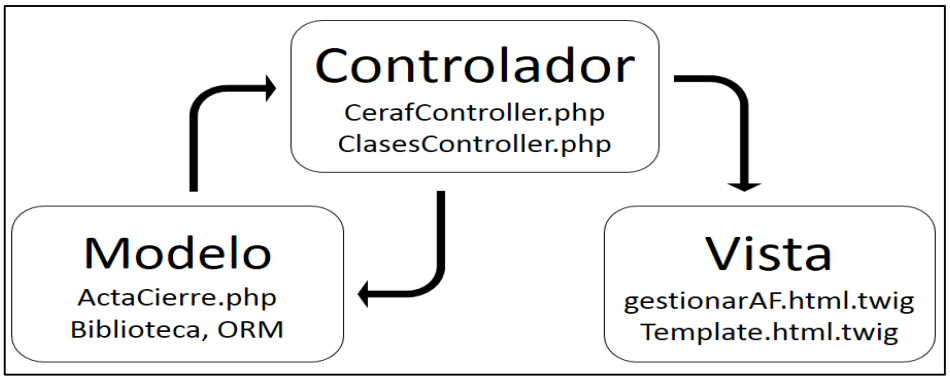


Ilustración 4. El patrón MVC en Symfony2. Fuente: elaboración propia.

El controlador

Un controlador es una función PHP que se encarga de obtener la información de la petición HTTPs y de generar y devolver la respuesta HTTPs (en forma de objeto de tipo "respuesta" (*response*) de Symfony2). El controlador contiene toda la lógica que la aplicación necesita para generar el contenido de la página. Sus componentes son: el controlador frontal, los filtros, las acciones y los objetos solicitud (*request*), respuesta (*response*) y sección (*session)* (29). La siguiente imagen ilustra el funcionamiento de dichos componentes.

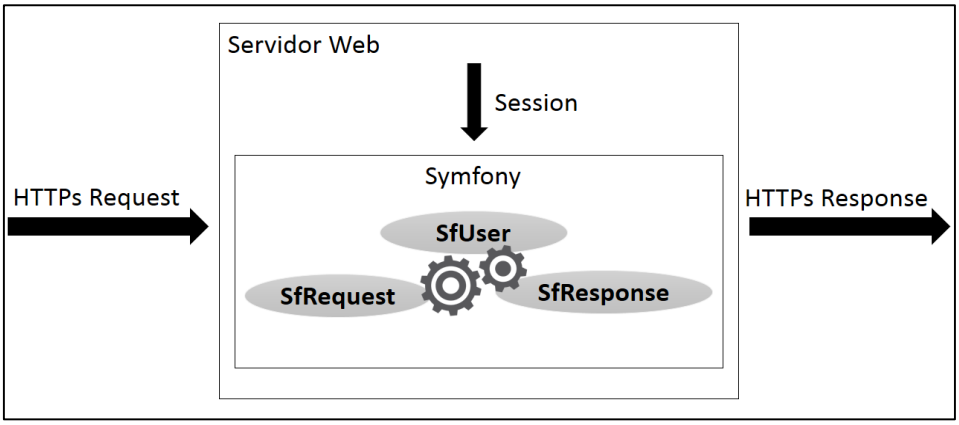


Ilustración 5. Modelo de funcionamiento de Symfony2 (Capa controladora). Fuente: (29).

La vista

La generación de vistas en Symfony se realiza según lo establecido por el patrón de diseño denominado *decorador*. Este patrón responde a la necesidad de añadir dinámicamente funcionalidad a un objeto (29). En Symfony dicho objeto sería la plantilla con la que se renderiza[5] una determinada acción de

[5] Acción realizada por una función en Symfony para devolver una vista con objetos incluidos.

algún módulo. La funcionalidad añadida dinámicamente sería el resto del documento HTML, definido en algunos de los ficheros alojados en el directorio apps/nombre_aplicacion/plantillas.

El modelo

La capa del modelo es la que se encarga de permitir el acceso a datos, logrando así una abstracción de la base de datos. Esta capa tiene entre sus componentes: el ORM (Mapeo relacional de objetos), los formularios, las extensiones propias que el programador realice de las clases del ORM y las clases y funciones propias que el programador construya para implementar la lógica de negocio (29).

La relación de todos los componentes presentes en cada capa se muestra en la siguiente imagen.

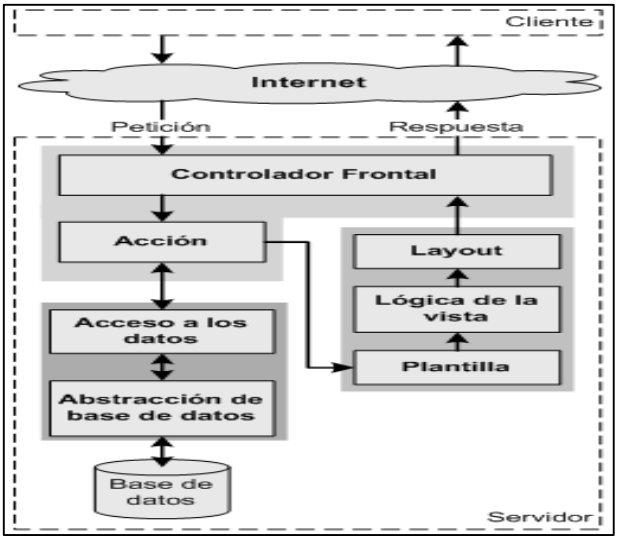


Ilustración 6. Flujo de trabajo de Symfony. Fuente: (29).

Diagrama de paquetes

La siguiente figura representa el diagrama de paquetes del sistema estructurado bajo la filosofía del marco de trabajo Symfony2, permitiendo obtener una abstracción más concreta de lo que sería la composición del sistema.

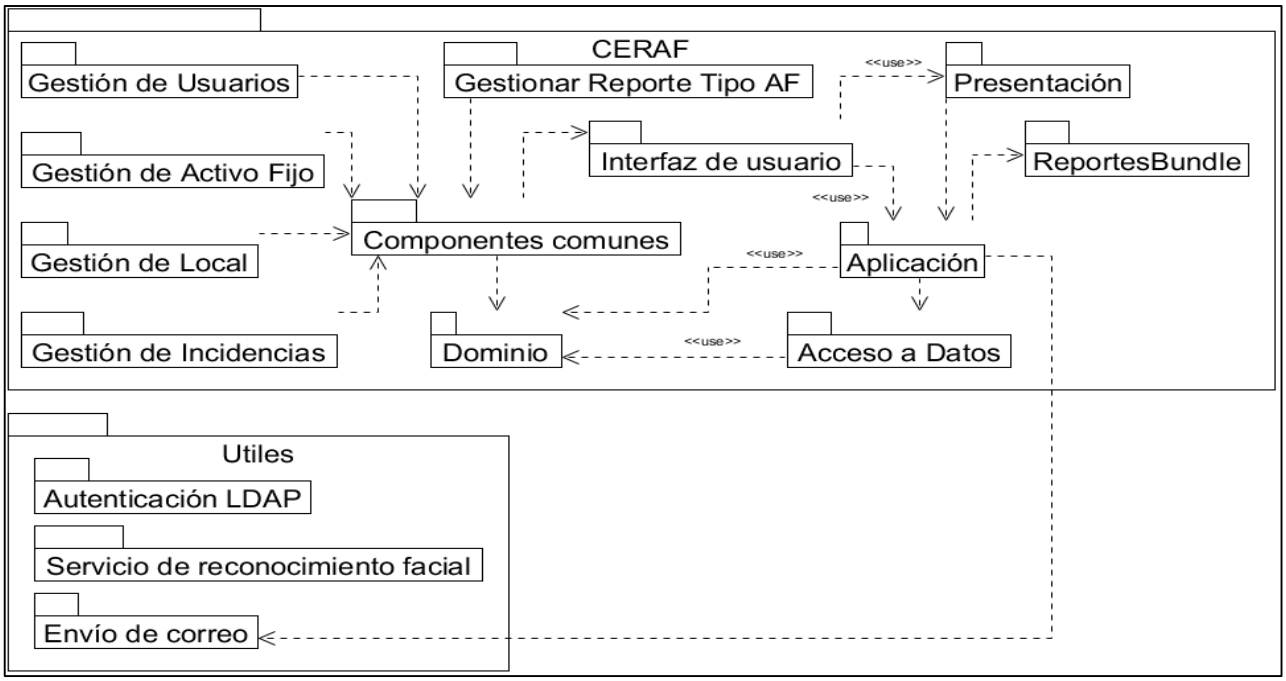


Ilustración 7. Diagrama de Paquete del sistema a desarrollar. Fuente: elaboración propia.

Diagrama de clases del diseño

Las clases del diseño sirven para describir la estructura del sistema mostrando sus clases, atributos y las relaciones entre ellos. A continuación se muestra un ejemplo de uno de los diagramas de clases del diseño creados. (Ver los restantes diagramas de clases en el Anexo 3).

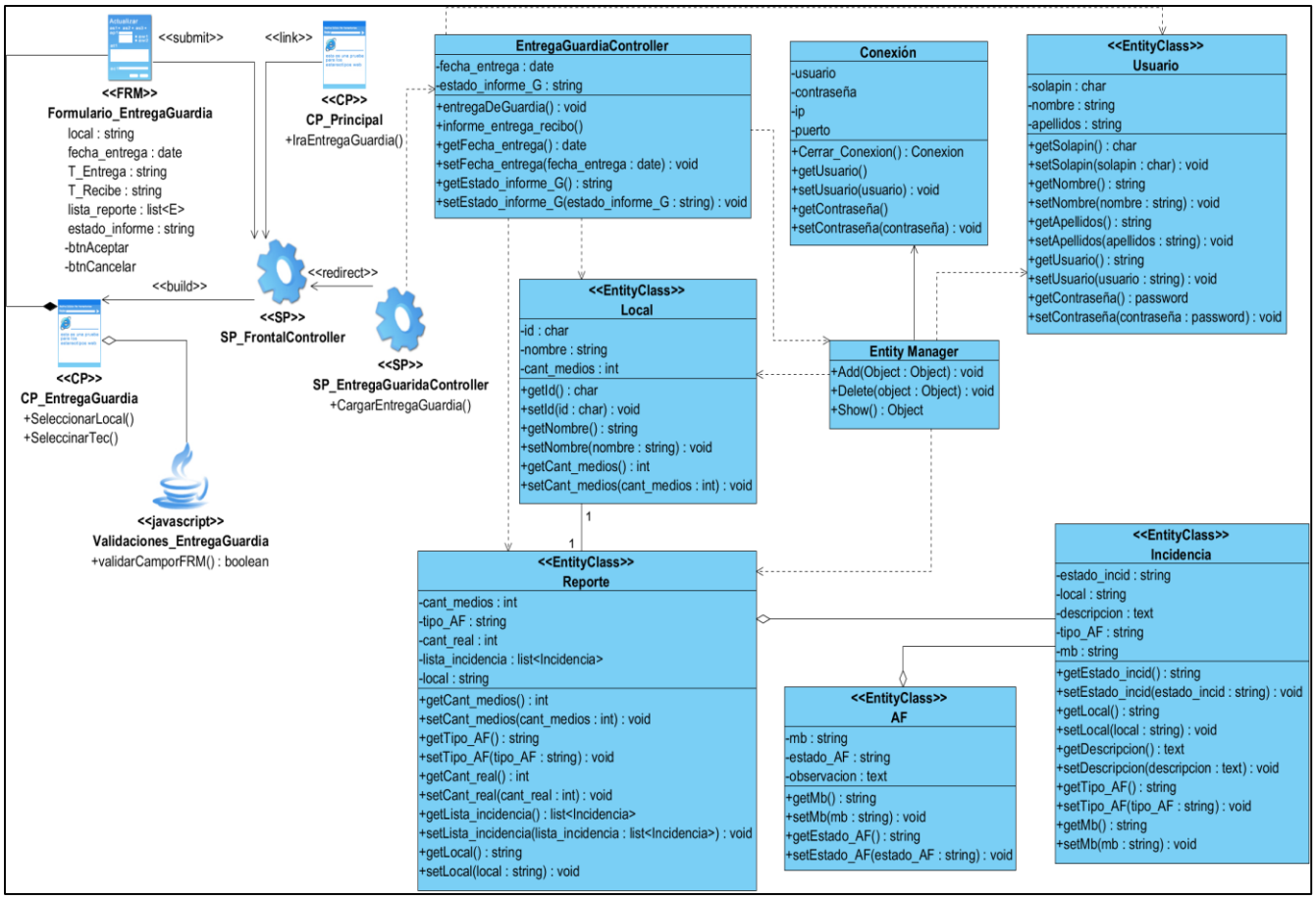


Ilustración 8. Diagrama de clase del diseño del escenario IER. Fuente: elaboración propia.

Los diagramas de clases del diseño generados muestran la utilización de los patrones de diseños presentes en las clases propuestas. A continuación se describen el uso de los mismos para el diseño de las clases del sistema.

Utilización de los patrones de diseño

Patrones GRASP

Experto: un ejemplo del uso de este patrón se evidencia en la clase *ActaCierreController* ya que la misma contiene toda la información referente a las actas de cierre, permitiendo crear una nueva y manipular las entidades creadas.

```php
class ActaCierreController extends Controller
{
    /**
     * Listando todas las ActaCierre entities.
     *
     */
    public function indexAction()
    {
        $em = $this->getDoctrine()->getManager();

        $entities = $em->getRepository('CierrePeriodoBundle:ActaCierre')->findAll();

        return $this->render('CierrePeriodoBundle:ActaCierre:index.html.twig', array(
            'entities' => $entities,
        ));
    }
```

Ilustración 9. Uso del patrón experto en la clase ActaCierreController. Fuente: elaboración propia.

Creador: es el responsable de la creación o instanciación de nuevos objetos o clases. Un ejemplo del uso de este patrón se evidencia en la clase *ActivoFijoController* permitiendo crear un nuevo AF en el sistema.

```php
class ActivoFijoController extends Controller
{
  public function createAction(Request $request)
    {
        $entity = new ActivoFijo();
        $form = $this->createCreateForm($entity);
        $form->handleRequest($request);
        if ($form->isValid()) {
            $em = $this->getDoctrine()->getManager();
            $em->persist($entity);
            $em->flush();

            return new JsonResponse(array('url'=>$this->generateUrl('af')),200);
        }
```

Ilustración 10. Uso del patrón creador en la clase ActivoFijoController. Fuente: elaboración propia.

Controlador: es un patrón que sirve como intermediario entre una determinada interfaz y el algoritmo que la implementa. Un ejemplo del uso de este patrón la podemos encontrar en la clase *TipoAfController* ya que en ella maneja la información almacenada para los tipos de AF.

```
class TipoAFController extends Controller
{
    public function indexAction(){...}
    public function createAction(Request $request){...}
    private function createCreateForm(TipoAF $entity){...}
    public function newAction(){...}
    public function showAction($id){...}
    public function editAction($id){...}
    private function createEditForm(TipoAF $entity){...}
    public function updateAction(Request $request, $id){...}
    public function deleteAction(Request $request, $id){...}
    private function createDeleteForm($id){...}
}
```

Ilustración 11. Uso del patrón controlador en la clase TipoAFController. Fuente: elaboración propia.

Alta Cohesión: en el sistema propuesto se evidencia la utilización de este patrón al asignar a cada clase las responsabilidades que le corresponde y se establecen las condiciones para que cada clase colabore con las demás. Un ejemplo de ello son los métodos (*usuarioAction*) de la clase controladora (*UsuarioController*), la cual es responsable de definir las acciones para las diferentes plantillas y colaborar con otras clases controladoras para realizar diferentes operaciones, instanciar objetos y acceder a las propiedades.

```
class UsuarioController extends Controller
  {
  public function createAction(Request $request){}
  public function newAction(){}
  public function showAction($id){}
  public function editAction($id){}
  }
```

Ilustración 12. Uso del patrón Alta Cohesión en la clase UsuarioController. Fuente: elaboración propia.

Bajo acoplamiento: se evidencia en el sistema propuesto en las clases que implementan la lógica del negocio y de acceso a datos en el modelo, puesto que estas clases no tienen relación con las de la vista o el controlador por lo que la dependencia en este caso es baja.

Patrones GoF

Decorador: la plantilla base *dashboard.html.twig* que se encuentra en el directorio src/Visual/DashBoardBundle/Resources/views/DashBoard , se conoce como plantilla global, contiene el código HTML, así como las referencias a los ficheros CSS y JavaScript que es tradicional en todas las páginas del sistema, para no tener que repetirlo. El contenido de la plantilla se integra en las demás vistas del sistema, a través de la utilización de los bloques los cuales se heredan jerárquicamente a medida que desciende el nivel de la página.

```
<html lang="en" class="no-js">
    <head>
        <meta charset="utf-8"/>
        <title>CERAF | {% block title %}{% endblock %}</title>
        <link href="{{ asset('bundles/dashboard/css/style-metronic.css') }}" rel="stylesheet" type="text/css"/>
        <link href="{{ asset('bundles/dashboard/css/style.css') }}" rel="stylesheet" type="text/css"/>
        <link href="{{ asset('bundles/dashboard/css/style-responsive.css') }}" rel="stylesheet" type="text/css"/>
        <link href="{{ asset('bundles/dashboard/css/plugins.css') }}" rel="stylesheet" type="text/css"/>
        <link href="{{ asset('bundles/dashboard/css/themes/default.css') }}" rel="stylesheet" type="text/css" id="style_color"/>
        <link href="{{ asset('bundles/dashboard/css/custom.css') }}" rel="stylesheet" type="text/css"/>
        {% block stylesheets %}{% endblock %}
        <link rel="icon" type="image/x-icon" href="{{ asset('bundles/dashboard/img/ceraf_ico.ico') }}" />
    </head>
<body class="{% block class_body %}{% endblock %}">
{% block body %}{% endblock %}
{% block javascripts %}{% endblock %}
    <script src="{{ asset('bundles/dashboard/plugins/jquery-1.10.2.min.js') }}" type="text/javascript"></script>
    <script src="{{ asset('bundles/dashboard/plugins/jquery-migrate-1.2.1.min.js') }}" type="text/javascript"></script>
    <script src="{{ asset('bundles/dashboard/plugins/bootstrap/js/bootstrap.min.js') }}" type="text/javascript"></script>
    <script src="{{ asset('bundles/dashboard/plugins/bootstrap-hover-dropdown/bootstrap-hover-dropdown.min.js') }}" type=
    "text/javascript"></script>
    <script src="{{ asset('bundles/dashboard/plugins/jquery-slimscroll/jquery.slimscroll.min.js') }}" type="text/javascript"></script>
    <script src="{{ asset('bundles/dashboard/plugins/jquery.blockui.min.js') }}" type="text/javascript"></script>
    <script src="{{ asset('bundles/dashboard/plugins/jquery.cokie.min.js') }}" type="text/javascript"></script>
    <script src="{{ asset('bundles/dashboard/plugins/uniform/jquery.uniform.min.js') }}" type="text/javascript"></script>
</body>
</html>
```

Ilustración 13. Uso del patrón decorador. Fuente: elaboración propia.

Inyección de dependencia: un ejemplo del uso de este patrón es en la clase *Usuario, a* la cual se le inyecta al atributo *$roles* su valor esperado desde el método *addRole*.

```
class Usuario extends BaseUser
{
   private $roles;
   public function addRole(\Comun\UsuarioBundle\Entity\Role $roles)
   {
        $this->roles[] = $roles;

        return $this;
   }
}
```

Ilustración 14. Uso del patrón Inyección de dependencia. Fuente: elaboración propia.

Modelo de datos

El modelo de datos permite estructurar la base de datos, en cuanto a los tipos de datos presentes y la forma en que se relacionan entre sí (30). El siguiente modelo representa las 19 entidades con que cuenta la solución propuesta para el almacenamiento de toda la información. La misma se encuentra normalizada para garantizar que no se pierdan datos cuando se borren campos, además de permitir

que no se guarde información duplicada. La base de datos se encuentra en tercera forma normal pues sus atributos dependen directamente de la clave primaria, o sea no se relacionan a través de otros atributos, eliminando de esta manera la dependencia transitiva (31).

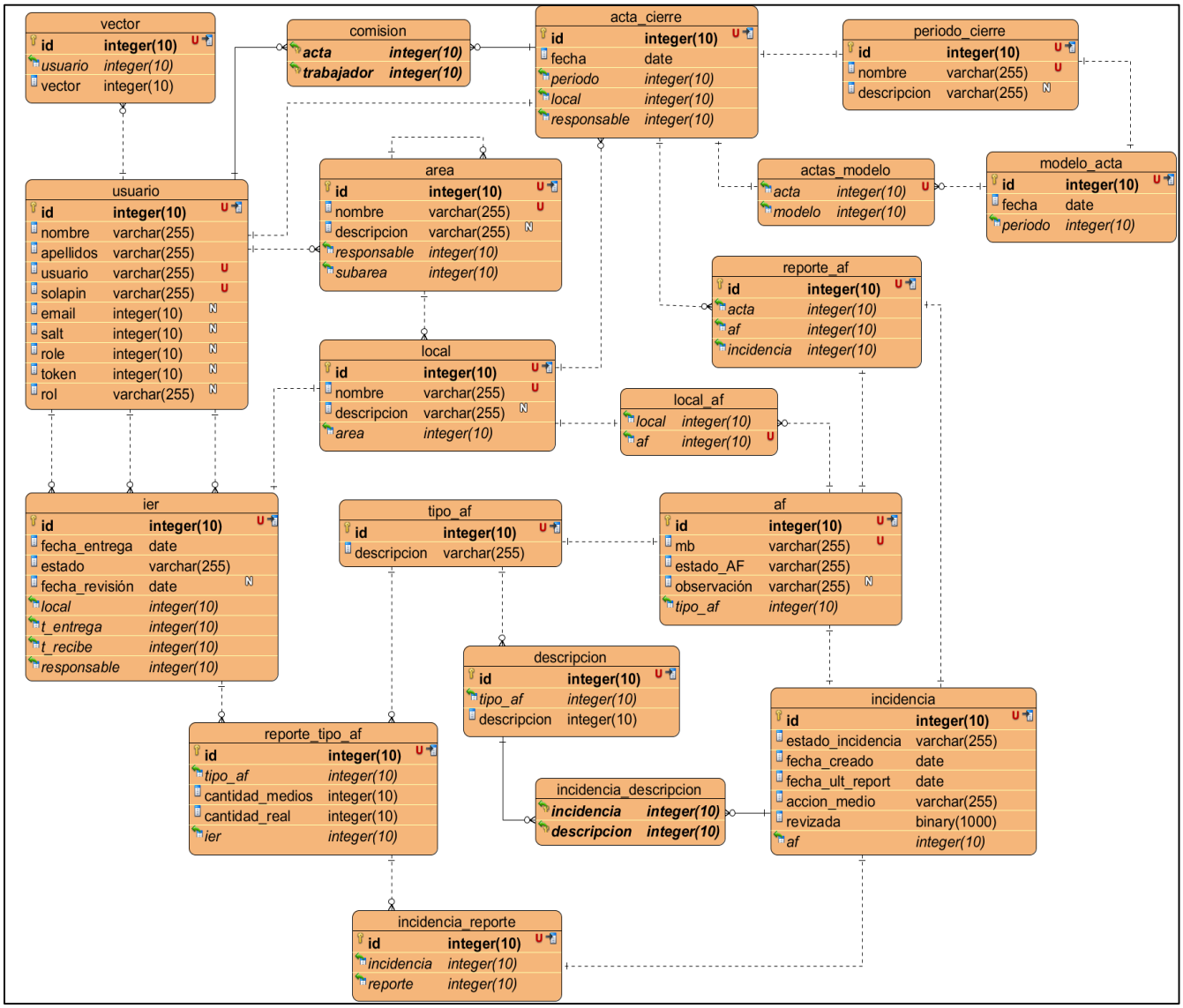


Ilustración 15. Modelo de datos de la solución propuesta. Fuente: elaboración propia.

Descripción de las tablas de datos:

- *acta_modelo:* se realiza para cada local en todos los periodos de cierre, registrando el reporte de todos los AF del local.
- *area:* contiene la información referente a las áreas con que cuenta la entidad.

- *af:* contiene el listado de medios con que cuenta la entidad, permitiendo conocer el local donde está asignado el medio y su estado (roto o buen estado).
- *usuario:* contiene toda la información referente a los usuarios que pueden acceder a la aplicación (token de seguridad para la entrega de guardia, rol y nivel de acceso, datos obtenidos mediante el directorio de servicios web de la UCI).
- *local:* contiene la información referente a los distintos locales con que cuenta la entidad, así como el área al que pertenece dicho local.
- *ier:* contiene la información de los IER de guardia generados para cada local, permitiendo saber la fecha en que fue creado, el personal que entrega, el que recibe, así como el responsable y el estado del informe (creado, revisado o correcto).
- *reporte_tipo_af:* contiene el reporte de medios según el tipo de AF con que cuenta en determinado IER de un local, permite conocer la cantidad de medios reales y la cantidad de medios a entregar en dicho local.
- *tipo_af:* contiene la información referente de todos los tipos de activos con que cuenta la entidad.
- *incidencia:* registra la información de los medios referentes a los reportes de AF realizado para un IER, permitiendo conocer la fecha en que se generó la incidencia, la acción realizada sobre el medio (extraído o local), el estado de la incidencia (pendiente o resuelta) y si ha sido revisada por el controlador del área.
- *reporte_af:* contiene la información referente a los reportes generados para los locales que son cierrados por periodo (departamentos, aulas), así como un listado de las incidencias que tienen los mismos.
- *modelo_acta:* registra todas las actas de cierre generadas para un determinado periodo de cierre.
- *periodo_cierre:* contiene las fechas para los cierres por periodos de los locales de la facultad.

La aplicación de métricas al diseño de un producto de software constituye un elemento fundamental a la hora de evaluar la calidad del mismo. Seguidamente se describen las métricas utilizadas para evaluar el diseño propuesto.

Validación del diseño

Una métrica es un instrumento que permite evaluar el software al inicio del proceso, que cuantifica además un criterio y persigue comprender mejor la calidad del producto, estimar la efectividad del proceso y mejorar la calidad del trabajo realizado a nivel de proyecto (32). Con el fin de desarrollar un

diseño robusto y sencillo se realizó la validación del mismo utilizando las métricas Tamaño Operacional de Clase (TOC) y Relaciones entre Clases (RC), ya que estas miden factores como: reutilización, encapsulamiento, complejidad y acoplamiento, siendo estos, elementos significativos en la programación orientada a objetos (POO).

Métrica Tamaño Operacional de Clases (TOC): las métricas orientadas a tamaño para una clase orientada a objetos (OO), está dado por el número de métodos asignados a una clase (tanto métodos heredados como métodos internos de dicha clase). Para esta métrica es importante que la reutilización sea inversamente proporcional a la complejidad de implementación y la responsabilidad. A mayor reutilización habrá menor complejidad y responsabilidad.

Se determinaron un total de 28 clases, con un promedio de procedimientos de 11.96. A continuación se presentan los resultados obtenidos

Tabla 6. Cantidad de clases según los atributos de calidad: Responsabilidad, Complejidad de implementación y Reutilización. Fuente: elaboración propia.

Atributo de calidad	Cantidad de clases	Promedio
Responsabilidad		
Baja	21	75
Media	7	25
Alta	0	0
Complejidad de implementación		
Baja	21	75
Media	7	25
Alta	0	0
Reutilización		
Baja	21	75
Media	7	25
Alta	0	0

Seguidamente, se describen los atributos que posibilitan medir la métrica TOC y la gráfica con los resultados luego de haber aplicado la misma.

Tabla 7. Atributos que posibilita medir la métrica TOC. Fuente: elaboración propia.

Atributos	Descripción
Reutilización	Significa cuán reutilizada es una clase o estructura de clase dentro de un diseño de software.
Responsabilidad	Responsabilidad que posee una clase en un marco conceptual correspondiente al modelado de la solución propuesta.
Complejidad de implementación	Grado de dificultad que tiene implementar un diseño de clases determinado.

Para la evaluación de las clases fueron utilizados los umbrales para medir la responsabilidad, la complejidad de implementación y la reutilización que propone la métrica TOC.

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

Ilustración 16. Atributos de calidad para las pruebas TOC. Fuente: elaboración propia.

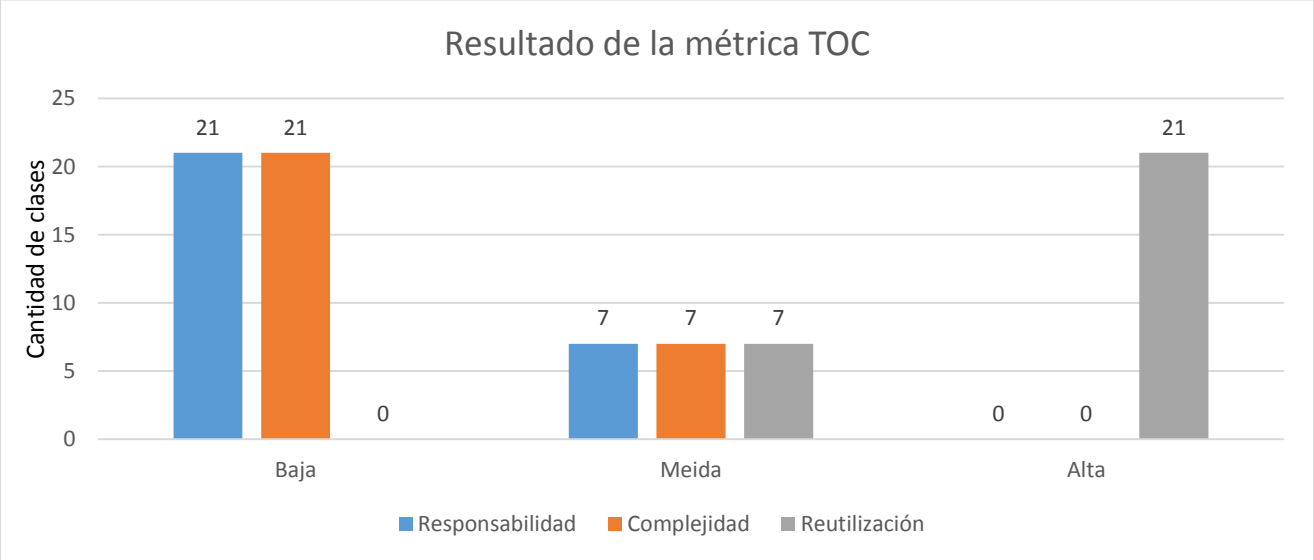


Ilustración 17. Comportamiento de los atributos de calidad de la métrica TOC. Fuente: elaboración propia.

Luego de aplicar la métrica TOC se puede concluir que la propuesta de diseño cumple con los requisitos de calidad propuestos, teniendo como resultado que las clases contienen un porciento bajo de responsabilidad y complejidad, mientras que la reutilización es alta.

Relaciones entre clase (RC): está dado por el número de relaciones de uso de una clase con otra. La métrica evalúa el acoplamiento, la complejidad de mantenimiento, reutilización y la cantidad de pruebas que permiten medir la calidad de una clase. El procedimiento comienza al calcular el promedio de asociaciones de uso, siendo en este caso 1.96 y con esto se determina las categorías a la que corresponde cada atributo de calidad. A continuación se muestra el resultado de aplicar las métricas RC al diseño del sistema.

Tabla 8. Cantidad de clases según los atributos de calidad: Acoplamiento, Complejidad de mantenimiento, Reutilización y Cantidad de pruebas propuestos por la métrica RC. Fuente: elaboración propia.

Atributo de calidad	Cantidad de clases	Promedio
Acoplamiento		
Ninguno	0	0
Baja	16	57.14
Media	3	10.71
Alta	9	32.14
Complejidad de mantenimiento		
Baja	26	92.85
Media	0	0
Alta	2	7.14
Reutilización		
Baja	2	7.14
Media	7	25
Alta	19	67.85
Cantidad de pruebas		
Baja	19	67.85
Media	7	25
Alta	2	7.14

Seguidamente, se describen los atributos que posibilitan medir la métrica RC y la gráfica con los resultados luego de haber aplicado la misma.

Tabla 9. Modo en que se afectan los atributos de calidad. Fuente: elaboración propia.

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

Para la evaluación de las clases fueron utilizados umbrales para el acoplamiento, complejidad de mantenimiento, la reutilización y cantidad de pruebas que propone la métrica RC.

	Categoria	Criterio
Acoplamiento	Ninguno	0
	Bajo	1
	Medio	2
	Alto	>2

	Categoria	Criterio
Complejidad Mant.	Baja	<= Prom.
	Media	Entre Prom. y 2*Prom.
	Alta	> 2*Prom.

	Categoria	Criterio
Reutilización	Baja	>2* Prom.
	Media	Entre Prom. y 2*Prom.
	Alta	<= Prom.

	Categoria	Criterio
Cantidad de Pruebas	Baja	<= Prom.
	Media	Entre Prom. y 2*Prom.
	Alta	> 2*Prom.

Ilustración 18. Umbrales utilizados por la métrica RC. Fuente: elaboración propia.

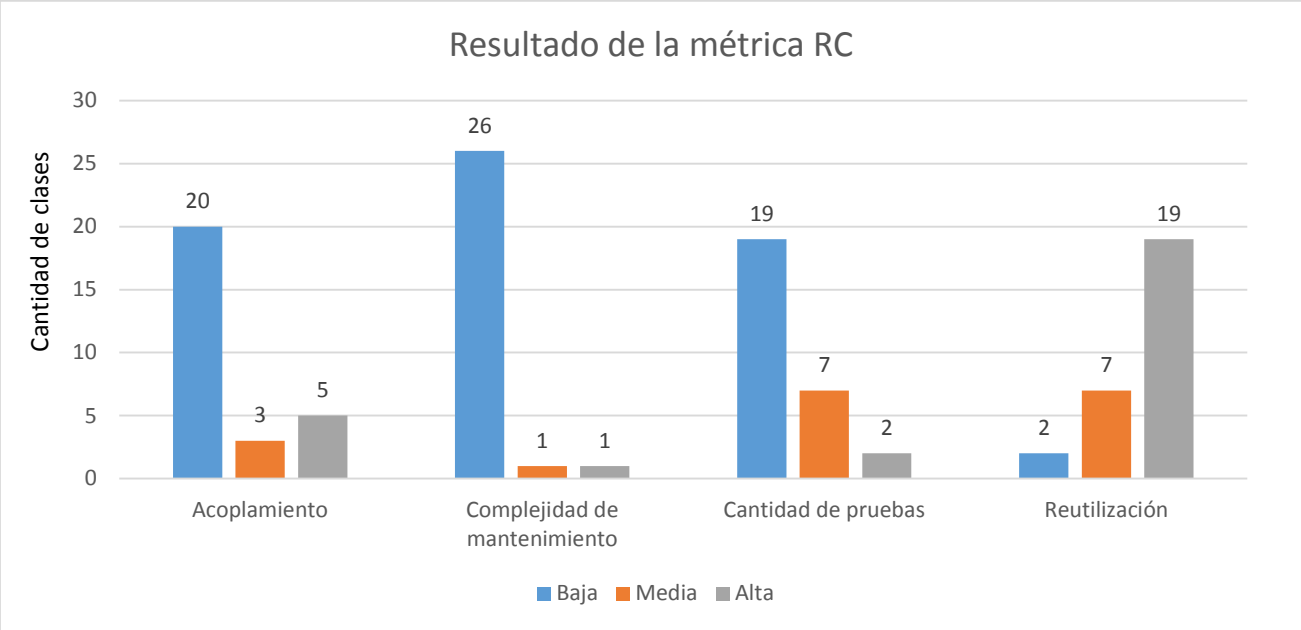


Ilustración 19. Comportamiento de los atributos de calidad de la métrica RC. Fuente: elaboración propia.

Los resultados después de aplicar la métrica de diseño RC y obtenidos los resultados de la evaluación del instrumento de medición de la métrica, se puede concluir que el diseño propuesto tiene una calidad aceptable teniendo en cuenta que aproximadamente el 92.68% de las clases empleadas poseen menos de tres dependencias de otras clases. Esto lleva a evaluaciones positivas de los atributos de calidad involucrados (bajo acoplamiento con 57.14%, baja complejidad de mantenimiento con un 67.85%, baja cantidad de pruebas con 67.85% y alta reutilización con un 67.85%). Favoreciendo de esta manera la reutilización de las clases, la baja complejidad de las mismas, así como la modificación e implantación del diseño propuesto.

Conclusiones del capítulo

- Con la aplicación de las técnicas de levantamiento de requisitos se obtuvieron los requisitos funcionales de la solución, los cuales fueron descritos mediante *historias de usuarios***,** teniendo como resultado un mejor entendimiento y comprensión de los requerimientos que debe satisfacer la propuesta de solución. Por otro lado los requisitos no funcionales fueron categorizados mediante los atributos de calidad que propone la metodología AUP-UCI para establecer las restricciones propuestas por los usuarios.
- La fase de análisis y diseño permitió la realización de los artefactos diseño de clases, modelo de datos, diagrama de componente y diagrama de despliegue y la validación de las clases del diseño mediante las técnicas TOC y RC, para facilitar la implementación de las historias de usuarios mediante un diseño confiable.

CAPÍTULO 3: IMPLEMENTACIÓN Y VALIDACIÓN DE LA PROPUESTA DE SOLUCIÓN

Introducción

En el presente capítulo se describe la nomenclatura usada tanto en el código como en los paquetes y clases del sistema. Así como los flujos de trabajo propuesto por la metodología AUP-UCI en la fase de *implementación*. Además, se muestran los resultados de la aplicación de pruebas que aseguran la calidad del sistema, siguiendo como estrategia la realización de las pruebas propuestas en las disciplinas de *pruebas internas y prueba de aceptación* presentes en la metodología utilizada.

Para una mejor comprensión del proceso de desarrollo del sistema son descritos los diagramas propuestos por la metodología para la fase de implementación.

3.1. Implementación

El objetivo de esta fase es realizar las actividades necesarias para poner a disposición de los usuarios finales, el sistema desarrollado. Seguidamente se describen los componentes en los cuales está estructurado el sistema desarrollado.

Diagrama de componente

El diagrama de componentes proporciona una visión física de la construcción del software. Muestra la organización de los componentes y sus relaciones. La siguiente imagen representa los componentes del sistema siguiendo la arquitectura Modelo-Vista-Controlador y su relación con el resto de los componentes utilizados. Cada capa es representada como un paquete, encapsulado todos sus componentes internos y mostrando su relación con los servicios externos.

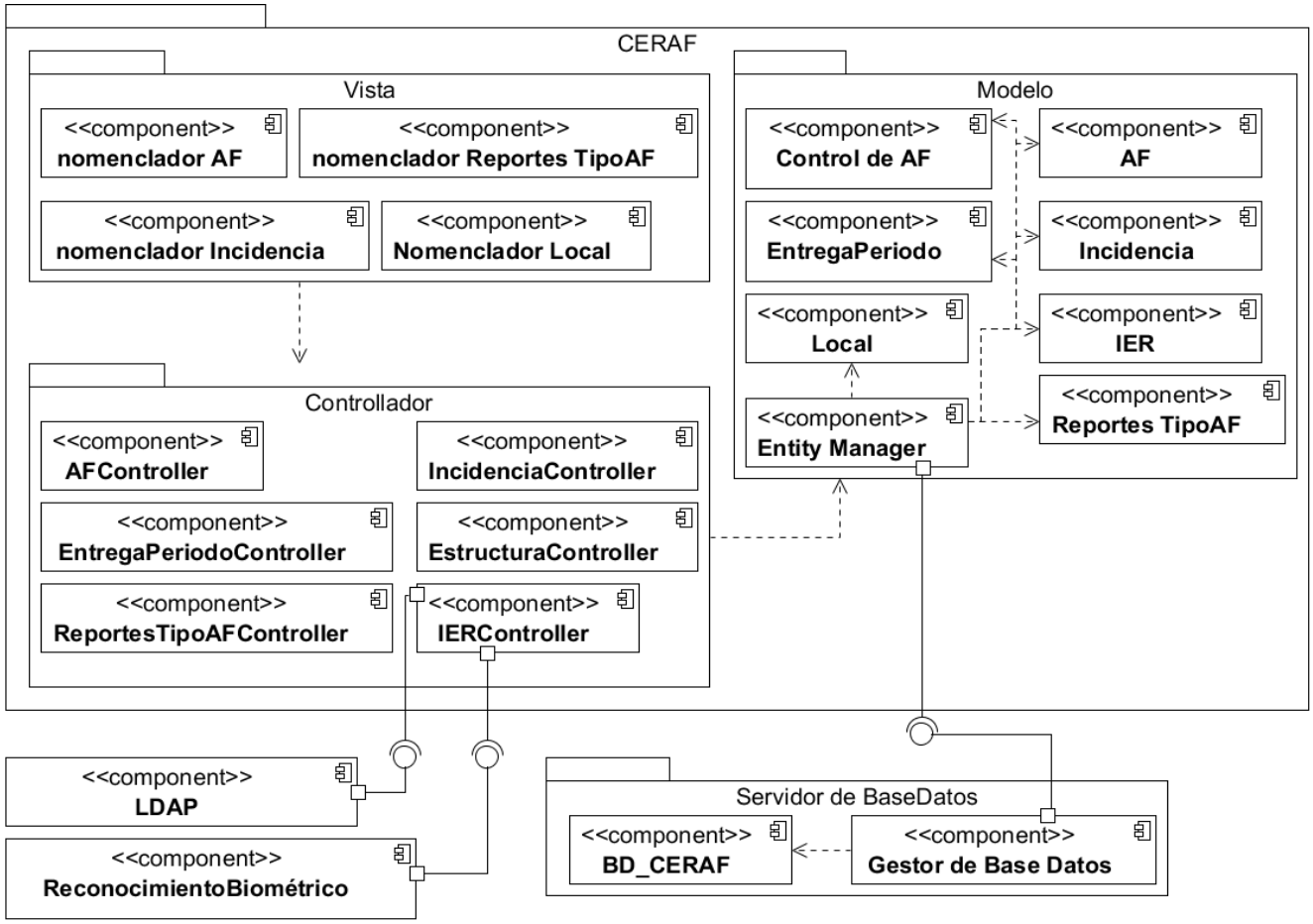


Ilustración 20. Diagrama de componentes del sistema CERAF. Fuente: elaboración propia.

Diagrama de despliegue

El diagrama de despliegue modela el hardware utilizado en la implementación del sistema y las relaciones físicas entre los componentes de hardware y software. Seguidamente se muestra el diagrama de despliegue.

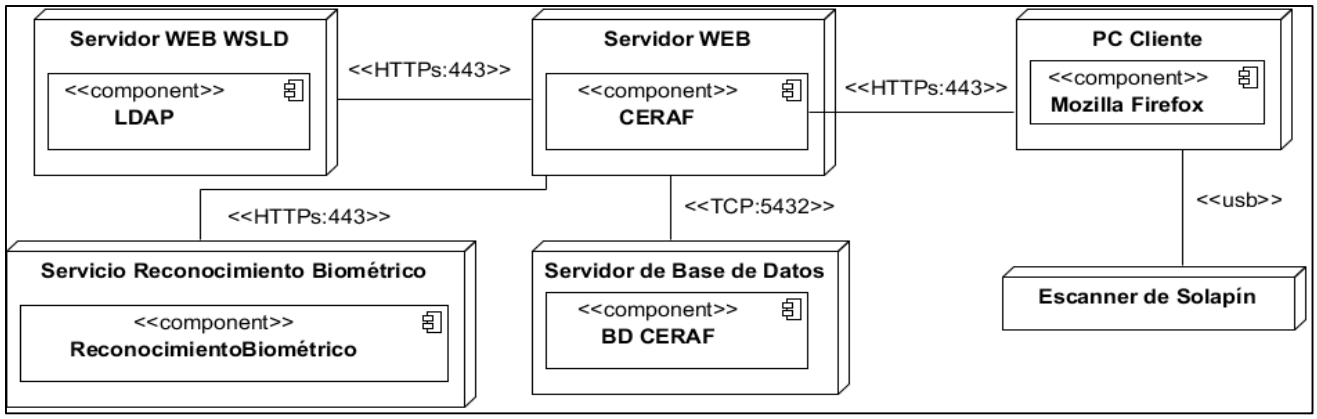


Ilustración 21. Diagrama de despliegue del sistema CERAF. Fuente: elaboración propia.

Descripción de los nodos físicos del sistema

- *PC Cliente:* el nodo representa una PC cliente desde la que se podrá acceder al sistema por medio de un navegador web e interactuar con todas las funcionalidades que este brinda.
- *Servicio de reconocimiento biométrico:* el nodo representa el servicio web que ofrece el centro de identificación de la universidad, el mismo es utilizado para identificar a los usuarios con el uso de la identificación biométrica.
- *Servidor Web***:** el nodo representa el servidor web donde está alojado el sistema así como los componentes almacenados en el mismo.
- *Servidor Web WSDL*[6]**:** el nodo representa el servidor de servicios web que ofrece la universidad. Entre los que ofrece se utiliza el servicio de Protocolo Ligero de Acceso a Directorios LDAP para la autenticación de los usuarios.
- *Servidor de Base de Datos***:** el nodo representa el servidor de Base de Datos PostgresSQL en el que está alojada la base de datos.
- *Escáner de Solapín***:** el dispositivo representa el escáner usado para confirmar la aceptación del IER.

Descripción de los protocolos utilizados:

- *HTTPS***:** protocolo para Transferencia de Hipertexto Seguro. Extensión del HTTP para la autenticación y encriptación de datos entre un servidor web y un navegador web.

[6] Web Services Description Languaje.

- *TCP/IP:* arquitectura definida por los protocolos, TCP (Protocolo de Control de Transmisión) y Protocolo de Internet (IP) (Transmission Control Protocol y Internet Protocol). Forma de comunicación básica que usa el Internet.
- *USB***:** Bus Universal en Serie (Universal Serial Bus) interfaz entre la PC y ciertos dispositivos.

Para una mejor comprensión del código generado, se hace uso del estándar de codificación de Symfony, logrando de esta manera tener una nomenclatura en común para todas las clases y métodos.

Estándar de codificación

Convenciones de nomenclatura: los nombres de las clases comienzan con la primera letra en mayúscula y el resto con minúscula, en caso de ser una palabra compuesta, iniciará cada palabra con letra mayúscula.

*Ejemplo***:** *ActaCierre.php, ReporteTipoAF.php*

Los nombres de los métodos y los atributos de las clases, comienzan con la primera letra en minúscula, en caso de que sea un nombre compuesto iniciarán cada palabra con letra mayúscula excepto la primera palabra. Adicionalmente le es agregado el sufijo *Action propuesto por el marco de trabajo para identificar los métodos del controlador que responden a peticiones HTTPs.*

Ejemplo: *editarAction, actualizarAction*

Nomenclatura según el tipo de clase: las clases que se encuentran dentro del paquete *Controller* se nombran adicionándoles el nombre del controlador de Symfony2, del cual heredan al final del nombre de la clase (*IerController*). El marco de trabajo se basa en esta técnica para identificar sus clases controladoras.

Ejemplo: *ReporteTipoAFController.php*

Las clases que se encuentran dentro del paquete *Form* se nombran adicionándoles como sufijo del nombre la palabra *Type*, El marco de trabajo propone esta técnica para un mejor entendimiento de sus clases.

Ejemplo: *ReporteTipoAFType, IerType*

Normas de comentarios

Se debe comentar todo lo que se haga dentro del desarrollo, logrando establecer un código legible y reutilizable y así se pueda aumentar su mantenibilidad a lo largo del tiempo.

Comentarios de clases: antes de declarar una clase se brinda una descripción de esta, donde se explica el propósito de la misma y se escribe de la siguiente manera.

```
/**
 * Ier controller.
 *
 * @Route("/ier")
 * controla todas la funciones del Informe de entrega y recibo generado
 * actuando de mediador entre los datos que gestiona la vista y el modelo
 * paquete: GuardiaLabBundle
 */
```

Ilustración 22. Nomenclatura de comentario en las clases del sistema. Fuente: elaboración propia.

Comentario en las funciones: los comentarios redactados al inicio de las funcionalidades describen el objetivo de la misma así como los parámetros con que cuenta y el tipo de resultado que arroja. A continuación se muestra un ejemplo.

```
/**
 * Crea un formulario para crear una entidad Activo Fijo.
 *
 * @param ActivoFijo $entity The entity
 *
 * @return \Symfony\Component\Form\Form The form
 */
```

Ilustración 23. Nomenclatura de comentario en las funciones del sistema. Fuente: elaboración propia.

Tipos de mensajes utilizados

Se definieron 3 tipos de mensajes para lograr un mejor entendimiento entre el usuario y las acciones que realiza, estos mensajes son:

- Mensaje de error.
- Mensaje de confirmación.
- Mensaje de información.

Mensaje de error: Muestra al usuario que ha realizado una opción incorrecta, Ejemplos:

- El usuario ya existe.
- El formulario tiene algunos errores.

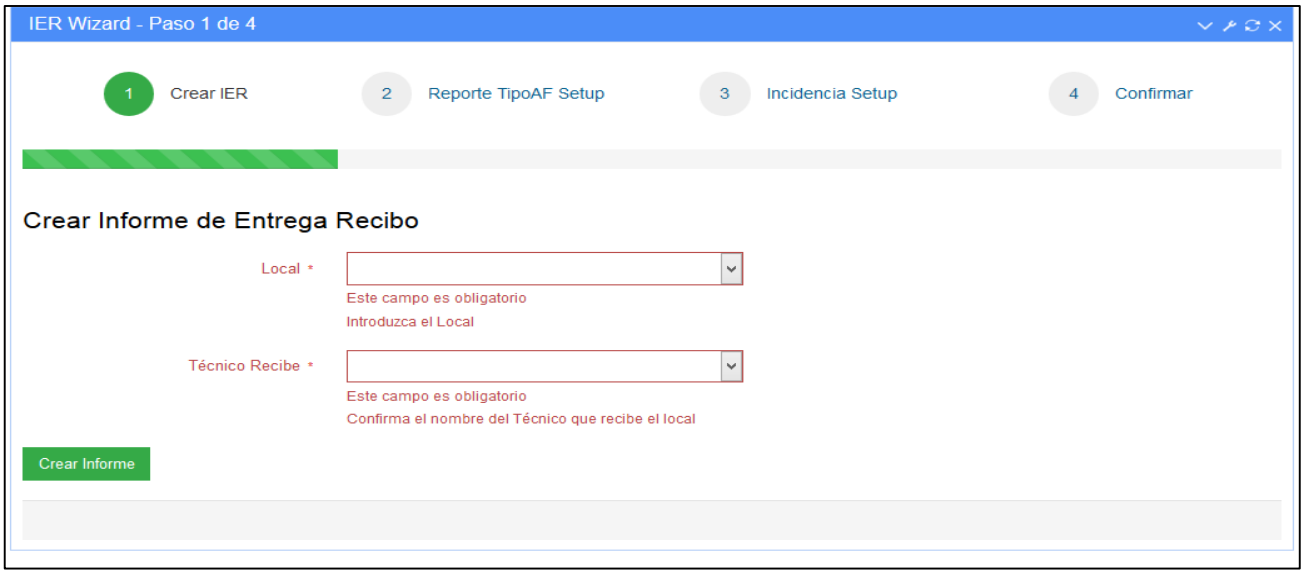


Ilustración 24. Mensaje de error en el formulario IER. Fuente: elaboración propia.

Mensaje de confirmación: se utiliza cuando es necesario asegurarse de que el usuario desea realizar una acción, por ejemplo cuando va a eliminar un elemento es necesario asegurarse que eso es lo que realmente desea el usuario, ejemplo:

- ¿Estás seguro que desea eliminar el elemento marcado?

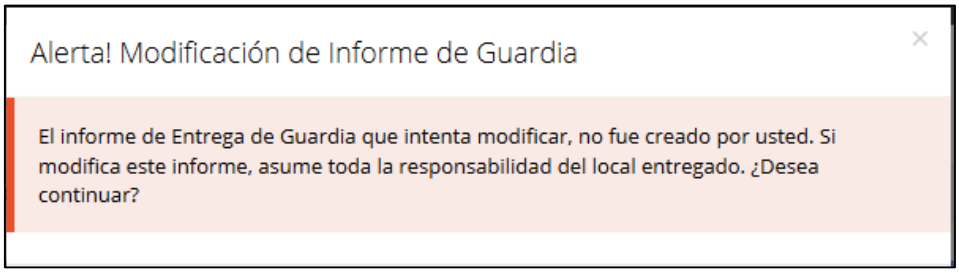


Ilustración 25. Mensaje de confirmación del sistema. Ejemplo: Modificación del IER. Fuente: elaboración propia.

Mensaje de información: se utiliza para brindarle alguna información al usuario, cuando se busca, actualiza, o ingresa un nuevo nomenclador, ejemplo:

- El Activo ha sido insertado correctamente.
- El Local ha sido modificado correctamente.

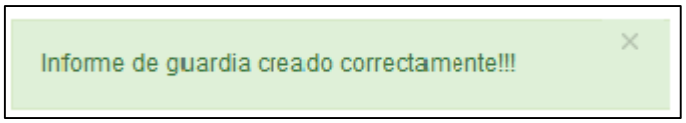


Ilustración 26. Mensaje de información del sistema. Fuente: elaboración propia.

Durante el proceso de desarrollo de software es necesario llevar un control estricto de las posibles salidas o resultados que arrojará el sistema, para ello se hace uso de un conjunto de pruebas. Las mismas se describen a continuación.

3.2. Pruebas Internas

El objetivo principal de esta disciplina es evaluar la calidad del producto que se desarrolló mediante ciertos factores de pruebas como son (33):

- Probar si el software no hace lo que debe.
- Probar si el software hace lo que no debe, es decir, si provoca efectos secundarios adversos.
- Descubrir un error que aún no ha sido descubierto.
- Encontrar el mayor número de errores con la menor cantidad de tiempo y esfuerzo posibles.
- Mostrar hasta qué punto las funciones del software opera de acuerdo con las especificaciones y requisitos del cliente.

A continuación de describen las *pruebas unitarias* y las *pruebas funcionales* pertenecientes a la disciplina de *pruebas internas* planteada por la metodología para validar la calidad del producto creado.

Pruebas unitarias

La base de este método es el de hacer pruebas en pequeños fragmentos al programa. Cada prueba deberá ser lo más independiente posible de las demás y encargarse de una tarea específica, en programación procedural u orientada a objetos se puede afirmar que estas unidades son los métodos o las funciones que tenemos definidos (34).

El objetivo de las pruebas unitarias es el aislamiento de partes del código y la demostración de que estas partes no contienen errores. Para realizar dichas pruebas, se recurrió a la utilización del marco de trabajo *PHPUnit*, el cual constituye un entorno para ejecutar pruebas internas en el lenguaje de programación PHP. Esta herramienta pertenece a la familia *xUnit* desarrollada por Kent Beck (35).

La ilustración 26 muestra la estructura de directorios donde se encuentran las pruebas dentro del proyecto Symfony2. El mismo está constituido por las carpetas *Controller* y *Entity*, donde se ubican las pruebas realizadas a los controladores y entidades respectivamente. Se realizaron un total de 17 pruebas a las entidades y a los controladores del sistema, todos los errores fueron corregidos en la medida en que se fueron identificando. Estas pruebas arrojaron como resultado final que las 17 pruebas fueron satisfactorias, para un 100%.

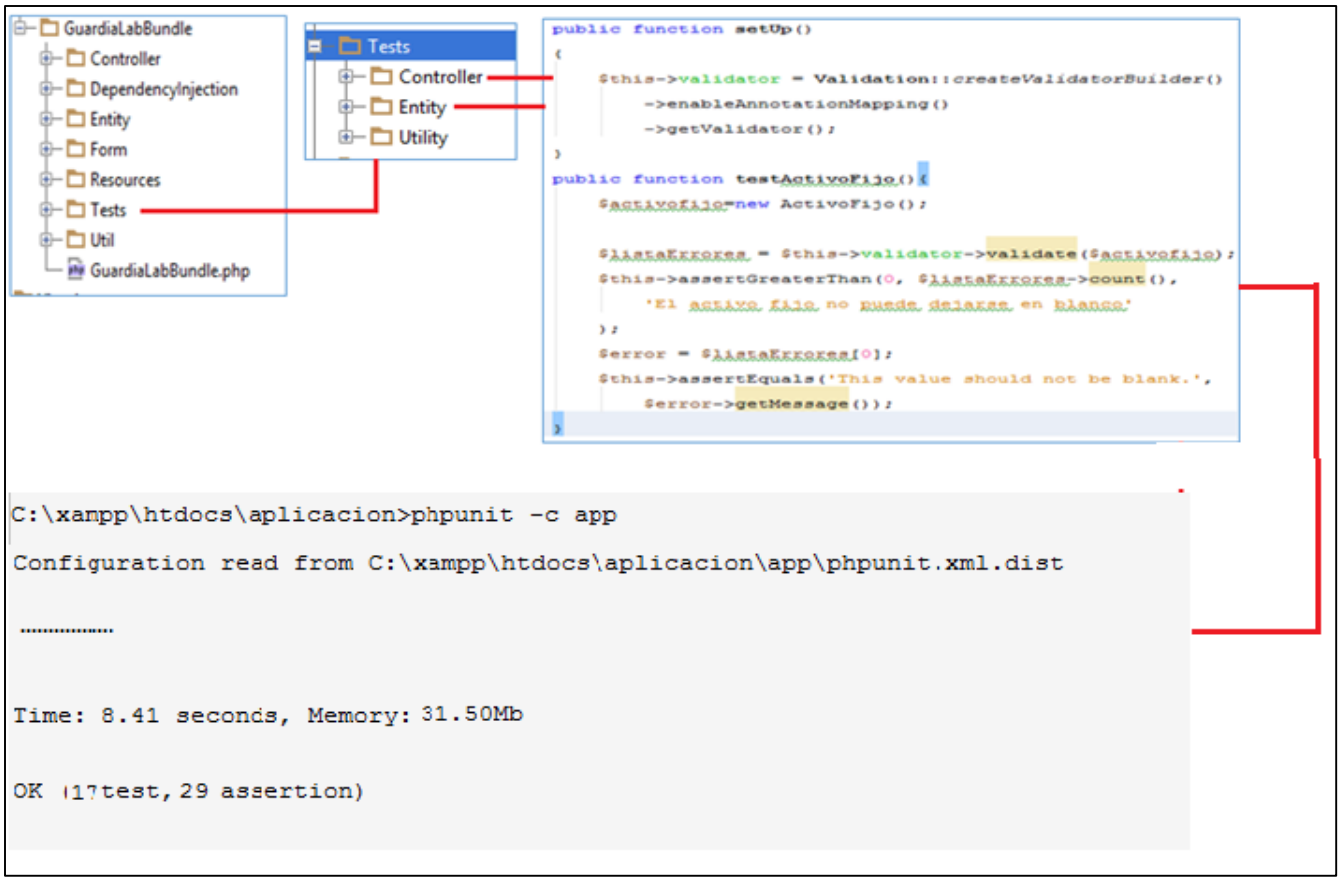


Ilustración 27. Resultado de las pruebas internas realizadas al código. Fuente: elaboración propia.

La realización de estas pruebas permitió llevar un control estricto de la implementación del sistema, arrojando como resultado que las funcionalidades del código responden a los requerimientos para los que fueron creados.

Pruebas de caja negra

Las pruebas de caja negra o pruebas funcionales se realizan sobre la interfaz del software, comprobando las entradas y salidas de datos. Estas pruebas se realizan para comprobar que cada función es operativa, utilizando el artefacto *diseño de caso de prueba***,** que tienen como objetivo introducir juegos de datos que ayuden a la ejecución de los casos y facilite que el sistema se ejecute en todas sus variantes. Se intenta encontrar con estas pruebas (36):

- Funciones incorrectas o ausentes.
- Errores de interfaz.
- Errores en estructuras de datos o en accesos a las Bases de Datos externas.

- Errores de inicialización y terminación.

Entre las técnicas para desarrollar la prueba de Caja Negra se encuentran (36):

- Técnica de la Partición de Equivalencia: divide el campo de entrada en clases de datos que tienden a ejercitar determinadas funciones del software.
- Técnica del Análisis de Valores Límites: prueba la habilidad del programa para manejar datos que se encuentran en los límites aceptables.
- Técnica de Grafos de Causa-Efecto: permite al encargado de la prueba validar complejos conjuntos de acciones y condiciones.

De las antes mencionadas, se aplicó la técnica de Partición de Equivalencia, esta permite definir casos de prueba que declaren clases de errores, reduciendo el número de casos de prueba a desarrollar para demostrar que las funciones del software son operativas. (Ver todos los DCP *en el Anexo 4*). A continuación se muestra un ejemplo de los DCP realizados.

Caso de prueba – Crear IER

Condiciones de ejecución:

- Se debe identificar y autenticar ante el sistema y además debe tener los permisos para ejecutar esta acción.
- Se debe seleccionar la opción crear IER.
- Debe existir al menos un local en el sistema.
- Debe existir al menos un técnico para recibir y un técnico para entregar el local.

SC 1 Adicionar IER

Tabla 10. Caso de Prueba: Crear IER. Fuente: elaboración propia.

Escenario	Descripción	Local	Técnico Recibe	Respuesta del sistema	Flujo central
CP 1.1: Adicionar IER introduciendo datos válidos.	Se crea un IER nuevo en el sistema	V Lab_101	V Fernando Torres	El sistema adiciona el IER y muestra el mensaje de información: "Informe de guardia creado correctamente."	• Se selecciona la opción crear nuevo IER. • El sistema carga automáticamente los atributos que necesitan ser introducidos por el usuario. • El usuario introduce los datos.

Escenario	Descripción	Local	Técnico Recibe	Respuesta del sistema	Flujo central
					• El usuario presiona el botón adicionar. • El sistema muestra el mensaje "Informe de guardia creado correctamente."
CP 1.2: Adicionar IER dejando al menos un campo vacío.	Se crea un IER nuevo en el sistema	I Vacío	I vacío	El sistema marca en rojo el campo mostrando el mensaje: "Este campo es obligatorio."	• Se selecciona la opción crear nuevo IER. • El sistema carga automáticamente los atributos que necesitan ser introducidos por el usuario. • El usuario introduce los datos. • El usuario presiona el botón adicionar. • El sistema muestra un mensaje de error y no se adicionan los datos al sistema.

Descripción de las variables.

Tabla 11. Descripción de las variables utilizadas en el DCP: Crear IER. Fuente: elaboración propia.

No	Nombre de campo	Clasificación	Valor Nulo	Descripción
1	Local	Lista desplegable	No	Selecciona el local al cual está asociado el IER que se crea
2	Técnico Recibe	Lista desplegable	No	Selecciona el nombre de la persona que va a recibir el local.

Caso de prueba – CP 1.1 Adicionar IER introduciendo datos válidos.

Ilustración 28. Escenario de prueba Adicionar IER introduciendo datos válidos. Fuente: elaboración propia.

Caso de prueba – CP 1.2 Adicionar IER dejando al menos un campo vacío.

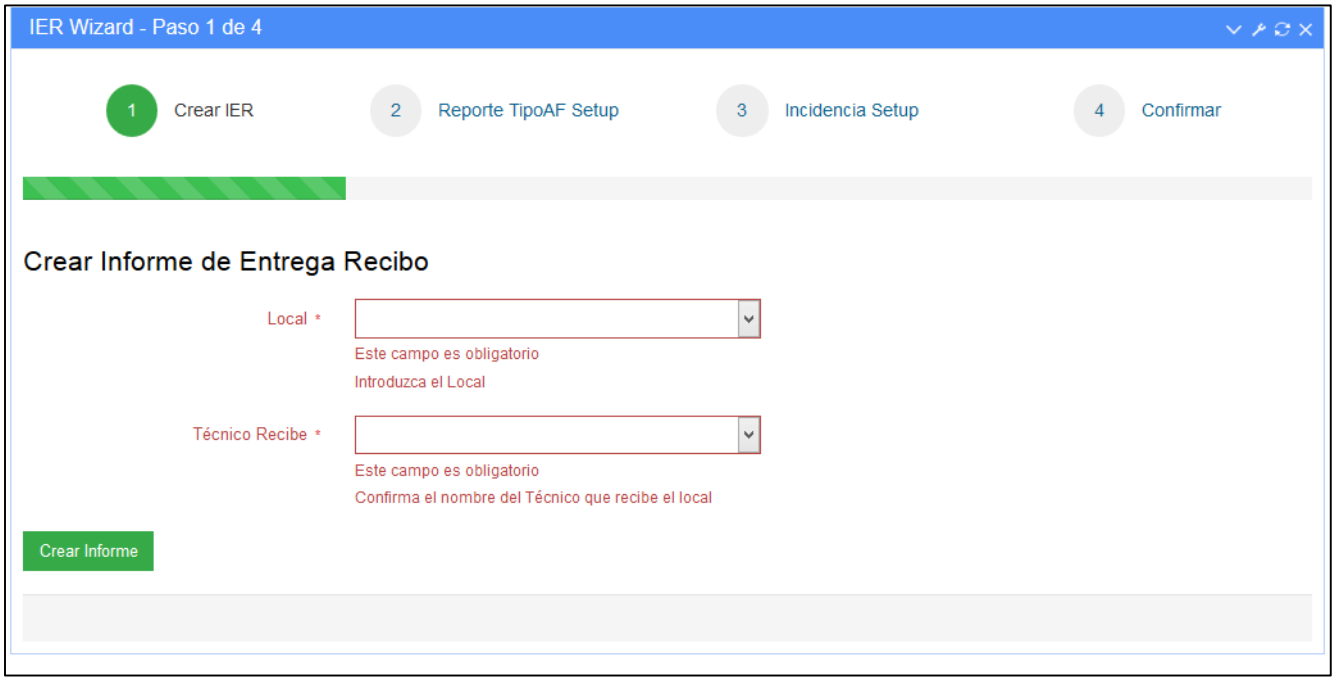


Ilustración 29. Escenario de prueba Adicionar IER dejando al menos un campo vacío. Fuente: elaboración propia.

La gráfica de la ilustración 30, muestra los resultados de las pruebas de aceptación realizadas al finalizar cada una de las iteraciones en que quedó dividido el proceso de desarrollo.

En sentido general, se refleja un ascenso en el número de casos de pruebas descritos y ejecutados en cada iteración con respecto a la anterior y una tendencia creciente en la efectividad de las pruebas.

En la primera iteración se implementaron y probaron 9 casos de prueba de las cuales 5 resultaron satisfactorias para un 55,55% de efectividad. En la segunda iteración se implementaron las funcionalidades descritas en otros 14 casos de pruebas y se realizaron correcciones a las 4 que quedaron pendientes de la iteración anterior. Se ejecutaron un total de 18 pruebas de aceptación, teniendo como resultado 15 pruebas satisfactorias y 3 no satisfactorias, para un 83.33% de efectividad. Todas las no conformidades pendientes de la primera iteración fueron resueltas. En la tercera iteración se terminó de probar los casos de prueba restantes. Se corrigieron los problemas de la iteración anterior y se realizaron 17 pruebas nuevas para un total de 20 pruebas de aceptación quedando solo 2 pendientes (90% de efectividad). Por último se repitieron todas las pruebas resultando satisfactoria en un 100%. En lo adelante se muestra el resultado de las pruebas funcionales por cada iteración.

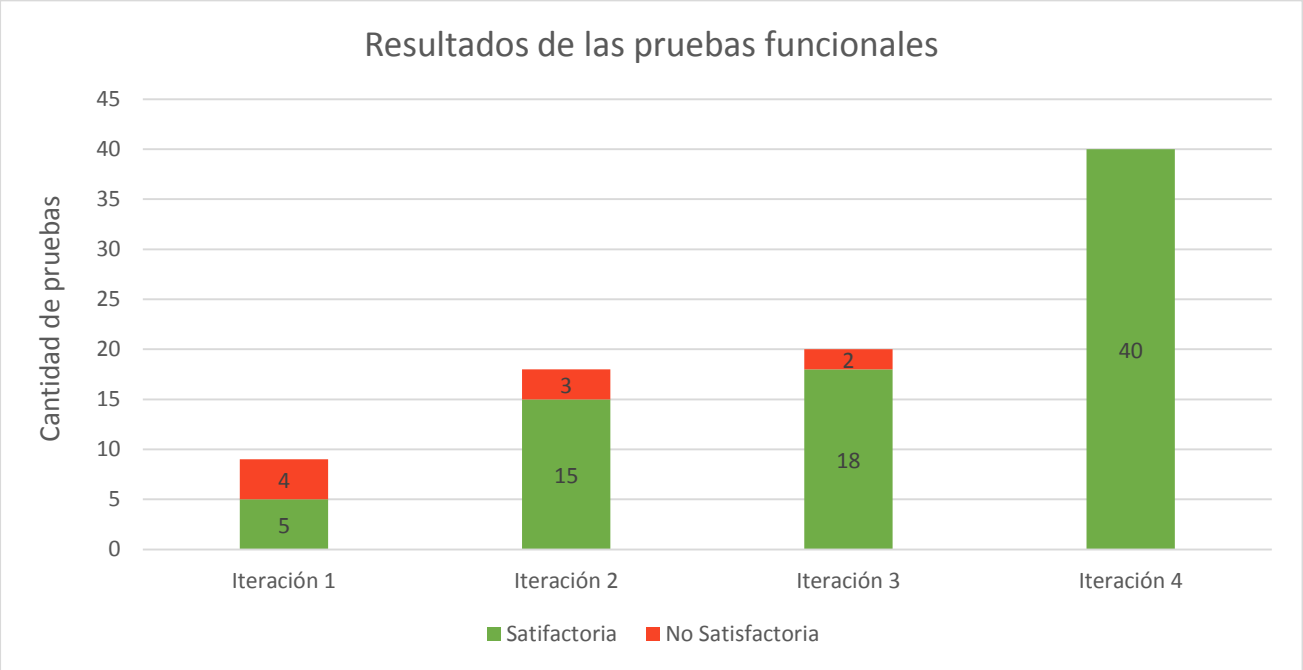


Ilustración 30. Resultado de las pruebas funcionales. Fuente: elaboración propia.

3.3. Pruebas de Aceptación con el cliente

Para realizar las pruebas de aceptación con el cliente, se utilizó el mismo principio de las pruebas funcionales, permitiendo que sea el propio cliente el que realice dichas pruebas, utilizando juegos de datos en cada una de las interfaces seleccionadas. Se realizaron 3 iteraciones quedando reflejada el resultado de las mismas en la ilustración 31. En la primera Iteración se realizaron pruebas a los módulos de seguridad, nomencladores y al módulo de entrega de guardia en los laboratorios, siendo identificados 8 no conformidades. Para la segunda iteración fueron probados nuevamente todos los

módulos de la primera iteración sumándole también pruebas al módulo encargado de la entrega por periodos y al módulo encargado de la gestión de incidencias, siendo identificadas 5 no conformidades. En la última iteración se probó el sistema en su totalidad, incluyendo los niveles de acceso y los mecanismos de seguridad implementados para la entrega de guardia de los laboratorios, demostrando que el sistema creado se encuentra apto para ser usado en el entorno real para el que fue creado.

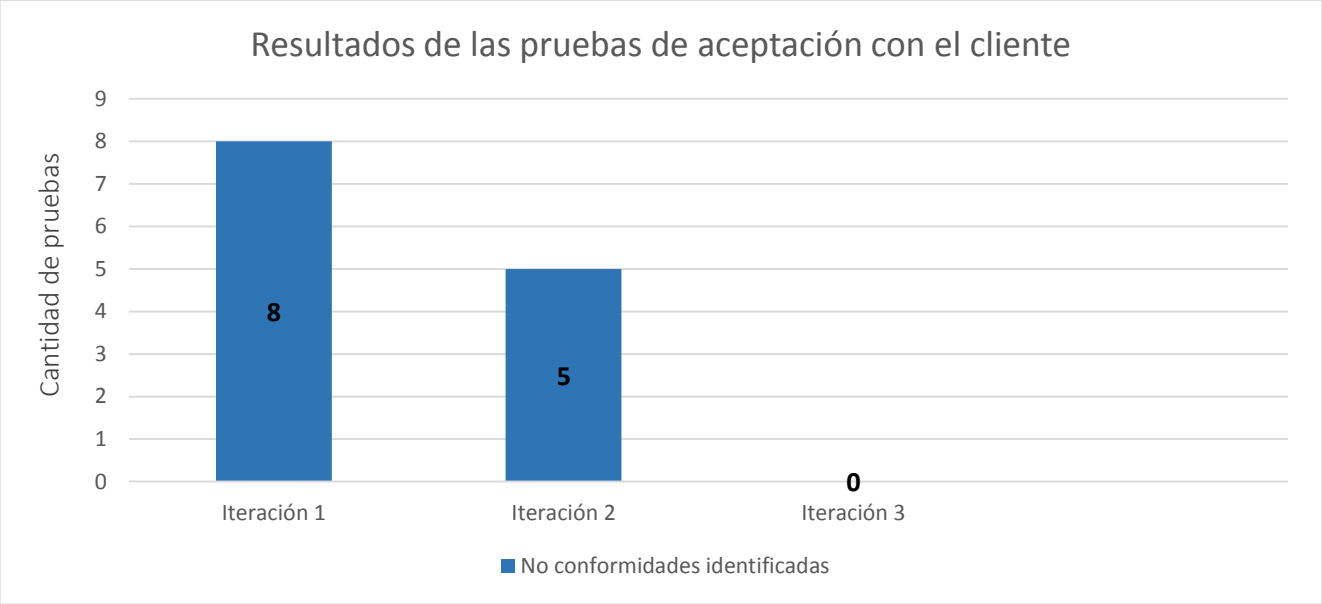


Ilustración 31. Cantidad de no conformidades encontradas por iteración. Fuente: elaboración propia.

La explotación del sistema en el entorno real para el que fue concebido, permitió generar un conjunto de reportes que evidencian las facilidades del sistema para cumplir con los requerimientos planteados al inicio de esta investigación. Identificando como resultado los que a continuación se muestran.

3.4. Control sobre los AF y fácil manejo y tratamiento de la información manipulada.

En la siguiente ilustración se muestra el inventario de los medios permitiendo conocer fácilmente qué estado tiene el activo (Buen estado o Roto) así como saber en qué local se encuentra el mismo.

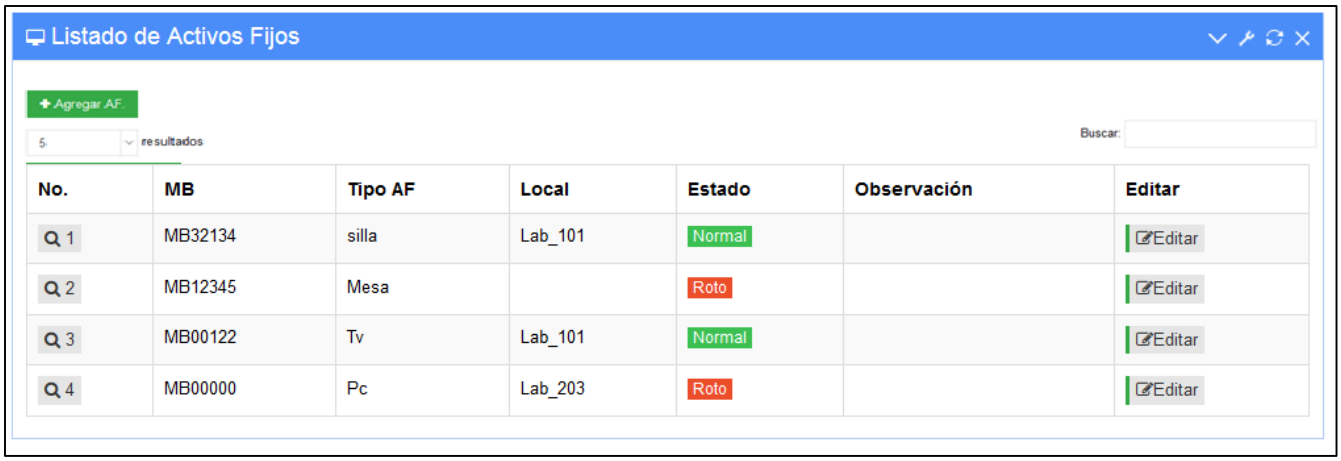


Ilustración 32. Inventario de AF. Fuente: elaboración propia.

Por otra parte el control sobre los informes de guardia creados para la entrega y recibo de los laboratorios docentes y las actas de control de AF creadas para los locales, permiten conocer en detalle sobre el estado de los reportes e incidencias generados para los medios ubicados en estos locales. De esta forma, se puede acceder al estado de las incidencias reportadas, conocer cuando se creó, quien la creó, cuántas veces se ha reportado y la acción que ocurre sobre él, o los AF reportados en las incidencias. A continuación se muestran los detalles de los reportes creados por día.

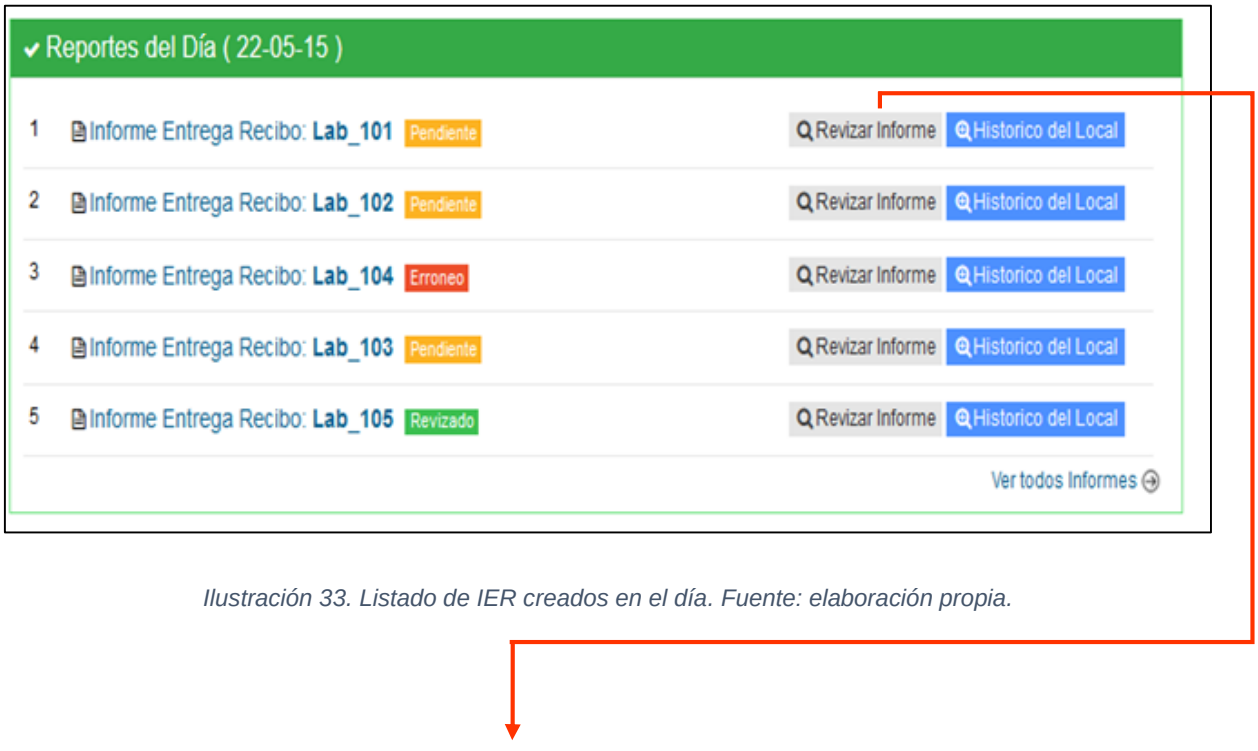


Ilustración 33. Listado de IER creados en el día. Fuente: elaboración propia.

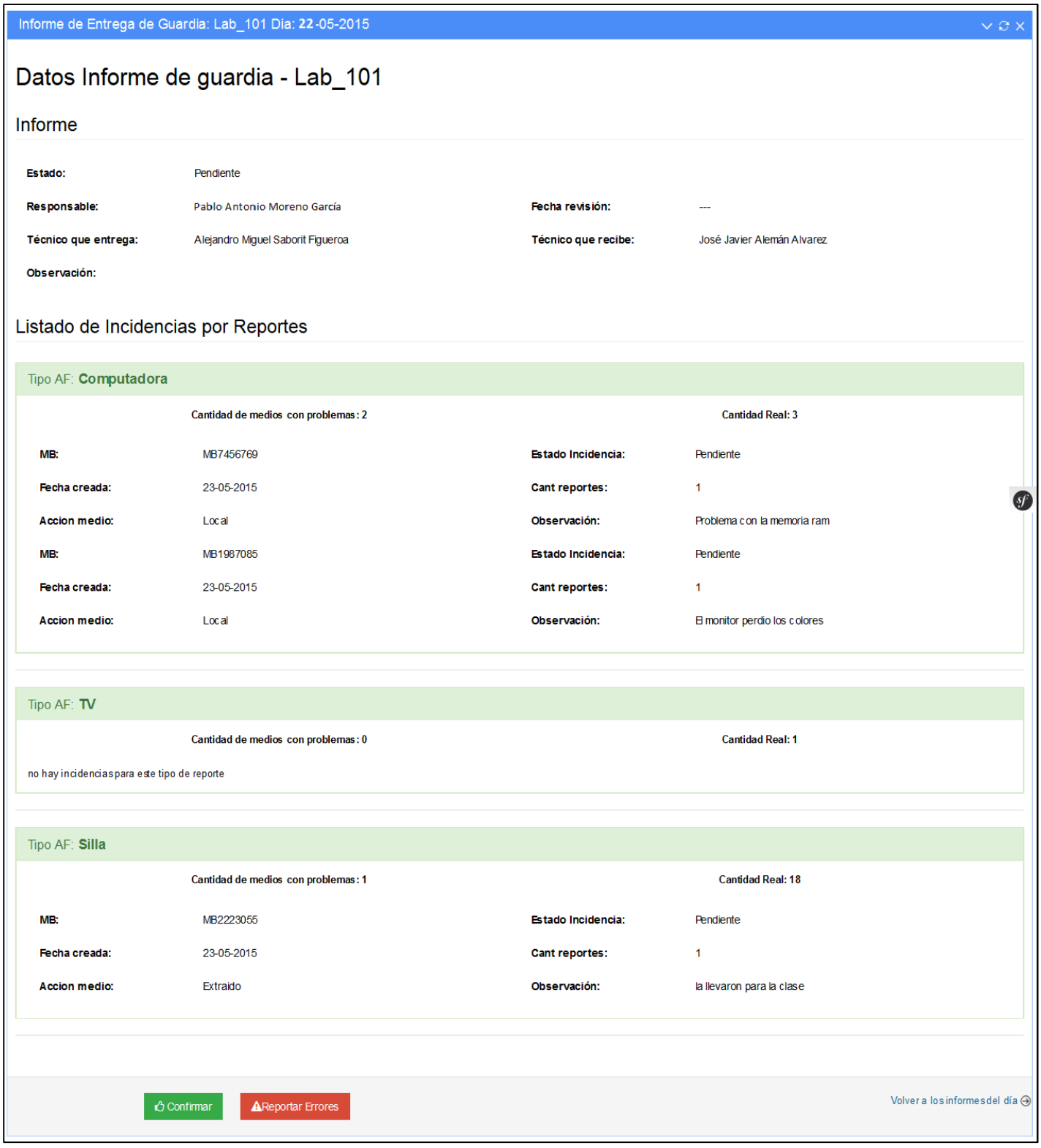


Ilustración 34. Detalle del IER creado. Fuente: elaboración propia.

Búsqueda de Informes de Entrega y recibo creados por Local:

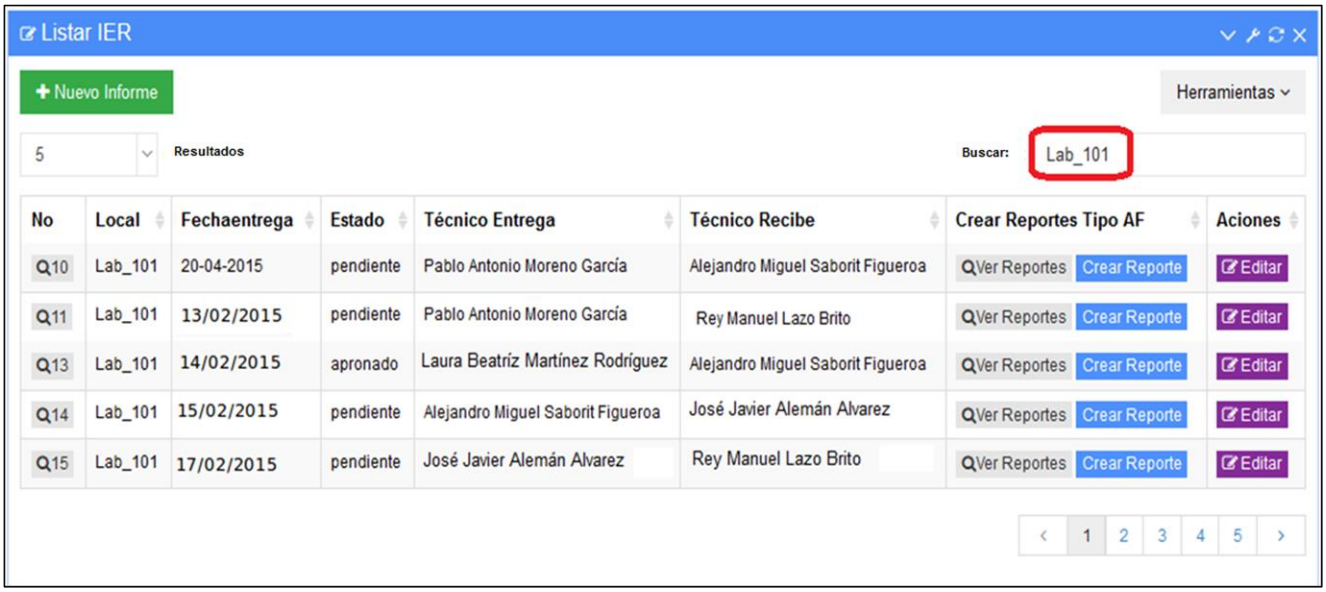


Ilustración 35. Listado de IER creados por local. Fuente: elaboración propia.

Búsqueda de Informes de Entrega y recibo de guardia por fecha:

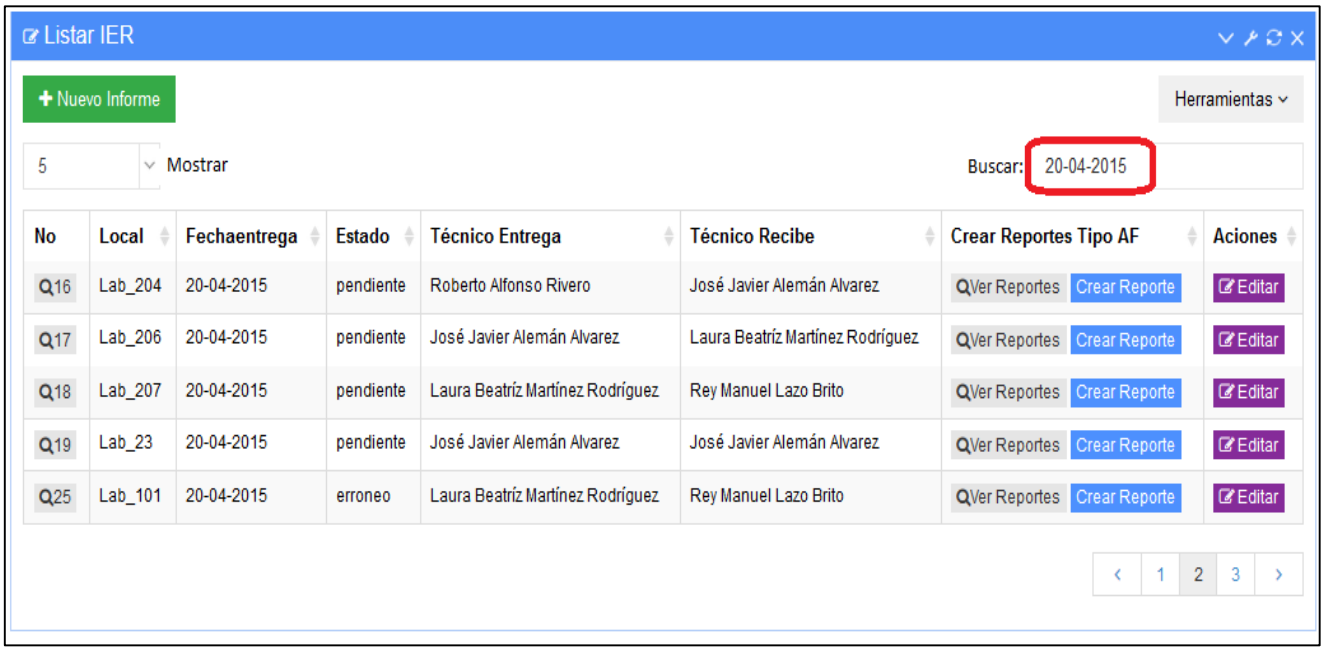


Ilustración 36. Listado de IER por fecha de creación. Fuente: elaboración propia.

Detalle de las Incidencias reportadas

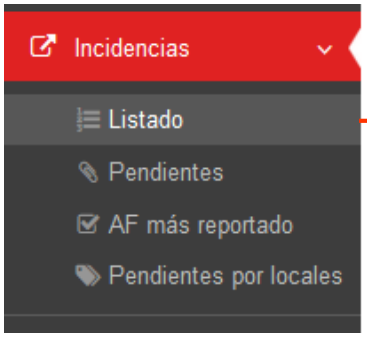


Ilustración 37. Listado de Incidencias. Fuente: elaboración propia.

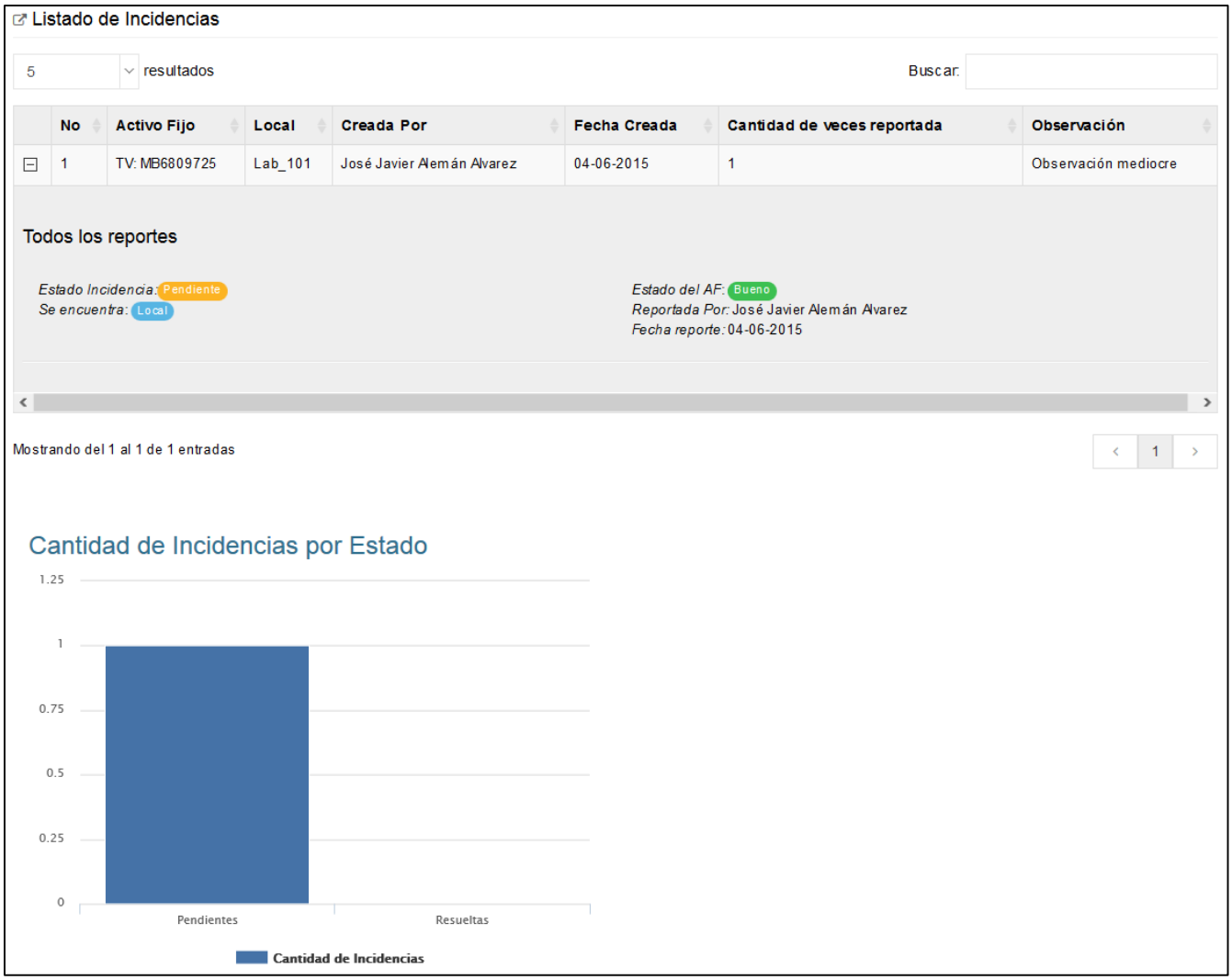


Ilustración 36. Listado de todas las incidencias reportadas. Fuente: elaboración propia.

Conclusiones parciales

- La utilización de los estándares de codificación *CamelCasing* y *PascalCasing* permitió estructurar y organizar el código logrando un lenguaje común y comprensible para todas las clases y métodos utilizados en el desarrollo del sistema.
- Las pruebas unitarias realizadas con la herramienta *PHPUnit***,** permitió probar el código, para corregir los errores detectados por dicha herramienta, obteniendo un código confiable que responde a los requisitos identificados.
- Se ejecutó un total de 40 diseños de casos de pruebas, corrigiendo por iteraciones las no conformidades encontradas y repitiendo las pruebas hasta lograr un estado 100% óptimo. Quedando demostrado que el sistema propuesto se encuentra listo para ser utilizado en el entorno real para el que fue concebido.

CONCLUSIONES

La investigación desarrollada y los resultados obtenidos permiten a los autores plantear las siguientes conclusiones:

- Las herramientas y tecnologías aplicadas durante el ciclo de vida del desarrollo de software haciendo uso de las buenas prácticas propuesta por la metodología, permitieron obtener varios artefactos por cada una de las fases, logrando una descripción detallada de los procesos internos de la entidad.
- La propuesta de solución fue validada mediante las pruebas unitarias realizadas con *PHPUnit*, las pruebas funcionales con los diseños de caso de pruebas y las pruebas de aceptación con el cliente, garantizando la calidad y compromiso con el cliente y promoviendo su utilización en el entorno real para el que fue concebida.
- El sistema **CERAF** (Control de Entrega y Recibo de Activos Fijos) permite el control de entrega y recibo de AF, las áreas, los locales así como los registros y detalles de los informes de guardia generados, además del manejo y tratamiento de las incidencias reportadas, demostrando que la propuesta de solución mejora el control sobre el proceso de entrega y recibo de AF en la Facultad 3.
- Los mecanismos de seguridad (Token de Seguridad y el Escáner de Solapín) implementados en el sistema, garantizan la autenticidad y no repudio de los informes de guardias creados por los técnicos de laboratorios a la hora de recibir los locales.

RECOMENDACIONES

- Hacer extensible uso del sistema de control de entrega y recibo de AF para todos los Departamentos de Tecnología de la universidad, por la rapidez que brinda a la hora de consultar información de interés sobre los medios, así como el control sobre todas las incidencias reportadas y las actas de control de AF para el cierre de los locales, tomando como base a la Facultad 3.
- Nomenclar las descripciones de las incidencias reportadas sobre los activos con el objetivo de poder realizar futuras búsquedas y conocer cuál es la causa que más rotura presenta.
- Terminación del mecanismo de autenticación biométrico para contribuir a garantizar la presencia física del responsable del local a la hora de realizar la entrega y el no repudio de los informes de guardia generados.
- Agregar la funcionalidad de consumo de ubicación de activos desde el ASSETS para lograr cambios dinámicos de ubicación, siendo estos permitidos por los administradores.
- Agregar la funcionalidad de exportar a PDF a otras acciones del sistema utilizadas por los administradores y usuario avanzados, tal es el caso de las funciones de agrupamiento y búsqueda de información importante dentro del sistema.

REFERENCIAS BIBLIOGRÁFICAS

1. FAM. La importancia de los activos fijos. [En línea] 11 de 2011. [Citado el: 9 de 12 de 2014.] http://floresamare.blogspot.com/2011/09/la-importancia-de-los-activos-fijos.html..

2. Contabilidad Financiera. [En línea] 2011. [Citado el: 09 de 12 de 2014.] https://prezi.com/lachcdr_iwn6/contabilidad-financiera.

3. debitoor. [En línea] 2012. [Citado el: 09 de 11 de 2014.] https://debitoor.es/glosario/activo-fijo..

4. Betsy Sanchez. Activos Fijos y Depreciacion. [En línea] 22 de 11 de 2014. [Citado el: 10 de 12 de 2014.] https://prezi.com/hbvnl1jqr4li/activos-fijos-y-depreciacion.

5. Alcofin. [En línea] 2012. [Citado el: 09 de 12 de 2014.] http://www.alcofin.com.mx/activos_fijos.html.

6. Active AF. [En línea] 2010. [Citado el: 09 de 12 de 2014.] http://controldeactivosfijos.com.mx/sistema-active-af.htm.

7. Corporation, LinkedIn. Introducción al sistema SAP R/3. *Introducción al sistema SAP R/3.* [En línea] 2014. [Citado el: 10 de 12 de 2014.] http://es.slideshare.net/americouriarte/introduccin-sistemas-sap-r3.

8. Versat_Sarasola. Versat_Sarasola. [En línea] Casa Consultora DISAIC, 2014. [Citado el: 09 de 12 de 2014.] http://www.disaic.cu/index.php.

9. Cimatel. Rodasxxi. *Rodasxxi.* [En línea] 2014. [Citado el: 12 de 12 de 2014.] http://www.rodasxxi.cu/rodasxxi.php.

10. S.A., D´MARCO. Sistema de Gestón Integral ASSETS. *Assets.* [En línea] 2014. [Citado el: 05 de 02 de 2015.] http://www.assets.co.cu.

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

12. Larman, Craig. *UML Y PATRONES.* Mexico : Prentice Hall, 1999.

13. Paradigm, Visual. Visual Paradigm. [En línea] 2012. [Citado el: 14 de 12 de 2014.] http:// www.visual-paradigm.com.

14. Moreno Navarro, Juan José. *Curso de Software basado en componentes.* 2014.

15. Eguiluz, Javier. *Desarrollo web ágil con Symfony 2.* 2013.

16. Symfony. *Symfony.* [En línea] [Citado el: 125 de 12 de 2014.] http://symfony.es/que-es-symfony.

17. Pressman, Roger S. *Ingeniería de Software. Un enfóque práctico.* [ed.] Concepción Fernández Madrid. España : Mc Graw Hill, 2005. Vol. I.

18. Patrones-de-Diseno-Decorador-Decorator. [En línea] 2014. [Citado el: 17 de 01 de 2015.] http://www.michael-pratt.com/blog/14/Patrones-de-Diseno-Decorador-Decorator.

19. PHP. *Hypertext Preprocessor.* [En línea] [Citado el: 15 de 12 de 2014.] http://php.net/.

20. Lenguajes de programación. *Kioskea.net.* [En línea] 2010. [Citado el: 12 de 12 de 2014.] http://es.kioskea.net/contents/304-lenguajes-de-programacion.

21. openwebcms. *openwebcms.* [En línea] [Citado el: 16 de 12 de 2014.] http://openwebcms.es/2013/que-es-bootstrap.

22. netbeans. *netbeans.* [En línea] [Citado el: 16 de 12 de 2014.] https://netbeans.org/features.

23. postgresql. *postgresql.* [En línea] [Citado el: 16 de 12 de 2014.] http://www.postgresql.org.es/sobre_postgresql.

24. GitLab. [En línea] [Citado el: 05 de 02 de 2015.] https://about.gitlab.com/about/.

25. Servidor HTTP apache. [En línea] The Apache Software Foundation., 1999-2015. [Citado el: 05 de 02 de 2015.] http://www.apache.org/foundation/license-faq.html#Marks.

26. daedalusNegocio. *Reglas de .* [En línea] Daedalus-Data, Decisions and Language, S. A. [Citado el: 10 de 03 de 2015.] http://www.daedalus.es/que-tecnologias-nos-diferencian/inteligencia-de-negocio/reglas-de-negocio/.

27. DEADELUS. Reglas de negocio. [En línea] 2013. [Citado el: 10 de 04 de 2015.] http://www.mountaingoatsoftware.com/topics/user-stories..

28. Jacobson, I, Booch, G y J, Rumbaugh. *EL PROCESO UNIFICADO DE DESARROLLO DE SOFTWARE.* s.l. : PERSON EDUCACIÓN.S.A.

29. Potencier, Ryan Weaver Fabien. *Symfony 2.4, el libro oficial.*

30. Ortega Paredes, José Grabiel . Contabilidad. Paradigma de reconstrucción a través del giro informático. [aut. libro] José Grabiel Ortega Paredes. *Contabilidad. Paradigma de reconstrucción a través del giro informático.* Madrid : EAE, 2012, pág. 33 a 37.

31. Base de Datos: Tercera Formal (3FN). [En línea] 29 de 08 de 2012. [Citado el: 16 de 05 de 2015.] http://tadebasegino.blogspot.com/2012/08/tercera-forma-normal-3fn.html.

32. Métrcias de Software. [En línea] 2011. [Citado el: 14 de 05 de 2015.] http://www.desarrolloweb.com/articulos-copyleft/articulo-metricas-de-software.html.

33. Pruebas de Software. *Gestión de la calidad y pruebas de Software.* [En línea] 2012. [Citado el: 02 de 02 de 2015.] http://pruebasdesoftware.com/laspruebasdesoftware.htm.

34. Ing. Alexander B Oré. Pruebas Unitarias. [En línea] 2014. [Citado el: 02 de 02 de 2015.] http://www.calidadysoftware.com/testing/pruebas_unitarias1.php.

35. etnassof. *PHPUnit Manual.* [En línea] 2015. [Citado el: 20 de 02 de 2015.] http://www.etnassoft.com/biblioteca/phpunit-manual/.

36. Pruebas de caja negra. [En línea] 2002. [Citado el: 02 de 20 de 2015.] http://www.globetesting.com/2012/08/pruebas-de-caja-negra.

GLOSARIO DE TÉRMINOS

A

Autenticidad: característica que se refiere a la comprobación y confirmación de la identidad real de los activos (procesos, sistemas, información) y/o actores (usuarios) y/o de la autorización por parte de los autorizadores, así como la verificación de estas tres cuestiones.

C

CMMI: Capability Maturity Model Integration o integración de modelos de madurez de capacidades. Es un modelo para la mejora y evaluación de procesos para el desarrollo, mantenimiento y operación de sistemas de software.

CSS: acrónimo de Cascading Style Sheets (Hojas de Estilo en Cascada), es un lenguaje usado para definir la presentación de un documento estructurado escrito en HTML o XML (y por extensión en XHTML). El W3C (World Wide Web Consortium) es el encargado de formular la especificación de las hojas de estilo que servirán de estándar para los agentes de usuario o navegadores.

H

HTML: siglas del inglés HyperText Markup Language. Es básicamente HTML expresado como XML válido. Es más estricto a nivel técnico, pero esto permite que posteriormente sea más fácil al hacer cambios o buscar errores entre otros.

I

IDE: Entorno de desarrollo integrado (en inglés Integrated Development Environment). Es un programa informático compuesto por un conjunto de herramientas de programación.

J

JavaScript: es un lenguaje de programación interpretado. Se define como orientado a objetos, basado en prototipos. Se utiliza principalmente en su forma del lado del cliente, implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas.

M

Multiplataforma: es un atributo conferido a programas informáticos o métodos y conceptos de cómputo que son implementados en múltiples plataformas informáticas. El software multiplataforma puede dividirse en dos tipos; uno requiere una compilación individual para cada plataforma que le da soporte, y el otro se puede ejecutar directamente en cualquier plataforma sin preparación especial.

N

No Repudio: Característica que permite garantizar la autoría de un mensaje y/o su envío.

S

Scanner de Solapín: se refiere a uno de los métodos de seguridad utilizados para controlar el proceso de entrega de guardia en los laboratorios docentes, garantizando la presencia física del responsable del local y el no repudio de los reportes generados.

Seguridad: la preservación de la confidencialidad, la integridad y la disponibilidad de la información, pudiendo además, abarcar otras propiedades, como la autenticidad, la responsabilidad, la fiabilidad y el no repudio.

Software libre: se refiere a la libertad de los usuarios para ejecutar, copiar, distribuir, estudiar, cambiar y mejorar el software.

T

Token de Seguridad: valor aleatorio generado a nivel de aplicación y asignado como valor único a cada trabajador para el control del proceso de entrega y recibo de los laboratorios docentes.

X

xUNit: conjunto de marcos de trabajo para el desarrollo de pruebas disponible para muchos lenguajes de programación y plataformas de pruebas.

ANEXOS

Anexo 1: Entrevista para la captura de requisitos

Objetivo: Identificar los principales requisitos funcionales y no funcionales que debería satisfacer el nuevo Sistema Informático de control del proceso de entrega y recibo de activos fijos.

Preguntas:

1. ¿Qué ventajas representa la utilización de un sistema informático para la gestión del proceso de entrega y recibo de activos fijos?
2. ¿Cuáles son las principales deficiencias en la realización de este proceso actualmente?
3. ¿Qué funciones le gustaría mejorar?
4. ¿Qué nuevas funciones le gustaría agregar?
5. ¿Cómo se identifican e informan las incidencias encontradas durante el proceso de guardia?
6. ¿Qué elementos se deben tener en cuenta para el control de los activos fijos?
7. ¿Cuándo se realizan las actas de control de activos fijos y quien es la persona encargada de controlar este proceso?
8. ¿Cómo se maneja el tratamiento de la información de los informes de guardia y las incidencias generadas diarias?
9. ¿Quiénes serán los usuarios del sistema?
10. ¿A qué funcionalidades tendrá acceso cada usuario?
11. ¿Cuáles son los principales reportes y notificación que debe tener el nuevo sistema?

Anexo 2: Historias de Usuario

Tabla 12. Historia de Usuario: Adicionar Área. Fuente: elaboración propia.

Adicionar Área	
Número: 1	**Nombre del requisito: Adicionar área**
Programador: Alejandro Miguel Saborit – José Javier Alemán	**Iteración Asignada:** 1
Prioridad: Media	**Tiempo Estimado:** -
Riesgo en Desarrollo: -	**Tiempo Real:**
Descripción: Ingresar una nueva área de control al sistema. El usuario registra la nueva área dando la ubicación donde esta se encuentra y una descripción de la misma. Luego el sistema la registra y la adiciona en la Base de datos como una nueva área.	
Observaciones:	

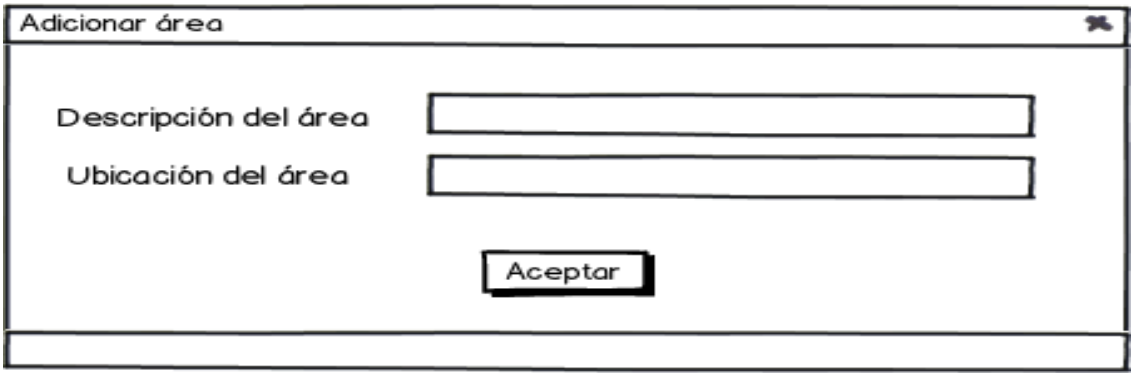
Adicionar Área
Prototipo de interfaz: 

Tabla 13. Historia de Usuario: Adicionar Elemento. Fuente: elaboración propia.

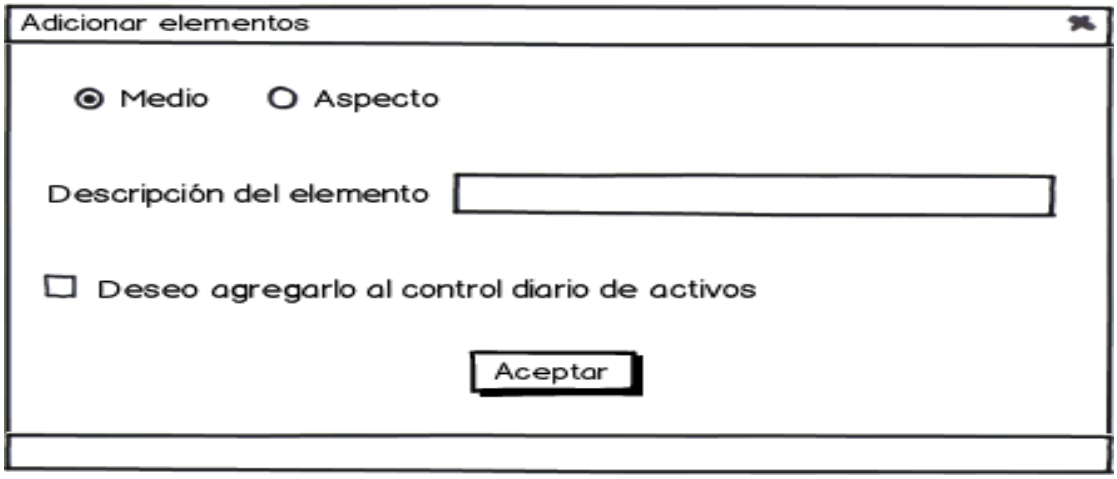
Adicionar Elemento	
Número: 2	**Nombre del requisito: Adicionar elemento**
Programador: Alejandro Miguel Saborit – José Javier Alemán	**Iteración Asignada:** 1
Prioridad: Media	**Tiempo Estimado:** -
Riesgo en Desarrollo: -	**Tiempo Real:**
Descripción: Pantalla para el control de Tipos de AF y AF.	
Observaciones:	
Prototipo de interfaz: 	

Tabla 14. Historia de Usuario: Adicionar Área. Fuente: elaboración propia.

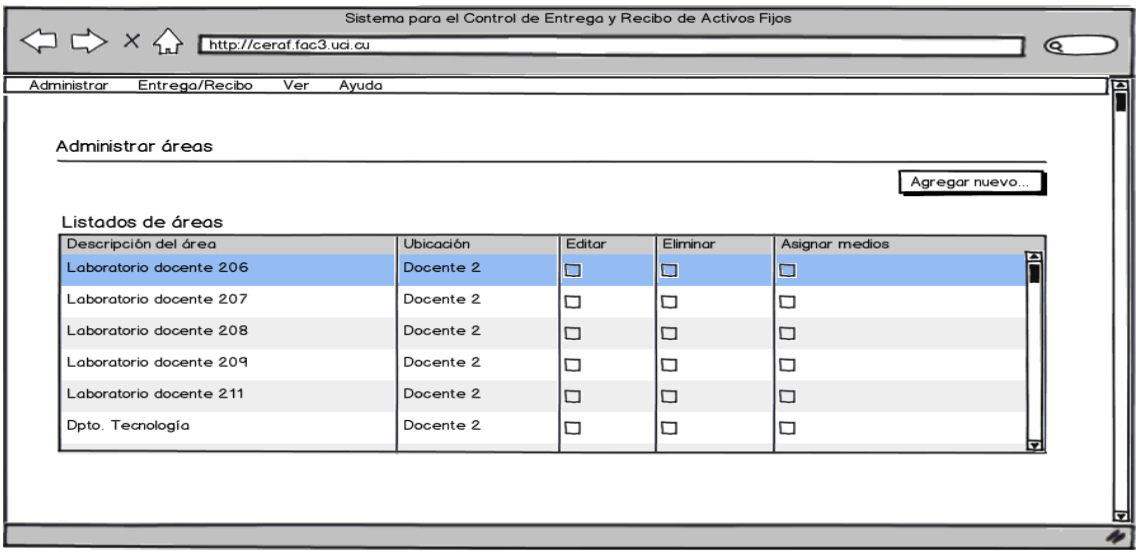
Administrar Área	
Número: 3	**Nombre del requisito: Administrar área**
Programador: Alejandro Miguel Saborit – José Javier Alemán	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo Estimado:** -
Riesgo en Desarrollo: -	**Tiempo Real:**
Descripción: Lista las áreas existentes en el sistema para poder realizar cualquier acción valida sobre ellas.	
Observaciones:	
Prototipo de interfaz: 	

Tabla 15. Historia de Usuario: Asignar Medios a Áreas. Fuente: elaboración propia.

Asignar Medios a Áreas	
Número: 4	**Nombre del requisito: Asignar Medios a Áreas**
Programador: Alejandro Miguel Saborit – José Javier Alemán	**Iteración Asignada:** 2
Prioridad: Alta	**Tiempo Estimado:** -
Riesgo en Desarrollo:	**Tiempo Real:**
Descripción: Selección y control de medios por Áreas. Se selecciona todos los medios que se desean asignar a una determinada área para su futuro empleo. El sistema registra el cambio y actualiza la Base de datos.	

Asignar Medios a Áreas
Observaciones:
Prototipo de interfaz: 

Tabla 16. Historia de Usuario: IER. Fuente: elaboración propia.

IER	
Número: 5	**Nombre del requisito: Asignar Medios a Áreas**
Programador: Alejandro Miguel Saborit – José Javier Alemán.	**Iteración Asignada:** 2
Prioridad: Alta	**Tiempo Estimado:** -
Riesgo en Desarrollo: -	**Tiempo Real:**
Descripción: Visualización de los datos levantados en el proceso de entrega de guardia, así como las incidencias y el personal implicado en el proceso.	
Observaciones:	

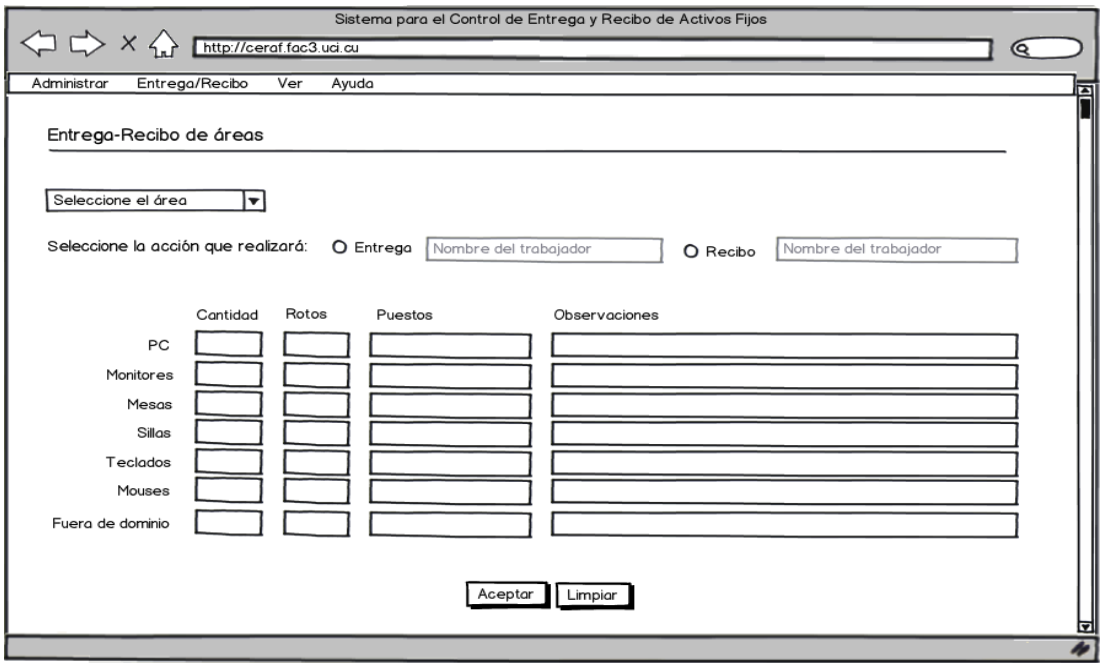
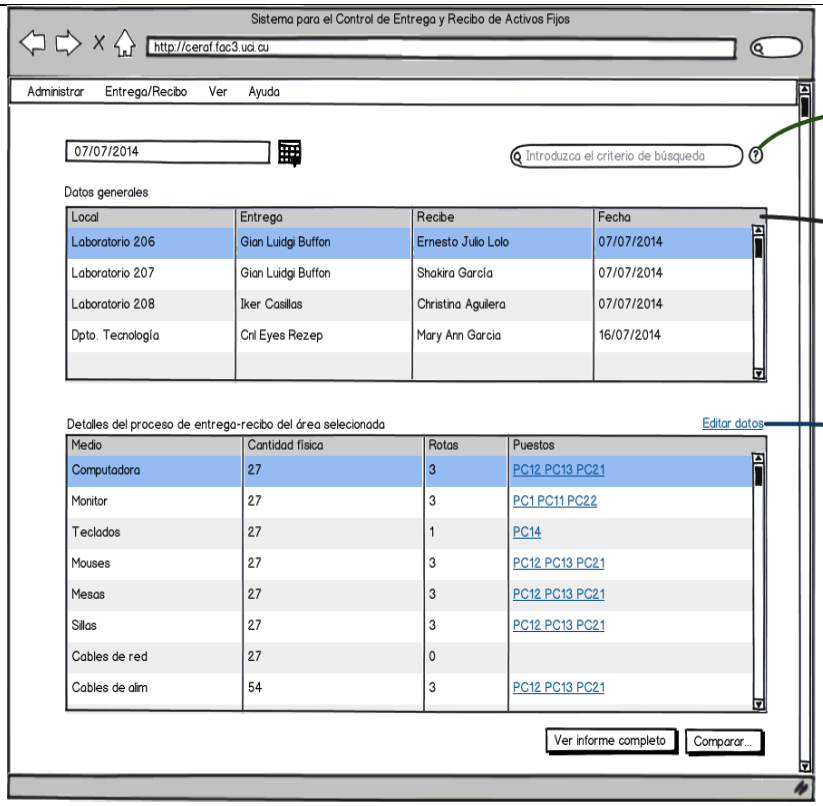
IER
Prototipo de interfaz: 

Tabla 17. Historia de Usuario: Inventario diario. Fuente: elaboración propia.

Inventario diario	
Número: 6	**Nombre del requisito: Asignar Medios a Áreas**
Programador: Alejandro Miguel Saborit – José Javier Alemán.	**Iteración Asignada:** 2
Prioridad: Alta	**Tiempo Estimado:** -
Riesgo en Desarrollo: -	**Tiempo Real:**
Descripción: Control estricto por valores de cantidad de los medios disponible para la entrega de guardia.	
Observaciones:	
Prototipo de interfaz:	

Inventario diario



Anexo 3: Diagramas de Clases del diseño

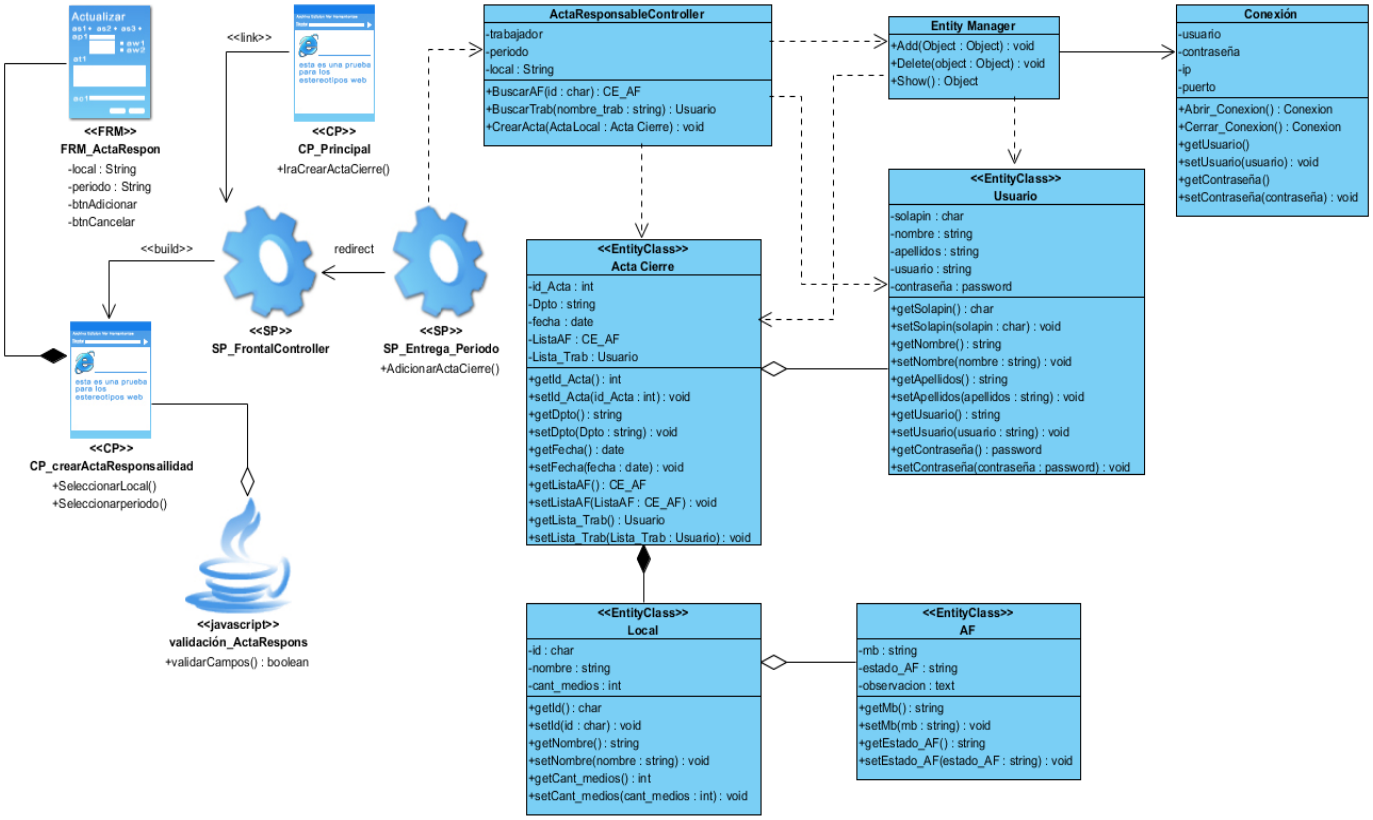


Ilustración 38. DCD Acta de Responsabilidad. Fuente: elaboración propia.

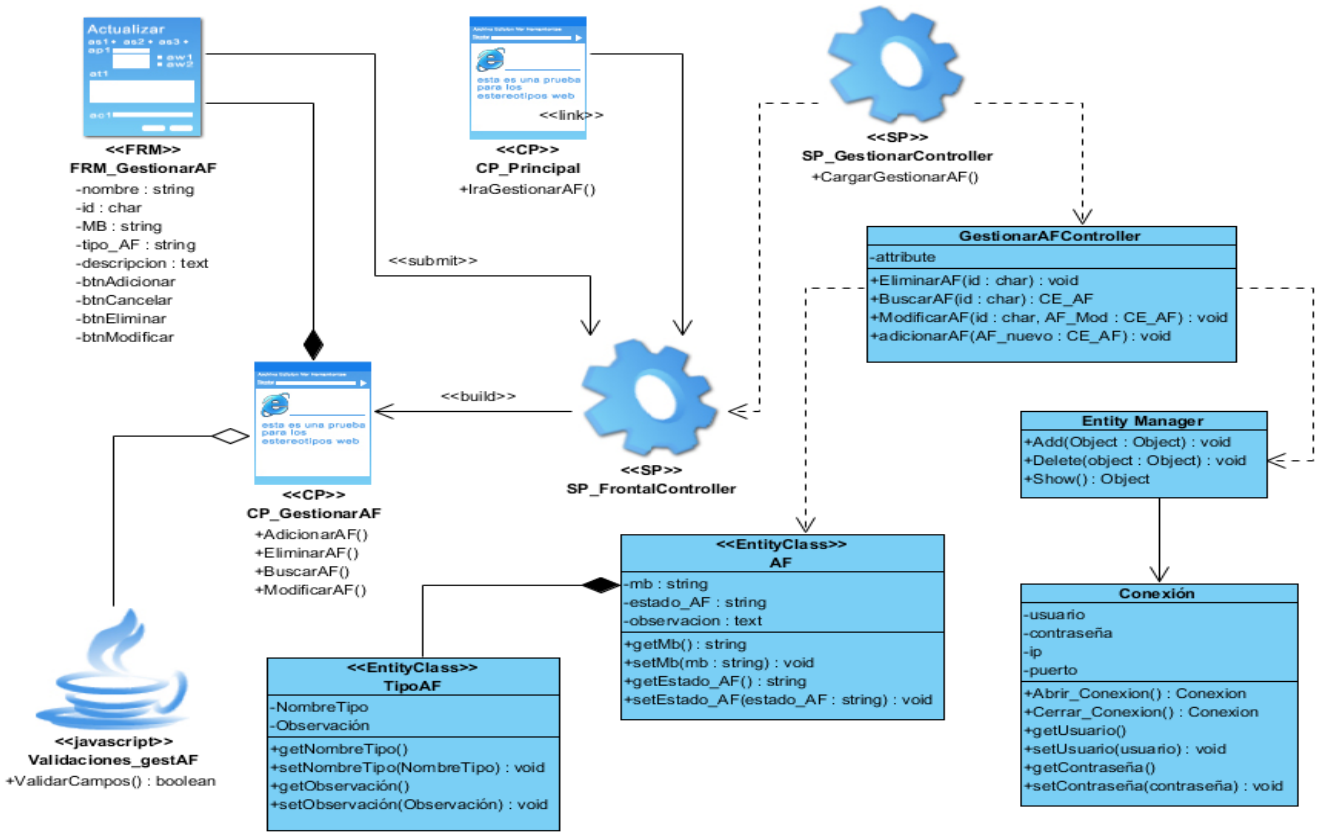


Ilustración 39. DCD: Gestionar AF. Fuente: elaboración propia.

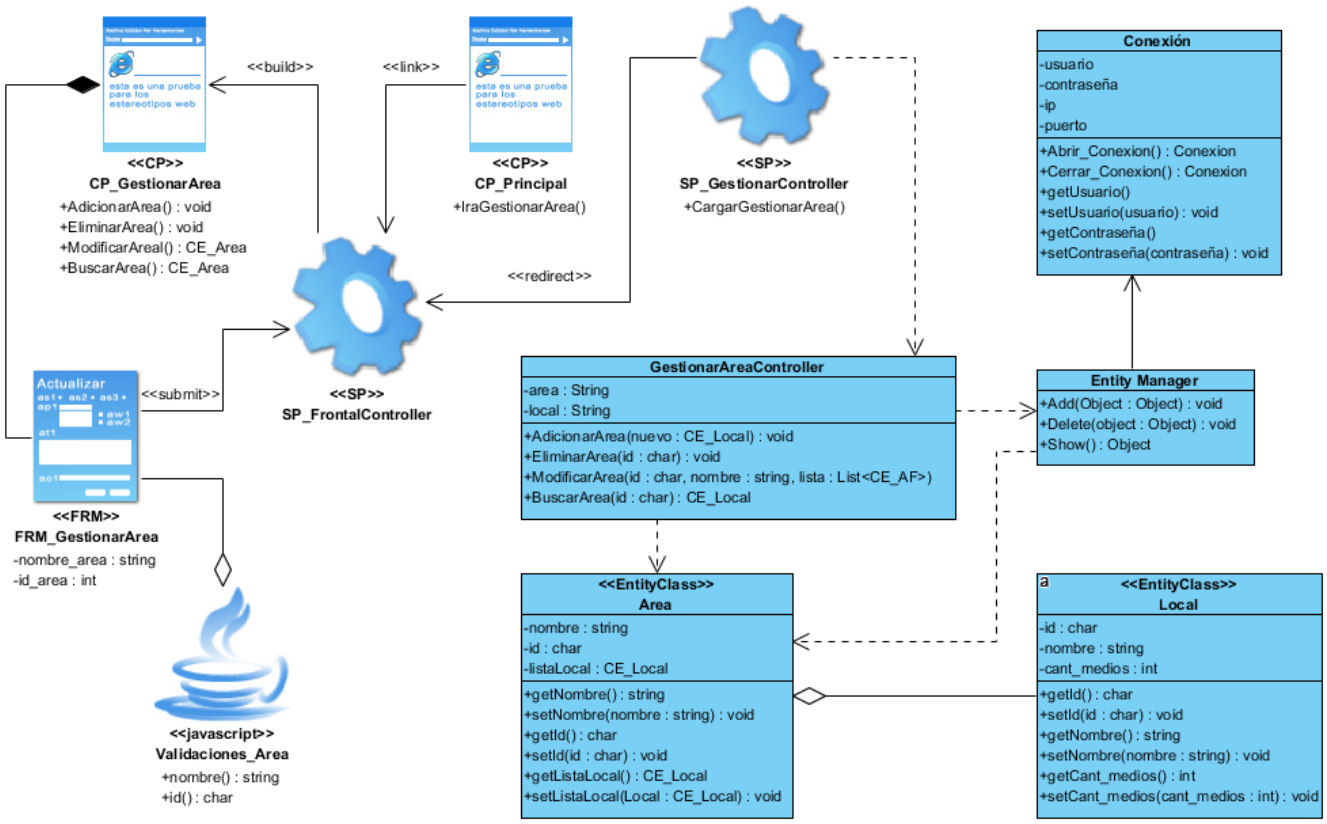


Ilustración 40. DCD Gestionar Área. Fuente: elaboración propia.

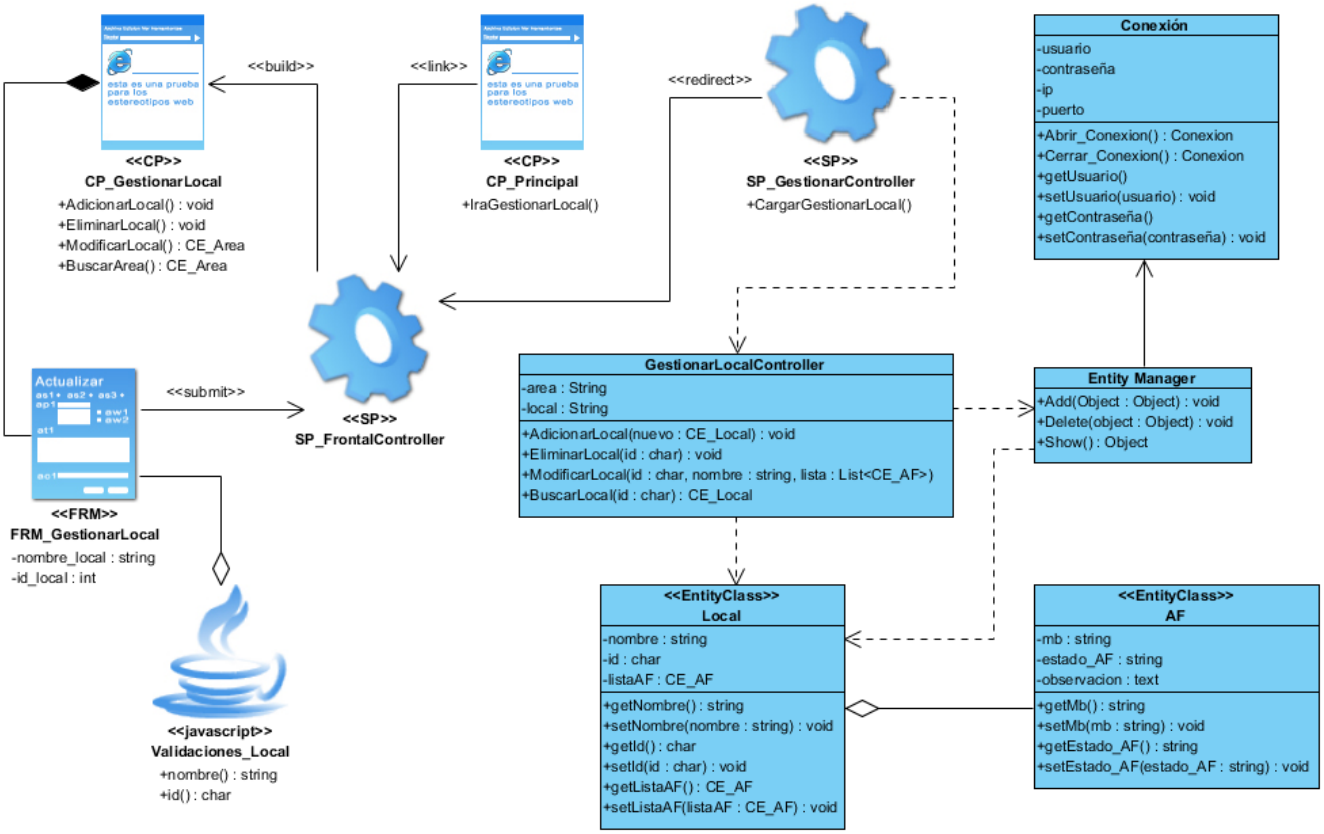


Ilustración 41. DCD Gestionar Local. Fuente: elaboración propia.

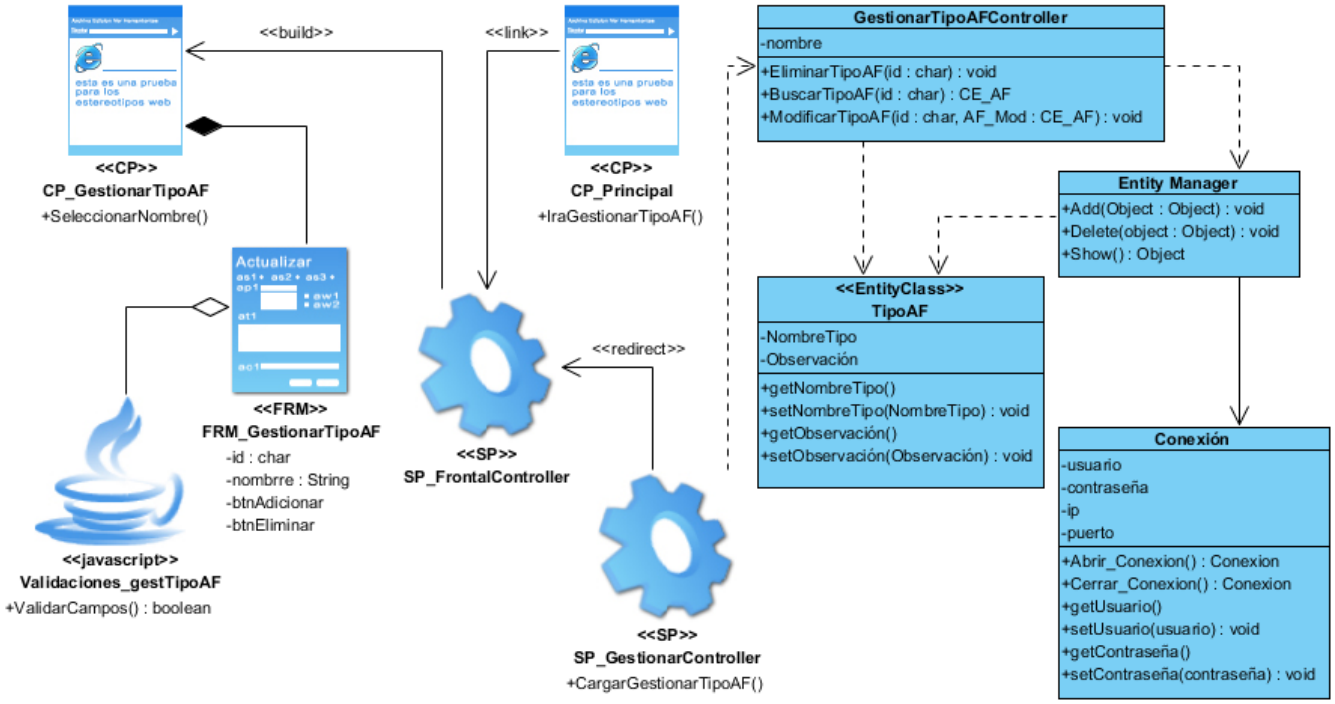


Ilustración 42. DCD Gestionar Tipo AF. Fuente: elaboración propia.

Anexo 4: Diagramas de Casos de Pruebas (DCP)

1. *DCP: Crear Acta de cierre*

Condiciones de ejecución:

- El usuario debe tener ser responsable de un local para poder crear el Acta de cierre de los medios de su local.
- Debe especificar el local y el periodo de cierre al cual está asociada el acta.
- No debe existir en el sistema un acta generada para el mismo local en el mismo periodo de cierre.

Tabla 18. DCP: Adicionar Acta de Cierre. Fuente: elaboración propia.

Escenario	Descripción	Periodo	Local	Respuesta del sistema	Flujo central
EC 1.1: Crear Acta de Cierre introduciendo datos válidos.	Se adiciona los datos del Acta de cierre introduciendo datos válidos.	V Vacaciones	V Aula 202	El sistema adiciona el acta de cierre y automáticamente carga todos los medios de ese local para editar las incidencias reportadas.	•Se selecciona la opción Cierre por periodo en el panel de opciones. •Se presiona en la opción de crear acta de cierre. •Se introducen los datos. •Se presiona el botón adicionar. •Al registrar los datos el sistema muestra una tabla con todos los medios presentes en el local seleccionado.
EC 1.2: Adicionar Acta de Cierre introduciendo un Local y un periodo al cual ya se le ha generado un acta de cierre	Se adiciona los datos del Acta de Cierre introduciendo datos inválidos.	V Vacaciones	V Aula 202	El sistema no permite introducir estos datos. Se muestra el mensaje de error "Ya existe un acta de cierre para este local en este periodo"	• Se introducen los datos a insertar. •Se presiona el botón adicionar •El sistema muestra un mensaje de error y no se adicionan los datos al sistema.
EC 1.3: Adicionar acta de cierre dejando campos vacíos.	Se adiciona los datos dejando campos obligatorios vacíos.	I Vacío	V Vació	El sistema marca en rojo el campo mostrando el mensaje: "Este campo es obligatorio."	• Se presiona el botón adicionar. •El sistema muestra un mensaje de error y no se adicionan los datos al sistema.

Tabla 19. Descripción de las variables del DCP: Adicionar Acta de cierre. Fuente: elaboración propia.

No	Nombre de campo	Clasificación	Valor Nulo	Descripción
1	Local	Lista desplegable	No	Selecciona el local.
2	Periodo de cierre	Lista desplegable	No	Selecciona el periodo de cierre asociado.

2. *DCP: AF adicionar:*

Condiciones de ejecución

- Se debe haber introducido el MB, el estado del medio y el tipo de AF.
- Debe existir al menos un tipo de AF.
- Se debe seleccionar la opción adicionar AF en el formulario de gestionar AF.

Tabla 20. DCP: Adicionar AF. Fuente: elaboración propia.

Escenario	Descripción	MB	Estado AF	Observ	Tipo AF	Respuesta del sistema	Flujo Central
EC 1.1 Adicionar AF introduciendo datos válidos	Se adiciona los datos del nuevo medio introduciendo valores válidos.	V	V	N/A	V	El sistema adiciona el nuevo medio y notifica que se insertó correctamente	Se introduce el MB, el estado del medio y la observación del nuevo medio. Se presiona el botón adicionar. El sistema muestra el mensaje "Medio adicionado correctamente".
		MB123	Roto		Silla		
EC 1.2 Adicionar AF introduciendo datos inválidos	Se adiciona los datos del nuevo medio introduciendo al menos un valor inválido.	I	V	N/A	V	El sistema no permite introducir datos incorrectos. Muestra un mensaje de error.	Se introducen los nuevos valores. Se presiona el botón adicionar. Se muestra un mensaje de error los datos no se adicionan.
		A@121	Roto		Silla		
EC 1.3 Adicionar AF dejando al menos un campo vacío.	Se adicionan los datos del nuevo medio dejando al menos un campo de los obligatorios vacío.	I	V	N/A	I	El sistema marca en rojo el campo mostrando el mensaje: "Este campo es obligatorio."	Se introducen los nuevos valores. Se presiona el botón adicionar. El sistema muestra el mensaje de error "campo obligatorio" y no adiciona los valores al sistema. Se presiona el botón cancelar.
		Vacío	Roto				
EC 1.4 Cancelar	Se cancela la operación	N/A	N/A	N/A	N/A	El sistema cierra la interfaz sin realizar ninguna operación	

Tabla 21. Descripción de las variables del DCP: Adicionar AF. Fuente: elaboración propia.

No	Nombre de campo	Clasificación	Valor Nulo	Descripción
1	MB	Campo de texto	No	El valor del MB es único comienza por las letras MB y sigue de 5 números.
2	Estado AF	Campo de selección	No	Los valores que toma son (correcto o roto).
3	observación	Campo de texto	Si	Texto.
4	Tipo AF	Campo de selección	No	Selecciona según el tipo de AF que pertenezca.

3. *DCP: Editar AF:*

Condiciones de ejecución

- El usuario registrado debe tener los permisos adecuado para realizar esta opción.
- Debe existir al menos un AF registrado en el sistema.
- Debe seleccionar la opción editar que tiene asociada cada AF.

Tabla 22. DCP: Editar AF. Fuente: elaboración propia.

Escenario	Descripción	MB	Estado AF	Observ	Tipo AF	Respuesta del sistema	Flujo Central
EC 1.1 Editar AF introduciendo datos válidos	Se adiciona los datos del nuevo medio introduciendo valores válidos.	V	V	N/A	V	El sistema actualiza el nuevo medio y muestra los cambios creados en una tabla	Se introduce el MB, el estado del medio y la observación del nuevo medio. Se presiona el botón actualizar. El sistema muestra la tabla con todos los datos actualizados.
		MB12	Roto		Silla		
EC 1.2 Editar AF introduciendo datos inválidos	Se adiciona los datos a modificar del medio introduciendo al menos un valor inválido.	I	V	N/A	V	El sistema no permite introducir datos incorrectos. Muestra un mensaje de error.	Se introducen los nuevos valores. Se presiona el botón adicionar. Se muestra un mensaje de error. Los datos no se actualizan.
		A@12	Roto		Silla		
EC 1.3 Editar AF dejando al menos un campo vacío.	Se adicionan los datos del nuevo medio dejando al menos un campo de los obligatorios vacío.	I	V	N/A	I	El sistema marca en rojo el campo mostrando el mensaje: "Este campo es obligatorio."	Se introducen los nuevos valores. Se presiona el botón adicionar. Se muestra un mensaje de error.
		Vacío	Roto				
EC 1.4 Cancelar	Se cancela la operación	N/A	N/A	N/A	N/A	El sistema cierra la interfaz sin realizar ninguna operación	Se presiona el botón cancelar.

Tabla 23. Descripción de las variables del DCP: Editar AF. Fuente: elaboración propia.

No	Nombre de campo	Clasificación	Valor Nulo	Descripción
1	MB	Campo de texto	No	El valor del MB es único comienza por las letras MB y sigue de 5 números.
2	Estado AF	Campo de selección	No	Los valores que toma son (correcto o roto).
3	observación	Campo de texto	Si	Texto.
4	Tipo AF	Campo de selección	No	Selecciona según el tipo de AF que pertenezca.

4. *DCP: Adicionar Local:*

Condiciones de ejecución

- El usuario registrado debe tener los permisos adecuado para realizar esta opción.
- Debe existir al menos un área registrada en el sistema.

Tabla 24. DCP: Adicionar Local. Fuente: elaboración propia.

Escenario	Descripción	Nombre	Descripción	Área	Respuesta del sistema	Flujo central
EC 1.1: Adicionar local introduciendo datos válidos.	Se adiciona los datos del local introduciendo datos válidos.	V Lab_101	N/A	V Área	El sistema adiciona el local y muestra el mensaje de información: “Local adicionado satisfactoriamente.” El sistema cierra la interfaz.	• Se introducen los datos a insertar • Se presiona el botón adicionar • El sistema muestra el mensaje "local adicionada correctamente".
EC 1.2: Adicionar local introduciendo datos inválidos.	Se adiciona los datos del local introduciendo datos inválidos.	I a/Ñ1332	N/A	I /yt%46	El sistema no permite introducir datos incorrectos. Se muestra un mensaje de error.	• Se introducen los datos a insertar. • Se presiona el botón adicionar. • El sistema muestra un mensaje de error y no se adicionan los datos al sistema.
EC 1.3: Adicionar área dejando campos vacíos.	Se adiciona los datos del local dejando campos obligatorios vacíos.	I Vacío	N/A	I	El sistema marca en rojo el campo mostrando el mensaje: “Este campo es obligatorio.”	• Se presiona el botón adicionar. • El sistema muestra un mensaje de error y no se adicionan los datos al sistema.

Tabla 25. Descripción de las variables del DCP: Adicionar Local. Fuente: elaboración propia.

No	Nombre de campo	Clasificación	Valor Nulo	Descripción
1	Nombre	Campo de texto	No	Texto.
2	descripción	Campo de texto	Si	Texto.
3	Área	Lista desplegable	No	Se selecciona el área al cual pertenece dicho local.

5. *DCP: Crear Incidencia:*

Condiciones de ejecución

- Debe existir al menos un AF registrado en el sistema.
- Debe seleccionar un medio al cual reportar la nueva incidencia.

Tabla 26. DCP: Crear Incidencia. Fuente: elaboración propia.

Escenario	Descripción	Acción medio	Obser.	Estado del AF	Respuesta del sistema	Flujo central
EC 1.1: Adicionar introduciendo datos válidos.	Se adiciona los datos introduciendo datos válidos.	V	N/A	V	El sistema adiciona la incidencia automáticamente carga los datos del AF reportado.	•Se presiona en la opción de crear incidencia. • Se introducen los datos. •Se presiona el botón adicionar. • Al registrar los datos el sistema muestra una tabla con todos los datos del medio al cual se le creo la incidencia.
		Extraído		Roto		
EC 1.2: Adicionar dejando campos vacíos.	Se adiciona los datos dejando campos obligatorios vacíos.	I	N/A	I	El sistema marca en rojo el campo mostrando el mensaje: “Este campo es obligatorio.”	• Se presiona el botón adicionar. • El sistema muestra un mensaje de error y no se adicionan los datos al sistema.
		Vacío				

Tabla 27. Descripción de las variables del DCP: Crear Incidencia. Fuente: elaboración propia.

No	Nombre de campo	Clasificación	Valor Nulo	Descripción
1	Acción medio	Lista de selección	No	Selección la acción realizada sobre el medio (Extraído, Local).
2	observación	Campo de texto	Si	Texto.
3	Estado del AF	Lista de selección	No	Selecciona el estado en el que se encuentra el medio seleccionado (Roto Buen Estado).

6. *DCP: Adicionar Modelo:*

Condiciones de ejecución

- Debe existir al menos un periodo de cierre en el sistema.

Tabla 28. DCP: Adicionar Modelo. Fuente: elaboración propia.

Escenario	Descripción	Periodo de cierre	Respuesta del sistema	Flujo Central
EC 1.1 Adicionar introduciendo datos válidos	Se adiciona los datos del nuevo modelo introduciendo valores válidos.	V Vacaciones	El sistema adiciona el nuevo modelo y automáticamente carga todas las actas de cierre generadas para ese periodo	Se introduce el dato solicitado. Se presiona el botón adicionar. El sistema adiciona el modelo y muestra todas las actas creadas en ese periodo.
EC 1.2 Adicionar introduciendo un periodo para el cual existe ya un modelo creado	Se adiciona los datos introduciendo un periodo al cual se le ha generado ya un acta de cierre (mismo año)	V Vacaciones	El sistema no permite introducir datos incorrectos. Muestra un mensaje de error.	Se introducen los nuevos valores. Se presiona el botón adicionar. Se muestra un mensaje de error. Los datos no se adicionan.
EC 1.3 Adicionar dejando campo vacío.	Se adicionan los datos del nuevo modelo dejando el campo de periodo vacío	I Vacío	El sistema marca en rojo el campo mostrando el mensaje: “Este campo es obligatorio.”	Se introducen los nuevos valores. Se presiona el botón adicionar. Se muestra un mensaje de error.
EC 1.4 Cancelar	Se cancela la operación	N/A	El sistema cierra la interfaz sin realizar ninguna operación	Se presiona el botón cancelar.

Tabla 29. Descripción de las variables del DCP: Adicionar Modelo. Fuente: elaboración propia.

No	Nombre de campo	Clasificación	Valor Nulo	Descripción
1	Periodo de cierre	Lista de selección	No	Selecciona el periodo de cierre al cual pertenece el modelo.

7. *DCP: Adicionar Usuario:*

Condiciones de ejecución

- Se debe buscar la persona por su usuario en la UDDI y la misma le devuelve los campos nombre, apellidos, usuario, correo, Solapín.
- Se debe introducir un rol al usuario adicionado.

Tabla 30. DCP: Adicionar Usuario. Fuente: elaboración propia.

Escenario	Descripción	Rol	Respuesta del sistema	Flujo central
EC 1.1: Adicionar usuario	Se adiciona el rol al usuario.	V	El sistema adiciona el rol al usuario seleccionado y	Se presiona el botón adicionar usuario.

Escenario	Descripción	Rol	Respuesta del sistema	Flujo central
introduciendo datos válidos.		Administrador	muestra el mensaje de información: “rol asignado correctamente.” El sistema cierra la interfaz.	El sistema muestra una interfaz para introducirle el usuario de la persona a buscar en la UDDI. Se presiona el botón buscar. El sistema devuelve los datos del usuario buscado. Se introduce el rol del usuario. Se presiona el botón adicionar. El sistema muestra el mensaje "usuario adicionado correctamente".
EC 1.2: Adicionar usuario dejando campos vacíos.	Se adiciona los datos del usuario dejando campos obligatorios vacíos.	I Vacío	El sistema marca en rojo el campo mostrando el mensaje: “Este campo es obligatorio.”	Se presiona el botón adicionar. El sistema muestra el mensaje "Campo obligatorio".
EC 1.3: Adicionar usuario a partir de un usuario que no existe en la UDDI.	Se adiciona los datos del usuario dejando campos obligatorios vacíos.	I Vacío	El sistema marca en rojo el campo mostrando el mensaje: “Este campo es obligatorio.”	Se presiona el botón adicionar. El sistema muestra el mensaje "Campo obligatorio".

Tabla 31. Descripción de las variables del DCP: Adicionar Usuario. Fuente: elaboración propia.

No	Nombre de campo	Clasificación	Valor Nulo	Descripción
1	Nombre	Campo de texto	No	texto

8. *DCP: Adicionar Reporte Tipo AF:*

Condiciones de ejecución

- Debe existir al menos un IER en el sistema.
- Debe existir al menos un tipo de AF registrado en el sistema.
- Debe existir al menos un AF asociado a un IER.

Tabla 32. DCP: Adicionar Reporte Tipo AF. Fuente: elaboración propia.

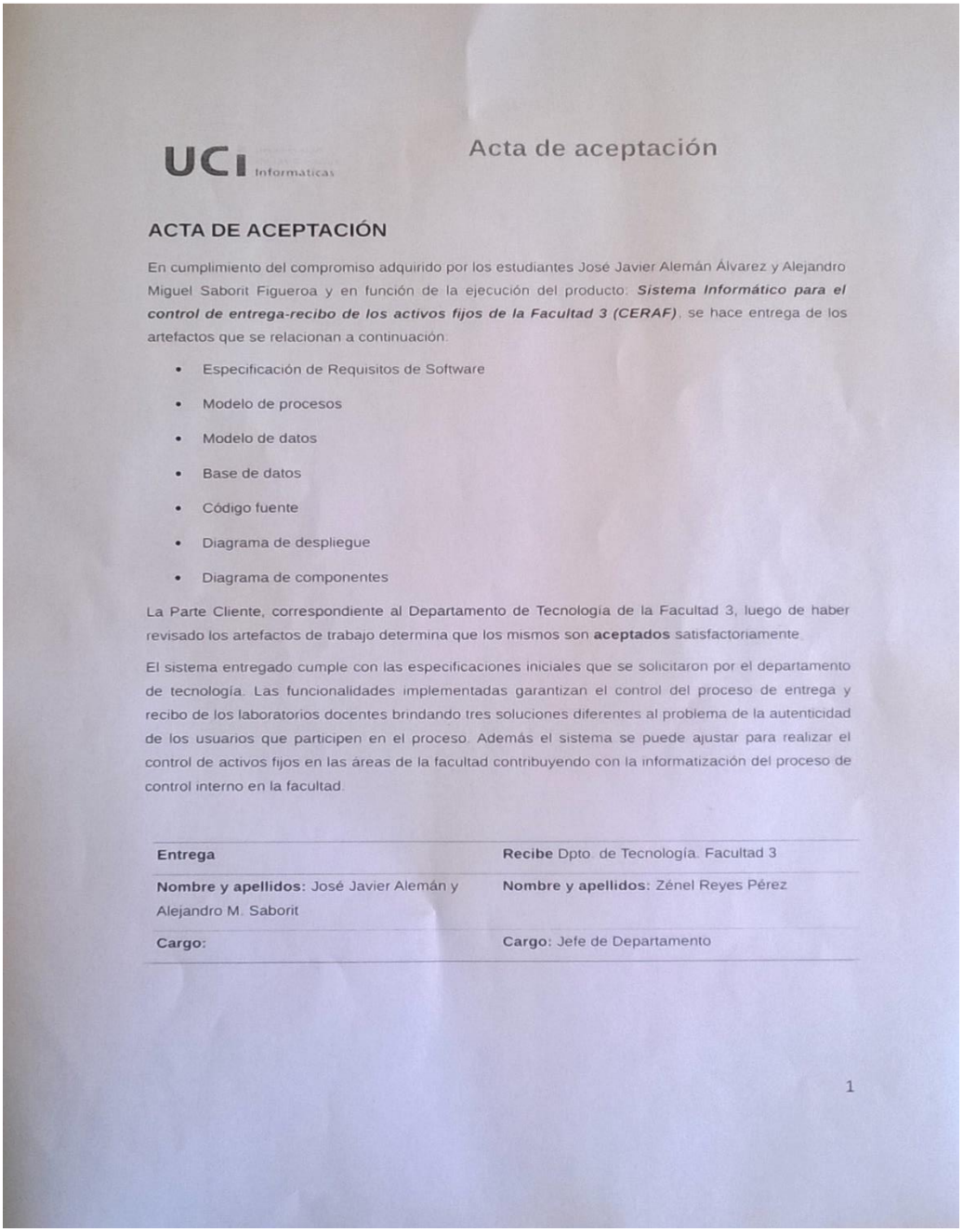
Escenario	Descripción	Tipo AF	Respuesta del sistema	Flujo central
EC 1.1: Adicionar Reporte Tipo AF introduciendo datos válidos.	Se adiciona los datos del Reporte Tipo AF introduciendo datos válidos.	V silla	El sistema adiciona el Reporte Tipo AF y muestra el mensaje de información: “El Reporte Tipo AF ha sido adicionado satisfactoriamente.” El sistema cierra la interfaz.	Se introducen los datos a insertar. Se presiona el botón adicionar. El sistema muestra el mensaje "Reporte Tipo AF adicionado correctamente".
EC 1.3: Adicionar Reporte Tipo AF dejando campos vacíos.	Se adiciona los datos del Reporte Tipo AF dejando campos obligatorios vacíos.	I Vacío	El sistema marca en rojo el campo mostrando el mensaje: “Este campo es obligatorio.”	Se presiona el botón adicionar. El sistema muestra un mensaje de error y no se adicionan los datos al sistema.

Tabla 33. Descripción de las variables del DCP: Adicionar Reporte Tipo AF. Fuente: elaboración propia.

No	Nombre de campo	Clasificación	Valor Nulo	Descripción
1	Tipo AF	Lista desplegable	No	Selecciona el tipo de AF.

Anexo 5: Carta de Aceptación del Cliente

Frente del acta:



UCI Informáticas

Acta de aceptación

ACTA DE ACEPTACIÓN

En cumplimiento del compromiso adquirido por los estudiantes José Javier Alemán Álvarez y Alejandro Miguel Saborit Figueroa y en función de la ejecución del producto: ***Sistema Informático para el control de entrega-recibo de los activos fijos de la Facultad 3 (CERAF)***, se hace entrega de los artefactos que se relacionan a continuación:

- Especificación de Requisitos de Software
- Modelo de procesos
- Modelo de datos
- Base de datos
- Código fuente
- Diagrama de despliegue
- Diagrama de componentes

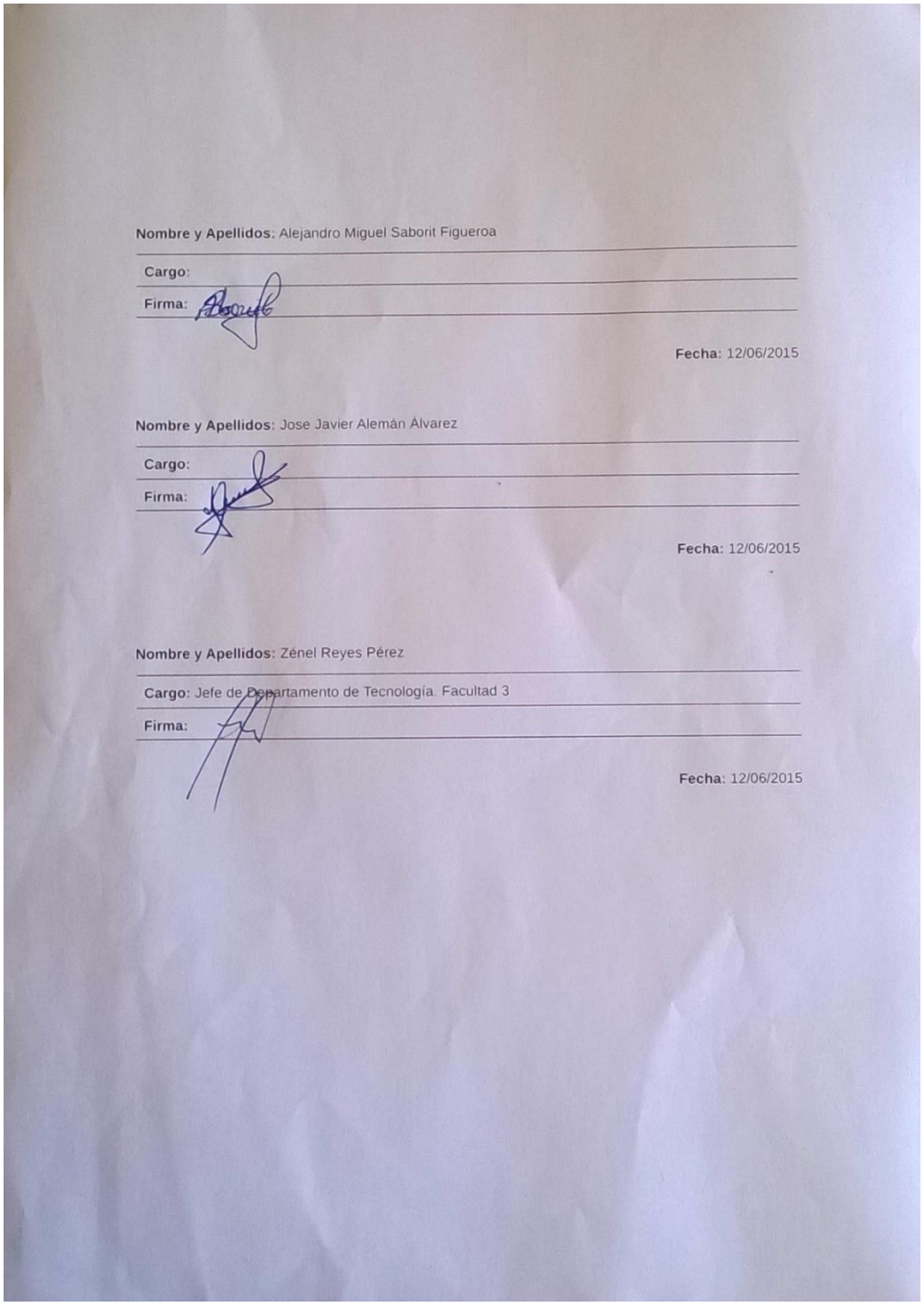
La Parte Cliente, correspondiente al Departamento de Tecnología de la Facultad 3, luego de haber revisado los artefactos de trabajo determina que los mismos son **aceptados** satisfactoriamente.

El sistema entregado cumple con las especificaciones iniciales que se solicitaron por el departamento de tecnología. Las funcionalidades implementadas garantizan el control del proceso de entrega y recibo de los laboratorios docentes brindando tres soluciones diferentes al problema de la autenticidad de los usuarios que participen en el proceso. Además el sistema se puede ajustar para realizar el control de activos fijos en las áreas de la facultad contribuyendo con la informatización del proceso de control interno en la facultad.

Entrega	**Recibe** Dpto. de Tecnología. Facultad 3
Nombre y apellidos: José Javier Alemán y Alejandro M. Saborit	**Nombre y apellidos:** Zénel Reyes Pérez
Cargo:	**Cargo:** Jefe de Departamento

1

Dorso del acta:



Nombre y Apellidos: Alejandro Miguel Saborit Figueroa

Cargo:

Firma:

Fecha: 12/06/2015

Nombre y Apellidos: Jose Javier Alemán Álvarez

Cargo:

Firma:

Fecha: 12/06/2015

Nombre y Apellidos: Zénel Reyes Pérez

Cargo: Jefe de Departamento de Tecnología. Facultad 3

Firma:

Fecha: 12/06/2015