Universidad de las Ciencias Informáticas

Facultad 1

"Componente para la Gestión de la Guardia Obrera Estudiantil en la Universidad de las Ciencias Informáticas"

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

Autores:

Liadelis Jones Perez.

Raúl Bach Eng.

Tutores:

Ing. Laritza González Marrero.

Ing. Mariela Milagros Bony Fernández.

Ing. Dariel Corchado López del Castillo.

La Habana, Cuba

Junio, 2016

"Calidad significa hacer las cosas bien incluso cuando nadie te está mirando".

Henry Ford

Declaración de Autoría

Declaramos que somos los únicos autores de este trabajo de diploma y conferimos a la Universidad de las Ciencias Informáticas (UCI) los derechos patrimoniales del mismo, con carácter exclusivo.
Para que así conste firmamos los presentes a los ___ días del mes de _____ del año ____.

_____

Liadelis Jones Perez.
Autor

_____

Raúl Bach Eng.
Autor

_____

Ing. Mariela Milagros Bony Fernández.
Tutor

_____

Ing. Laritza González Marrero.
Tutor

_____

Ing. Dariel Corchado López del Castillo.
Tutor

Agradecimientos

Agradezco infinitamente a Dios por darme la dicha de tener conmigo a mi adorada abuelita quien ha guiado mis pasos en cada minuto de mi vida.

A mi mamá por ser la luz de mi vida, por sus grandes consejos, por su dedicación y por albergar siempre ese deseo de verme cumplir mis sueños, por su apoyo incondicional y su transparente amor.

A mi padrastro por ser la persona que me ha visto crecer desde que tenía apenas 6 años, por su esfuerzo, dedicación y su compromiso de convertirme en una persona de bien y preparada para el futuro.

A mi familia, dígase tías y primas más cercanas por su preocupación en cada instante de esta etapa de mi vida.

A mi ex -pareja porque considero que a pesar de todo estuvo ahí apoyándome en los primeros momentos de este gran proyecto.

A mi compañero de tesis por ser una persona perseverante, comprometida, por su motivación y el esfuerzo entregado en cada momento.

Agradezco a mis tutoras por su entrega, dedicación y paciencia durante toda esta travesía.

A las profesoras Dianela, Arlennys, Anisleidis y Mercedes por su disposición y tiempo empleado.

A mis compañeros de los años, en especial a Leydys, Yoniel, Annayera, Yani, Osneidis por sus buenos consejos y ayuda incondicional.

A mis compañeros de aula, que estuvieron siempre ahí para mí. Que sepan que nunca los olvidaré a pesar de la distancia.

A los muchachos del centro por su apoyo cuando más lo hemos necesitado.

A todo aquel que se ha preocupado por mí y ha sido capaz de decir **"CuEnta CoNmIgO"**.

Al tribunal de tesis por guiarnos hacia la meta y por la confianza depositada en cada uno de nosotros.

En fin agradecer a todos los que de una forma u otra han propiciado los grandes resultados alcanzados en el transcurso de todo este tiempo, de corazón **"MUCHAS GRACIAS"**.

Liadelis Jones Perez

A mi mamá por ser la mejor madre del mundo, apoyarme en cada paso de mi vida y su amor incondicional.

A mi papá por ser mi más grande amigo, por todo su sacrificio y su amor desmedido.

A mis abuelas por todo su cariño. En especial a mi abuela Eva por ser esa persona tan maravillosa, por toda su entrega y dedicación y siempre haber estado ahí para mí.

Al amor de mi vida, Yanet Coba Díaz, por ser mi tesoro más grande y la dueña de mi felicidad. A tí mi amor gracias, sin ti esto hubiese sido imposible.

A mis tíos Rey y Robertico, mis tías Aleida y Myrialis y mis hermanos Robertico y Siumey. Mi familia gracias por todo.

A mi padrastro por todo su apoyo y cariño.

A Grecia, Ofelia y familia por estar siempre presentes.

A mis grandes amigos Dennis, Dayrol y Yoandri por apoyarme en todo momento.

A mis amigos del apartamento Leonard, Dairon, Jose y Duanny.

A mi compañera de tesis por todo su esfuerzo y dedicación.

A mis tutores por todo el apoyo que nos brindaron, por la confianza que depositaron en nosotros, por dedicarnos su tiempo y paciencia.

A los muchachos del centro por su apoyo en todo momento.

A cada persona que dedico un segundo de su tiempo para preguntar: ¿Cómo te va con la tesis?

A todas las personas que hicieron posible que este sueño se hiciera realidad, sinceramente muchas gracias.

Raúl Bach Eng

Dedicatoria

Especialmente a mis padres por tanto sacrificio durante estos 5 años de esfuerzo, voluntad y firmeza.

A mi abuelita querida por haberme inspirado a lograr lo que con tanto sacrificio he alcanzado.

Liadelis

Resumen

La presente investigación persigue como objetivo el desarrollo de un Componente para el Sistema de Gestión del Ciudadano, que mejore la gestión del proceso de la guardia obrera estudiantil en la UCI, cuya misión principal es preservar la integridad física de los medios, recursos, instalaciones y personas de la universidad. Actualmente, la planificación y control de la guardia no se ejecuta de forma eficiente y centralizada, lo que en ocasiones provoca que existan dificultades durante la realización de la misma. Este comportamiento ha demostrado que el procedimiento empleado para garantizar la seguridad y protección de la institución se ha convertido en un proceso inestable e inapropiado. Dicha situación ha sido objeto de preocupación y alerta por parte de los directivos de la universidad, por lo que en consecuencia para el comienzo del curso 2014-2015 se confecciona un proyecto de rediseño de la guardia obrera estudiantil, que permitirá perfeccionar la planificación, organización, control y cumplimiento de la misma. A raíz de estos sucesos se propone el desarrollo de un Componente para la Gestión de la Guardia Obrera Estudiantil el cual permitirá gestionar un conjunto de elementos que son cruciales en el proceso de gestión de la guardia como son: postas, turnos, zonas, recursos, potencial, planificación, entre otros. Además, facilita llevar a cabo un mejor control de la gestión de asistencias e incidencias ocurridas durante realización de la guardia.

Palabras claves: gestión, planificación, control, seguridad, guardia obrera estudiantil.

Índice

Introducción 1
Capítulo 1: La necesidad del componente y las tecnologías 5
1.1 Introducción 5
1.2 Conceptos asociados al dominio de la investigación 5
1.3 Análisis de soluciones de control de turnos y horarios 6
1.4 Análisis de soluciones precedentes en la UCI 8
1.5 Metodología a utilizar para la implementación de la solución 9
1.6 Lenguajes a utilizar para la implementación de la solución 11
Lenguaje de programación PHP 5.5.9-1 11
Lenguaje de Marcado de Hipertexto HTML 4 12
Lenguaje de marcas extensible XML 1.0 12
Lenguaje de consulta estructurado SQL 2012 13
Hojas de Estilo en Cascada CSS 3 13
1.7 Herramientas y tecnologías para el desarrollo de la aplicación 13
Herramienta Visual Paradigm para UML v 8.0 13
Apache JMeter 2.8.4 14
Administrador de base de datos PgAdmin III 1.14.0 14
Sistema gestor de base de datos PostgreSQL 9.4 15
Servidor Apache 2.4.7 15
Evolus Pencil 2.0.5 16
Subversión 0.12.1 16
Entorno de desarrollo integrado NetBeans 8.0 16
1.8 Marco de trabajo a utilizar para el desarrollo de la solución 17
Symfony 2.8 17
JQuery 17
1.9 Conclusiones parciales 18
Capítulo 2: Definición del componente 19
2.1 Introducción 19
2.2 Descripción del proceso de gestión de la guardia obrera 19

2.3 Descripción de las reglas del negocio....21
2.4 Requisitos del sistema....22
Definición de las técnicas de obtención de requisitos....22
Requisitos funcionales....22
Descripción de requisitos....25
Requisitos no funcionales....30
2.5 Descripción de la propuesta de solución....32
2.6 Descripción de la arquitectura....33
2.7 Patrones arquitectónicos....34
2.8 Patrones de diseño....37
Patrones de diseño GRASP....38
Patrones de diseño GoF....42
2.9 Patrones de diseño de Base Datos. 42
2.10 Modelo de Base de Datos. 43
2.11 Modelo de despliegue 44
2.12 Conclusiones parciales....45
Capítulo 3: Construcción y validación del componente 46
3.1 Introducción....46
3.2 Estilos de programación 46
3.3 Estándares de codificación....47
Indentación, llaves de apertura y cierre, y tamaño de las líneas 47
Convención de nomenclatura 48
Documentación....49
Comentarios 49
3.4 Validación de requisitos....49
Criterios para validar los requisitos 50
Técnicas de validación de requisitos....51
3.5 Estrategias de pruebas....51
Pruebas de integración 52
Pruebas de sistema 53
Pruebas de resistencia 54

Pruebas de rendimiento 54
3.6 Conclusiones parciales 60
Conclusiones generales 61
Referencias bibliográficas 62
Bibliografía consultada 67

Índice de Figuras

Figura 1: Fases de la metodología DAC 11
Figura 2: Proceso de planificación de la guardia 20
Figura 3: Proceso de realización de la guardia 21
Figura 4: Mapa de navegación de la agrupación funcional Planificación 33
Figura 5: Arquitectura Cliente-Servidor 34
Figura 6: Patrón arquitectónico Modelo-Vista-Controlador 35
Figura 7: Patrón arquitectónico MVC con Symfony 2 37
Figura 8: Patrón Experto 39
Figura 9: Patrón Creador 40
Figura 10: Patrón Controlador 41
Figura 11: Diagrama de despliegue de la propuesta de solución 44
Figura 12: Ejemplos de estándares de codificación 47
Figura 13: Ejemplos de nomenclatura para clases 48
Figura 14: Ejemplos de nomenclatura para funciones 48
Figura 15: Documentación de las funciones 49
Figura 16: Prueba de desempeño a la funcionalidad Estado de planificación 55
Figura 17: Relación de no conformidades por iteración 60

Índice de tablas

Tabla 1: Reglas del negocio 21
Tabla 2: Requisitos funcionales del componente GOE 23
Tabla 3: Descripción del requisito Crear Posta 25
Tabla 4: Requisitos no funcionales del componente GOE 30
Tabla 5: Pruebas de software 52
Tabla 6: Prueba de integración del componente Seguridad 53
Tabla 7: Resultados de las pruebas de sistema 55
Tabla 8: Diseño de caso de prueba Crear planificación 56
Tabla 9: Relación de no conformidades por iteraciones 60

Introducción

Desde sus inicios el hombre sintió la necesidad de proteger sus bienes y sus familias, una actividad que se le hacía cada vez más complicada y que lo obligó a ir en búsqueda de nuevas estrategias que le permitieran salvaguardar sus intereses. Con el transcurso del tiempo fueron creciendo las ciudades y las personas poseían cada vez más bienes para poder satisfacer sus necesidades. Esto conllevó, a un perfeccionamiento de sus pensamientos, técnicas, instrumentos y medidas de seguridad. Es entonces en el año 27 antes de Cristo (a.c) que se crea en el imperio romano la Primera Guardia, con el propósito de mantener el orden y proteger la integridad física de sus dirigentes, marcando así, el nacimiento de los primeros sistemas de seguridad (1).

Actualmente, la seguridad se ha convertido en un aspecto indispensable para dar protección a los recursos de las instituciones, pues de ella depende la integridad de toda su estructura y su misión principal es velar por el cuidado y custodia de todos sus componentes internos, preservando por encima de todo, la vida humana.

Cuba es un país que a pesar de no poseer una alta infraestructura tecnológica y modernos sistemas de vigilancia ha logrado crecerse ante las dificultades y ha puesto en práctica un grupo de medidas y acciones en aras de garantizar la seguridad de sus organizaciones. La Universidad de las Ciencias Informáticas (UCI) es una de estas instituciones que presenta un gran número de medios que necesitan ser protegidos; para este propósito entre una de sus áreas se encuentra la Dirección de Seguridad y Protección, facultada de orientar y dirigir el proceso de gestión de la guardia obrera estudiantil, el cual constituye uno de los procesos fundamentales de la universidad para el cuidado y custodia de sus recursos y bienes materiales.

Debido a la cantidad de incidencias e indisciplinas ocurridas en la realización de la guardia obrera estudiantil, en el curso 2013-2014 no pudieron ser controlados un conjunto de hechos delictivos que causaron la pérdida de bienes materiales a la universidad. Por consiguiente, se hizo necesaria la confección del Proyecto de Rediseño de la Guardia Obrera Estudiantil, donde se establecen pautas, postas, horarios y otros elementos a tener en cuenta en las distintas áreas de la Universidad para realizar una mejor gestión de la guardia obrera estudiantil.

Actualmente la mayoría de las áreas elaboran la planificación manualmente, provocando que se cometan errores por parte del planificador y que ante situaciones imprevistas se le dificulte la realización de

cambios a la misma. Además, en el momento en que es concebida la planificación de la guardia, le resulta muy difícil al planificador tener en cuenta el historial de guardia de cada persona, causando que se repitan los turnos y/o los días de la semana (fundamentalmente los fines de semana), lo que conlleva a que existan inconformidades por parte del personal que debe realizar la guardia. De igual manera, en varias ocasiones se desconoce la situación de las personas que conforman los equipos de guardia (personal de certificado médico o cumpliendo misión, etc.) o la planificación es enviada al personal de la universidad con un período de antelación muy corto. Todos estos factores traen como consecuencia, que estas personas se ausenten a la guardia en el turno asignado y los equipos queden incompletos. También, la reiteración con que se realiza la guardia obrera no es balanceada en todas las áreas de la universidad, lo que trae consigo que algunas personas les corresponda hacer el guardia con mayor frecuencia. Durante el proceso de control de la guardia, se deben generar reportes que son de interés para la Dirección de Seguridad y Protección y otros directivos de la universidad. Dichos reportes se confeccionan haciendo uso de herramientas ofimáticas, lo que puede provocar que no sean generados con la calidad requerida y en el tiempo establecido. Esto puede conllevar a que en determinados momentos exista pérdida y duplicidad de la información lo que dificulta la retroalimentación entre los directivos y el personal interesado sobre las medidas administrativas que se aplican a las personas que incumplen con la guardia. Estos problemas finalmente estimulan a que se reiteren las indisciplinas, por la existencia de un control deficiente del proceso de la guardia.

Debido a la situación detallada anteriormente se plantea como **problema de investigación:** ¿Cómo mejorar la gestión del proceso de la guardia obrera estudiantil en la UCI? Se definen como **objeto de estudio** de la investigación el proceso de gestión de turnos de trabajo, mientras que el **campo de acción** lo constituye el proceso de la guardia obrera estudiantil en la UCI.

Para dar solución al problema planteado se determina como **objetivo general** de la investigación: Desarrollar un componente informático para el Sistema de Gestión del Ciudadano (SGC), que mejore la gestión de la guardia obrera estudiantil en la UCI.

A partir del objetivo general se derivan los siguientes **objetivos específicos:**

1. Fundamentar los elementos teóricos necesarios para el desarrollo de la investigación.
2. Diseñar la propuesta de solución a partir del proceso de desarrollo de software utilizado.
3. Implementar el Componente para la Gestión de la Guardia Obrera Estudiantil en el Sistema de Gestión del Ciudadano a partir de los requisitos definidos.

4. Validar la solución desarrollada ejecutando pruebas de software.

Para guiar el desarrollo de la investigación se plantea como **idea a defender:** El desarrollo del Componente para la Gestión de la Guardia Obrera Estudiantil en el Sistema de Gestión del Ciudadano mejorará la gestión del proceso de la guardia obrera estudiantil en la UCI.

Con la finalidad de cumplir con los objetivos específicos planteados se proponen las siguientes **tareas de investigación:**

- ✓ Análisis de sistemas similares al que se pretende desarrollar, para lograr un mejor entendimiento del objeto de estudio.
- ✓ Descripción de las tecnologías a utilizar para adquirir el conocimiento necesario para el desarrollo del sistema.
- ✓ Descripción de los requisitos del sistema, garantizando así un hilo conductor en el desarrollo de la solución.
- ✓ Confección del modelo de datos con el objetivo de describir la estructura lógica de la información persistente manejada por los componentes definidos.
- ✓ Definición de los estándares de codificación de los módulos con la finalidad de mantener la uniformidad en el código.
- ✓ Implementación de la solución para dar cumplimiento al objetivo propuesto.
- ✓ Ejecución de pruebas a los componentes desarrollados para validar la solución implementada.

Para el desarrollo de la investigación se utilizaron los siguientes métodos científicos:

Métodos teóricos

- ❖ El **Analítico-Sintético** se utilizó con el fin de hacer analogías y sintetizar documentos, artículos y otras fuentes en la búsqueda de sistemas de planificación y control de turnos de trabajo.
- ❖ La **Modelación** se utilizó para representar a través de diagramas, la lógica de negocio y las clases que conforman la solución informática permitiendo una mejor comprensión, explicación y descripción de los procesos que intervienen en la solución.
- ❖ El **Análisis-Documental** se utilizó para la interpretación y el análisis de los documentos rectores asociados al campo de acción lo que proporcionó un mejor dominio y comprensión del tema de investigación.

Métodos empíricos

- **Entrevista**, para dialogar con planificadores de diferentes áreas de la universidad con el objetivo de obtener mayor información sobre la situación actual del proceso de planificación y control de la guardia obrera estudiantil y conocer los principales problemas que afectan el cumplimiento satisfactorio de la misma. Para consultar la entrevista ver anexo 1.

El documento se encuentra estructurado de la siguiente manera:

Capítulo 1: "La necesidad del componente y las tecnologías". Se realiza el análisis del estado del arte del tema estudiado, se plantean los principales elementos teóricos asociados a la investigación. Se describen las tecnologías definidas por el Grupo de Arquitectura que se utilizan en la implementación de la solución.

Capítulo 2: "Definición del componente". Se propone la solución al problema de la investigación. Se define el diseño del sistema, se describen los patrones de diseño a emplear, se realiza la descripción de requisitos donde se establecen los prototipos de interfaz de usuario y se confecciona el diagrama de despliegue de la solución.

Capítulo 3: "Construcción y validación del componente". Se describen los estándares de codificación a utilizar, se definen las pruebas para verificar la calidad del Componente para la Gestión de la Guardia Obrera Estudiantil y se presentan los resultados.

Capítulo 1: La necesidad del componente y las tecnologías

1.1 Introducción

En aras de materializar una propuesta de solución se hace necesario abordar aspectos que fundamentan teóricamente y describen el dominio de esta investigación. El presente capítulo contiene un análisis del estado del arte de los sistemas de gestión de turnos, haciendo referencia a diferentes elementos teóricos de la investigación. Además, se definen qué metodología, herramientas y lenguajes se utilizan para el desarrollo del componente.

1.2 Conceptos asociados al dominio de la investigación

En Cuba el concepto de *Guardia Obrera* es un término muy manejado; en la Gaceta Oficial de la República de Cuba, Artículo 21, Ordinario del 30 de abril del 2007, Acuerdo No.5993, se establece la realización de la guardia obrera, independientemente de que existan o no otras formas de seguridad y protección físicas. La administración debe establecer los puntos u objetivos a cuidar, los horarios, el lugar donde se realiza la guardia y crear las condiciones necesarias para que se ejecute (2) .

La *guardia obrera estudiantil* como parte del Diseño del Sistema de Seguridad y Protección de la UCI, constituye un subsistema importante en la seguridad física de las instalaciones, medios, recursos y personas. Su objetivo es mantener la vigilancia activa de protección en las posiciones fijas y de rondas establecidas en todas las áreas de la universidad, informando acerca de todos los incidentes que atenten contra la seguridad de la institución, en aras de minimizar los riesgos y las amenazas de actividades delictivas en todas las áreas de la comunidad universitaria. Con el propósito de lograr una mejor planificación y control de la guardia obrera estudiantil se establece un conjunto de principios en el documento: "Proyecto de Rediseño de la guardia obrera estudiantil" (ver anexo 8). Actualmente el proceso de gestión de la guardia obrera estudiantil se encuentra distribuido por diferentes áreas, cada una de ellas cuenta con un planificador, responsable de conformar la planificación de la guardia y remitirla vía correo electrónico a toda la comunidad universitaria.

En el mundo a pesar de no existir el término de Guardia Obrera se maneja un concepto muy similar llamado *control de turnos*, donde se definen aspectos esenciales como los puestos de trabajo, los cuales conforman la unidad organizativa básica en el control de la planificación y la producción. Los turnos tienen como objetivo conformar el horario del puesto de trabajo diario y es donde se indican los distintos ciclos de fechas y el programa de turnos, los días en que el puesto de trabajo está activo y qué turno de trabajo tiene en cada día (3).

1.3 Análisis de soluciones de control de turnos y horarios

A partir de los conceptos analizados anteriormente, se realizó un estudio de algunos sistemas que gestionan horarios o turnos de trabajo, con el objetivo de estudiar los procesos reflejados en los mismos para el control de turnos. Además, se evaluaron las funcionalidades que brindan estos sistemas desde el punto de vista de diseño, usabilidad, flexibilidad en la planificación de los procesos principales.

- **Visual Time Live**

Visual Time Live es una solución diseñada por la compañía Robotics S.A que tiene como objetivo automatizar el control horario y flexibilizar los calendarios para poder optimizarlos en función de las necesidades reales de trabajo de la empresa. Este sistema gestiona todas las incidencias que se están produciendo sobre el plan horario; ausencias, retrasos, excesos y enlaza los datos con su programa de nómina. Además, estructura de forma gráfica la organización de la empresa, gestiona múltiples contratos por empleado, almacena la información que considera necesaria y define todo tipo de horarios y calendarios. Por otra parte, permite crear calendarios anuales, mensuales, o del número de días que se desee; permite definir, por empleado, tanta información como se desee (4); así como el tiempo de presencia de sus empleados, el cual es gestionado en la aplicación de una forma 100% automatizada, rápida y eficiente.

Para Visual Time, el calendario es la pantalla principal. Mediante el calendario se puede visualizar, en un potente, visual e intuitivo formato, toda la información necesaria para realizar el seguimiento horario del personal de la empresa. Visual Time dispone de calendarios individuales para cada empleado, permitiendo realizar ajustes individualmente en función de las necesidades de la empresa, por ejemplo, cambios de vacaciones por puntas de trabajo o por petición personal del empleado, sin afectar a los calendarios del resto del personal (4).

Al conectarse el usuario a Visual Time Live, la aplicación le informa de todas las tareas que requieren de su intervención y las puede ir solucionando cuando desee o tenga tiempo sin interferir con la solución. Su interfaz intuitiva le facilita su uso y aprendizaje (4). El diseño de la aplicación le brinda al usuario un entorno de trabajo cómodo, intuitivo, y sencillo. La utilización de iconos descriptivos de acceso directo permite la localización de funciones concretas. Además, dispone de opciones para facilitar el acceso rápido a las tareas diarias o más habituales.

- **Jano Planificación Hospitalaria**

Jano Planificación Hospitalaria es un sistema desarrollado por la compañía EMR Software (Electronic Medical Records Software), diseñado específicamente para centros sanitarios. Su entorno de trabajo permite definir de forma personalizada cada centro y su zonificación (edificios, áreas, espacios) en el cual se asignan los puestos de trabajo. Del mismo modo la aplicación incorpora controles y asistentes que facilitan la labor de organización, optimización y reparto de cargas de trabajo optimizadas, además del gestor de cuadrantes de planificación (5). También permite múltiples frecuencias y periodicidades (diarias, semanales, quincenales, mensuales, bimestrales, trimestrales y semestrales). Por otro lado, planifica tareas puntuales, calcula la diferencia de horarios previstos y horarios reales, gestiona suplencias y substituciones del personal (5).

- **aTurnos**

aTurnos es una herramienta para la gestión de recursos humanos con funcionalidades específicas asignadas según diferentes tipos de roles. Esta herramienta permite ajustar los recursos existentes a la demanda atendiendo a las prioridades definidas por la dirección de recursos humanos de las empresas, una vez dimensionados los equipos posibilita ajustar las desviaciones teniendo en cuenta las bajas, los picos de trabajo y las nuevas necesidades operativas de las instituciones. Además, facilita mover el personal de los equipos de una localización a otra, retrasar o adelantar las horas de entrada y salida o incluso ajustar las curvas de trabajo. Presenta una gestión tan dinámica que para controlar el personal se hace necesario que los trabajadores validen que físicamente están en su puesto de trabajo (6).

La herramienta presenta también un menú cuadrante que muestra los turnos del equipo con los cambios actualizados del mes actual, los turnos asignados a cada compañero por día del mes y los tipos de turnos actualizados. Por otra parte, esta presenta un cuadrante muy destacado que posibilita realizar cambios de turnos entre compañeros, los cuales son registrados en un menú histórico en el cual aparece por defecto, todos los cambios del mes entre los compañeros de un equipo de trabajo. También cuenta con la funcionalidad de gestionar los días festivos de forma tal que el administrador pueda definir el día, mes y el año en que se va a celebrar la festividad (6).

Los sistemas internacionales analizados no son de código abierto, lo que dificulta la obtención y modificación del código fuente para ser utilizado por cualquier institución. Sin embargo, permitieron obtener un conjunto de elementos a tener en cuenta para que el desarrollo del componente proporcione la mayor cantidad de funcionalidades posible y además brinde mejor usabilidad para los usuarios. También

aportaron varios elementos como: definir horarios, definir calendarios, gestionar de asistencia, realizar notificaciones y asignar postas.

1.4 Análisis de soluciones precedentes en la UCI

En la UCI existen soluciones para la gestión de la guardia obrera estudiantil, a pesar de no usarse actualmente, se consideran importantes debido al estudio realizado sobre el funcionamiento de la guardia obrera estudiantil en diferentes áreas y etapas.

- **Sistema para la Gestión de la Guardia Obrera Estudiantil en la Facultad 3**

El Sistema para la Gestión de la Guardia Obrera Estudiantil en la Facultad 3 fue desarrollado en el año 2013 con el fin de obtener una herramienta que permita la gestión de la planificación de la Guardia Obrera Estudiantil (GOE) en la Facultad 3 de la UCI. El mismo fue desarrollado utilizando el marco de trabajo Sauxe 2.2 con tecnologías y herramientas definidas por el centro CEIGE. Permite la gestión de procesos que tributan al adecuado desempeño, organización y funcionamiento de la guardia en esta facultad. Presenta entre sus principales funcionalidades: planificar guardia, notificar planificación y cambios de guardia, así como registrar incidencias. Sin embargo, la herramienta no permite gestionar horario, ni estado de las personas, dígase: certificado médico, embarazada o cumpliendo misión, lo que puede provocar ausencias en la realización de la guardia. Además tampoco permite la planificación de la guardia al personal externo para períodos vacacionales.

- **Sistema de planificación y control de la Guardia Obrera Estudiantil en la Facultad 4**

El sistema fue desarrollado en el año 2015 con el propósito de lograr una mayor efectividad en el proceso de planificación y control de la guardia obrera estudiantil de la Facultad 4. El mismo se implementó haciendo uso del marco de trabajo Symfony 2.5 y la metodología de desarrollo de software XP (Extreme Programming). Además cuenta con un conjunto de funcionalidades como: gestionar planificación, cambiar estado de disponibilidad de los usuarios, crear brigadas de estudiantes y realizar intercambios de turnos de guardia. También permite controlar asistencia y generar reportes. Sin embargo, esta aplicación es poco flexible pues no permite crear nuevos turnos de guardia, ni asociar características a personas.

- **El Sistema de Información Geográfica para la Seguridad y Protección en la Universidad de las Ciencias Informáticas (GeoQ-Guardián)**

La plataforma GeoQ-Guardian fue desarrollada en el año 2013 por el centro de Geo-Informática y Señales Digitales (GEYSED) perteneciente a la UCI. Esta plataforma se desarrolló bajo la metodología de desarrollo de software RUP (Rational Unified Process) y surge por la necesidad de optimizar el proceso de

planificación y control de la guardia en la Universidad. El sistema permite la gestión de postas, turnos de guardia, zonas y personal que realiza la guardia. También presenta las funcionalidades: planificar guardia, modificar la planificación de la guardia, controlar guardia, generar reportes e imprimir la información gestionada. Esta aplicación a pesar de ser bastante abarcadora en cuanto a la cantidad de funcionalidades que ofrece, no permite gestionar características ni asociarlas a las personas; al ser una aplicación de escritorio su acceso se limita al ordenador donde está instalada y requiere de instalación personalizada en cada una de las áreas.

El análisis de los sistemas desarrollados en la UCI emitió que estos sistemas no cumplen con todas las funcionalidades definidas para la planificación de la guardia. Además presentan una estructura poco flexible debido a que no permiten la creación de nuevos turnos o postas y por lo general constituyen sistemas muy personalizados. Por otra parte estos sistemas presentan una determinada arquitectura: marco de trabajo y herramientas utilizadas, por lo que sería muy engorroso estudiarlos para reutilizarlos y/o adaptarlos a la arquitectura empleada. Estos indicadores demuestran que estas soluciones no son las más idóneas, además sus diseños no se corresponden con las nuevas resoluciones establecidas para la planificación de la guardia obrera estudiantil en la universidad (ver anexo 8). No obstante su estudio permitió la extracción de un conjunto de aspectos que sirven de guía para la concepción del Componente para la Gestión de la Guardia Obrera Estudiantil, entre los que se pueden encontrar: gestionar postas, gestionar turnos, gestionar planificación y reportar incidencias.

1.5 Metodología a utilizar para la implementación de la solución

Para estructurar, planear y controlar el proceso de desarrollo del Componente para la Gestión de la Guardia Obrera Estudiantil, el Departamento de Desarrollo de la DIN[1] de la UCI utiliza la metodología DAC[2].

[1] **DIN**: Dirección de Informatización, se encarga de informatizar varios procesos que se desarrollan en la Universidad de las Ciencias Informáticas.

[2] **DAC**: Desarrollo Ágil con Calidad, metodología diseñada por el Departamento de Desarrollo de la Dirección de Informatización de la UCI.

- **Proceso de desarrollo de software DAC**

La metodología DAC se basa en un proceso de desarrollo de software que combina las metas y prácticas de las áreas de procesos del nivel dos de CMMI[3] con las buenas prácticas de la dirección y desarrollo ágil de proyectos de software. Es un proceso colaborativo, recursivo-iterativo, además es incremental y guiado por procesos y requisitos. Su modelo del proceso es una adaptación del modelo en Cascada a los modelos Programación Extrema y Desarrollo Concurrente. Está enfocado a proyectos pequeños o grandes divididos en sub-proyectos que desarrollan software de gestión basado en componentes.

DAC plantea que el problema una vez identificado y definido debe ser descompuesto en problemas más pequeños, y si es necesario, realizar con estos la misma operación. Cada sub-problema será resuelto mediante un componente y el problema solucionado será el software o producto final; por lo que las entregas en DAC son a nivel de iteración, en la que habrá obligatoriamente un incremento del producto a partir de la solución de un componente del mismo. Además, en cada iteración se define como mínimo un hito a cumplir por cada fase del proceso. Al finalizar cada iteración se realiza la integración del componente al producto obtenido hasta el momento realizando pruebas de integración. Al finalizar las iteraciones se pueden realizar liberaciones del componente y transición del mismo dentro de la fase Cierre de iteración. Las iteraciones no tienen que desarrollarse todas al mismo tiempo, sino que al contar con un equipo pequeño este se va a ir moviendo de una iteración a otra a medida que estas vayan terminando de acuerdo a un orden de prioridad establecido en el plan del proyecto (7).

Este proceso tiene ocho actividades del marco de trabajo del proceso común, llamadas: fases o procesos del ciclo de vida: inicio, análisis, diseño arquitectónico, requisitos, construcción, cierre de iteración (opcional), liberación, transición y cierre, ocurriendo las iteraciones concurrentes entre las fases de requisitos, construcción y cierre de iteración. Además, entre las fases de requisitos y construcción puede ocurrir un ciclo pues a medida que los requisitos son especificados estos pueden ir entrando a la fase de construcción. El proceso tiene también dos áreas de procesos de protección: gestión de proyectos y soporte, así como dos fases o procesos horizontales cuyas tareas están presentes en varias de las fases del proceso común en forma de subprocesos: arquitectura y planificación (7).

[3] **CMMI:** *Capability Maturity Model Integration* (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.

En la *figura 1 se* muestran las diferentes fases por las que circula la propuesta de solución.

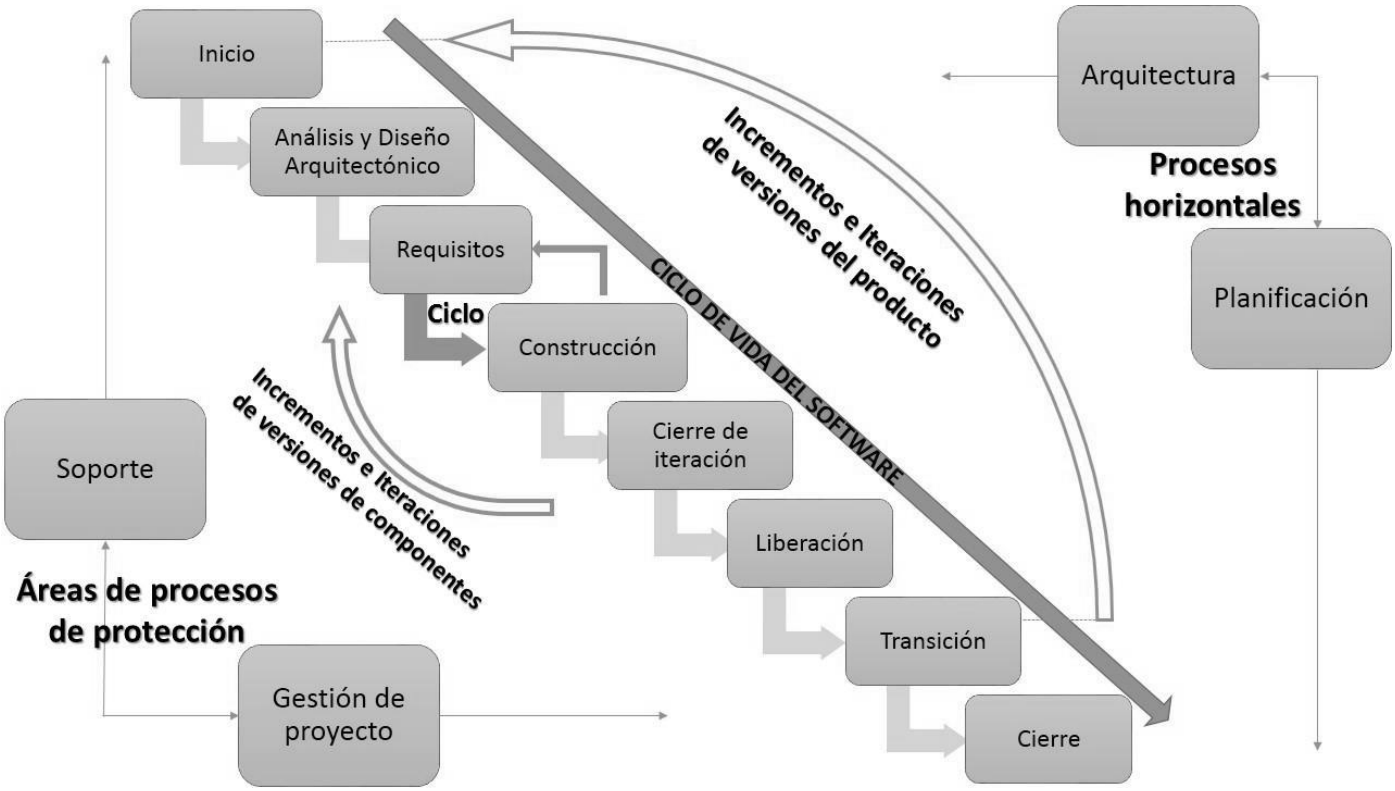


Figura 1: Fases de la metodología DAC (8).

1.6 Lenguajes a utilizar para la implementación de la solución

Con el objetivo de desarrollar el Componente para la Gestión de la Guardia Obrera Estudiantil, se utilizaron los siguientes lenguajes, los cuales fueron definidos en las políticas de desarrollo de la Dirección de Informatización (DIN).

- **Lenguaje de programación PHP 5.5.9-1**

PHP[4] es un lenguaje interpretado de alto nivel embebido en páginas HTML[5] y ejecutado en el servidor (9). Se conecta a servidores de bases de datos tales como MySQL, Oracle y PostgreSQL. Debido a su amplia distribución, está perfectamente soportado por una gran comunidad de desarrolladores. Es un lenguaje de

[4] **PHP:** *Hypertext Preprocessor* («Preprocesador de hipertexto »).

[5] **HTML:** *HyperText Markup Language* («lenguaje de marcas de hipertexto»).

programación de alto nivel, totalmente compatible con los modernos métodos orientados a objetos, prácticas y principios.

PHP compila para el código una serie de instrucciones siempre que estas son accedidas. La concepción de lenguaje interpretado es que las instrucciones son ejecutadas una por una, hasta que termina el código. Es usado principalmente en interpretación del lado del servidor para la generación de páginas web dinámicas. Las principales características de PHP son: rapidez, facilidad de aprendizaje, soporte multiplataforma tanto de diversos sistemas operativos como de servidores HTTP[6]. Este lenguaje se distribuye de forma gratuita bajo una Licencia Pública General (por sus siglas en inglés GPL) (10) .

- **Lenguaje de Marcado de Hipertexto HTML 4**

HTML es un lenguaje de marcas orientado a la publicación de documentos en Internet. La mayoría de las marcas son semánticas. HTML es un lenguaje extensible, al que se le pueden añadir nuevas características, marcas y funciones. Los documentos HTML están formados por una serie de bloques de texto con una entidad lógica (titulares, párrafos y listas). La interpretación de estas entidades se deja al navegador, lo cual da una gran flexibilidad a la presentación del documento, que puede ser mostrado, por ejemplo, en terminales gráficos o de texto. El HTML, es el lenguaje que permite diseñar los hipertextos. Hoy en día, la mayoría de los procesadores de textos disponen de opciones para guardar los documentos en este formato, por lo que no presenta dificultad (11).

- **Lenguaje de marcas extensible XML 1.0**

El lenguaje de marcas extensible es un metalenguaje que permite definir lenguajes de marcado adecuados a usos específicos. Deriva del lenguaje SGML[7] y permite definir la gramática de lenguajes específicos para estructurar documentos grandes. A diferencia de otros lenguajes, XML da soporte a bases de datos, siendo útil cuando varias aplicaciones se deben comunicar entre sí o integrar información. Después de diseñado y puesto en producción, es posible extender XML con la adición de nuevas etiquetas, de modo que se pueda continuar utilizando sin complicación alguna. Si un tercero decide usar un documento creado en XML, es sencillo entender su estructura y procesarla. XML mejora la compatibilidad entre aplicaciones. Se puede comunicar aplicaciones de distintas plataformas, sin que importe el origen de los datos (12).

[6] **HTTP:** *Hypertext Transfer Protocol* («protocolo de transferencia de hipertexto»).

[7] **SGML:** *Standard Generalized Markup Language* o "Estándar de Lenguaje de Marcado Generalizado".

- **Lenguaje de consulta estructurado SQL 2012**

Es el lenguaje estándar ANSI/ISO de definición, manipulación y control de bases de datos relacionales. Es un lenguaje declarativo: sólo hay que indicar qué se quiere hacer. En cambio, en los lenguajes procedimentales es necesario especificar cómo hay que hacer cualquier acción sobre la base de datos. El SQL es un lenguaje muy parecido al lenguaje natural; concretamente, se parece al inglés, y es muy expresivo. Por estas razones, y como lenguaje estándar, el SQL es un lenguaje con el que se puede acceder a todos los sistemas relacionales comerciales (13).

- **Hojas de Estilo en Cascada CSS 3**

CSS es un lenguaje usado para definir la presentación de un documento estructurado escrito en HTML o XML (y por extensión en XHTML[8]). World Wide Web (WWW[9]) es la encargada de formular la especificación de las hojas de estilo que servirán de estándar para los agentes de usuario o navegadores. El CSS sirve para definir la estética de un sitio web en un documento externo y eso mismo permite que modificando ese documento (la hoja CSS) se pueda cambiar la estética entera de un sitio web, el mismo puede variar totalmente de estética cambiando solo la CSS, sin tocar para nada los documentos HTML (14).

1.7 Herramientas y tecnologías para el desarrollo de la aplicación

Con el objetivo de cumplir con las tareas de la investigación se determinó hacer uso de las herramientas informáticas previamente definidas por el departamento de desarrollo de la DIN.

- **Herramienta Visual Paradigm para UML v 8.0**

Visual Paradigm es una herramienta UML profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Permite construir todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación (15).

---

[8] **XHTML:** *Extensible HyperText Markup Language*. XHTML es básicamente HTML expresado como XML válido.

[9] **WWW:** En informática, la *World Wide Web* (WWW) o Red informática mundial comúnmente conocida como la web, es un sistema de distribución de documentos de hipertexto o hipermedios interconectados y accesibles vía Internet. Con un navegador web, un usuario visualiza sitios web que pueden contener texto, imágenes, vídeos u otros contenidos multimedia y navegar a través de esas páginas usando hiperenlaces.

Tiene la capacidad de crear el esquema de clases a partir de una base de datos y crear la definición de base de datos a partir del esquema de clases. Permite invertir código fuente de programas, archivos ejecutables y binarios en modelos UML al instante, creando de manera simple toda la documentación. Está diseñada para usuarios interesados en sistemas de software de gran escala con el uso del acercamiento orientado a objeto. Apoya los estándares más recientes de las notaciones de Java y de UML. Incorpora el soporte para trabajo en equipo, que permite que varios desarrolladores trabajen a la vez en el mismo diagrama y vean en tiempo real los cambios hechos por sus compañeros.

Visual Paradigm cumple con las políticas actuales de migración a software libre, siendo multiplataforma, de forma tal que facilita la modelación del software independientemente del sistema operativo que se emplee (16).

- **Apache JMeter 2.8.4**

Apache JMeter, es una herramienta *Open Source* realizada en java que se utiliza para realizar pruebas de rendimiento y resistencia, generalmente a aplicaciones web. JMeter permite realizar simulaciones de gran carga en el servidor, red o aplicación para comprobar su fuerza y para analizar el rendimiento ante diferentes tipos de sobrecarga (17).

- **Administrador de base de datos PgAdmin III 1.14.0**

PgAdmin III es una aplicación gráfica para administrar el gestor de bases de datos *PostgreSQL*. Es capaz de gestionar versiones a partir de *PostgreSQL* 7.3 ejecutándose en cualquier plataforma. Está diseñado para responder a las necesidades de todos los usuarios, desde escribir consultas SQL (Lenguaje de Consulta Estructurado) simples hasta desarrollar bases de datos complejas. La interfaz gráfica soporta todas las características de PostgreSQL y facilita enormemente la administración. La aplicación también incluye un editor SQL con resaltado de sintaxis, un editor de código de la parte del servidor y un agente para lanzar scripts programados. La conexión al servidor puede hacerse mediante conexión TCP/IP[10] y puede encriptarse mediante SSL[11] para mayor seguridad (18).

[10] **TCP/IP:** es un conjunto de protocolos de red que permiten la trasmisión de datos entre computadoras.

[11] **SSL** (***Secure Sockets Layer*****) o Capa de conexión segura:** es un protocolo criptográfico que proporciona comunicaciones seguras por una red.

- **Sistema gestor de base de datos PostgreSQL 9.4**

Se trata de un Sistema Gestor de Bases de Datos que incorpora el modelo relacional para sus bases de datos y es compatible con el lenguaje de consulta SQL[12]. Resulta ser muy capaz y muy confiable, tiene buenas características de rendimiento. Es un sistema multiplataforma por lo que opera en varios sistemas operativos como Unix, Mac OS, *Windows*, Linux. Es de código abierto lo que hace posible que los usuarios puedan realizar las modificaciones pertinentes al código fuente según la necesidad de estos (19). Entre sus principales características resaltan el uso de procedimientos almacenados, el soporte de integridad referencial y manejo de distintos tipos de datos. Utiliza la tecnología Control de Concurrencia Multi-Versión (por sus siglas en inglés MVCC) el cual permite que se ejecuten sobre una tabla varias transacciones a la vez, pues para cada transacción se muestra una versión de la tabla y no la original (20). Este gestor es robusto y brinda una API[13] flexible para programar en C/C++, Java, PHP, Python, entre otros. Se distribuye bajo la Licencia de Distribución de *Software* de *Berkeley* (por sus siglas en inglés BSD[14]), además cuenta con una comunidad de desarrollo que constantemente realiza mejoras al *software*.

- **Servidor Apache 2.4.7**

El servidor HTTP Apache es un servidor web HTTP de código abierto, para plataformas Unix (BSD, GNU/Linux, etc.), Microsoft Windows, Macintosh y otras, que implementa el protocolo HTTP. Es usado principalmente para enviar páginas web estáticas y dinámicas en la World Wide Web. Muchas aplicaciones web están diseñadas asumiendo como ambiente de implantación a Apache, o que utilizarán características propias. La arquitectura del servidor Apache es muy modular. Permite personalizar la respuesta ante posibles errores y posibilita configurarlo para que ejecute un determinado script cuando ocurra un error en concreto. Permite la creación de ficheros de log facilitando, de este modo, el control de las acciones realizadas en el servidor (21).

[12] **SQL:** *Structured Query Language* («lenguaje de consulta estructurado »).

[13] **API:** Interfaz de Programación de Aplicaciones.

[14] **BSD:** *Berkeley Software Distribution* («distribución de *software berkeley*»). Es una licencia de software libre permisiva como la licencia de OpenSSL o la MIT License.

- **Evolus Pencil 2.0.5**

Pencil es una herramienta para elaborar prototipos, disponible para todas las plataformas, construida con el propósito de ofrecer una herramienta de creación de prototipos, libre y de código abierto de interfaz gráfica de usuario que sea fácilmente de instalar y utilizar para crear maquetas de plataformas de escritorio populares. Tiene entre sus características principales que los proyectos pueden ser exportados en los formatos HTML, PNG (*Portable Network Graphics)*, documento de Word y PDF (*Portable Document Format)*, además, permite la instalación de plantillas definidas por el usuario, las operaciones de dibujo estándar: alinear, escalar y rotar y la adición de los objetos externos (22).

- **Subversion 0.12.1**

Subversion es una herramienta de control de versiones open source[15] basada en un repositorio cuyo funcionamiento se asemeja enormemente al de un sistema de ficheros. Es software libre bajo una licencia de tipo Apache/BSD. Utiliza el concepto de revisión para guardar los cambios producidos en el repositorio. Entre dos revisiones sólo guarda el conjunto de modificaciones (delta), optimizando así al máximo el uso de espacio en disco. Subversión permite al usuario crear, copiar y borrar carpetas con la misma flexibilidad con la que lo haría si estuviese en su disco duro local. Dada su flexibilidad, es necesaria la aplicación de buenas prácticas para llevar a cabo una correcta gestión de las versiones del software generado. Subversión puede acceder al repositorio a través de redes, lo que le permite ser usado por personas que se encuentran en distintas computadoras. A cierto nivel, la posibilidad de que varias personas puedan modificar y administrar el mismo conjunto de datos desde sus respectivas ubicaciones fomenta la colaboración. Se puede progresar más rápidamente sin un único conducto por el cual deban pasar todas las modificaciones (23).

- **Entorno de desarrollo integrado NetBeans 8.0**

NetBeans es un entorno de desarrollo integrado disponible para Windows, Mac, Linux y Solaris. Consiste en un IDE de código abierto y una plataforma de aplicaciones que permiten a los desarrolladores crear aplicaciones de escritorio, web y móviles, soporta varios lenguajes, entre ellos: PHP, JavaScript, C++, etc. Puede ser integrado fácilmente con Symfony2. Está apoyado por una comunidad de desarrolladores y,

[15] **Open source:** Código abierto.

ofrece una amplia documentación y recursos de capacitación, así como una gran cantidad de plugins de terceros (24).

1.8 Marco de trabajo a utilizar para el desarrollo de la solución

- **Symfony 2.8**

Es un completo framework diseñado para optimizar el desarrollo de las aplicaciones web basado en el patrón Modelo Vista Controlador. Para empezar, separa la lógica de negocio, la lógica de servidor y la presentación de la aplicación web. Proporciona varias herramientas y clases encaminadas a reducir el tiempo de desarrollo de una aplicación web compleja. Además, automatiza las tareas más comunes, permitiendo al desarrollador dedicarse por completo a los aspectos específicos de cada aplicación. Además, está desarrollado completamente en PHP 5.3 y es compatible con la mayoría de gestores de bases de datos, como MySQL, PostgreSQL, Oracle y Microsoft SQL Server. Se puede ejecutar tanto en plataformas *nix (Unix, Linux, etc.) como en plataformas Windows (25).

- **JQuery**

JQuery es un framework JavaScript que permite de una manera ágil crear aplicaciones web, es libre, su código se distribuye bajo la Licencia del Instituto Tecnológico de Massachusetts (por sus siglas en inglés MIT) y la Licencia Pública General de GNU en su versión 3. Cuenta con funcionalidades para acceder al árbol del DOM [16] del documento HTML, posee selectores para elegir los elementos por clases e identificadores, o para seleccionar por tipo de elemento. Permite el manejo de los eventos que generan los periféricos de entrada como el mouse o ratón y el teclado. Es multiplataforma y extensible a través de plugins [17], permitiendo hacer más amplias sus funcionalidades, o agregar aquellas que no estén en existencia. Su tamaño es realmente pequeño, disminuyendo el ancho de banda o tráfico de red, desde el servidor al dispositivo cliente. Posee una amplia documentación en línea[18]. Las características de jQuery permitirán desarrollar las interfaces de usuario interactivas de manera ágil, simplificando la cantidad de líneas de código para realizar una operación y también aplicaciones que hacen uso de Asynchronous JavaScript And XML (AJAX) técnica de desarrollo web para crear aplicaciones interactivas (26).

[16] **DOM:** Modelo de objetos del documento.

[17] **Plugins:** Componente de *software* que permite ampliar las funcionalidades del mismo.

[18] http://docs.jquery.com.

1.9 Conclusiones parciales

El presente capítulo propició sentar las bases teóricas teniendo en cuenta los principales conceptos asociados al dominio de la investigación. Además, se realizó un análisis de los sistemas similares lo que permitió identificar los principales problemas existentes y a su vez encontrar elementos de interés que pudieran ser utilizados en la implementación del nuevo componente. Por último, se describieron la metodología, las herramientas, lenguajes y tecnologías a utilizar para la solución propuesta.

Capítulo 2: Definición del componente

2.1 Introducción

En este capítulo se presenta la propuesta de solución al problema planteado en el marco teórico de la investigación. Se define el diseño del sistema teniendo en cuenta el modelo de proceso, la descripción de las reglas del negocio y los requisitos funcionales y no funcionales del sistema. Además, se especifica la arquitectura, patrones de diseño y base de datos empleados. Finalmente se muestran el modelo de base de datos y el diagrama de despliegue construidos.

2.2 Descripción del proceso de gestión de la guardia obrera

El proceso de gestión de la guardia obrera estudiantil en la UCI está conformado por dos subprocesos. El primero, consiste en planificar la guardia obrera estudiantil e inicia cuando la Dirección de Seguridad y Protección envía el Plan de Seguridad y Protección al responsable de la región de guardia, quien se encarga de consultarlo y enviárselo al planificador. El responsable de región de guardia solicita a los jefes administrativos el potencial y lo envía al planificador. Cuando el responsable de la región no posee el potencial necesario envía una solicitud a la Dirección de Seguridad y Protección para que asigne más personas a esa región. Una vez que el planificador tiene el Plan de Seguridad y Protección y el potencial, confecciona la planificación y la envía por correo electrónico a todos los estudiantes, trabajadores y profesores implicados. Si los involucrados están conforme con la planificación actual proceden a realizar la guardia; si alguien no está de acuerdo con la misma envía una solicitud de cambio al planificador de la guardia, este efectúa los cambios y los notifica. Si están satisfechos con el nuevo cambio proceden a realizar la guardia. En caso contrario se repite el proceso. El segundo proceso comienza con la realización de la guardia y como parte de su cumplimiento, se genera un reporte de incidencias que el responsable del equipo de guardia entrega al planificador de guardia. El planificador remite las incidencias a la Dirección de Seguridad y Protección siendo el máximo responsable de tomar las medidas pertinentes.

En la *figura 2* se muestra el proceso de planificación de la guardia.

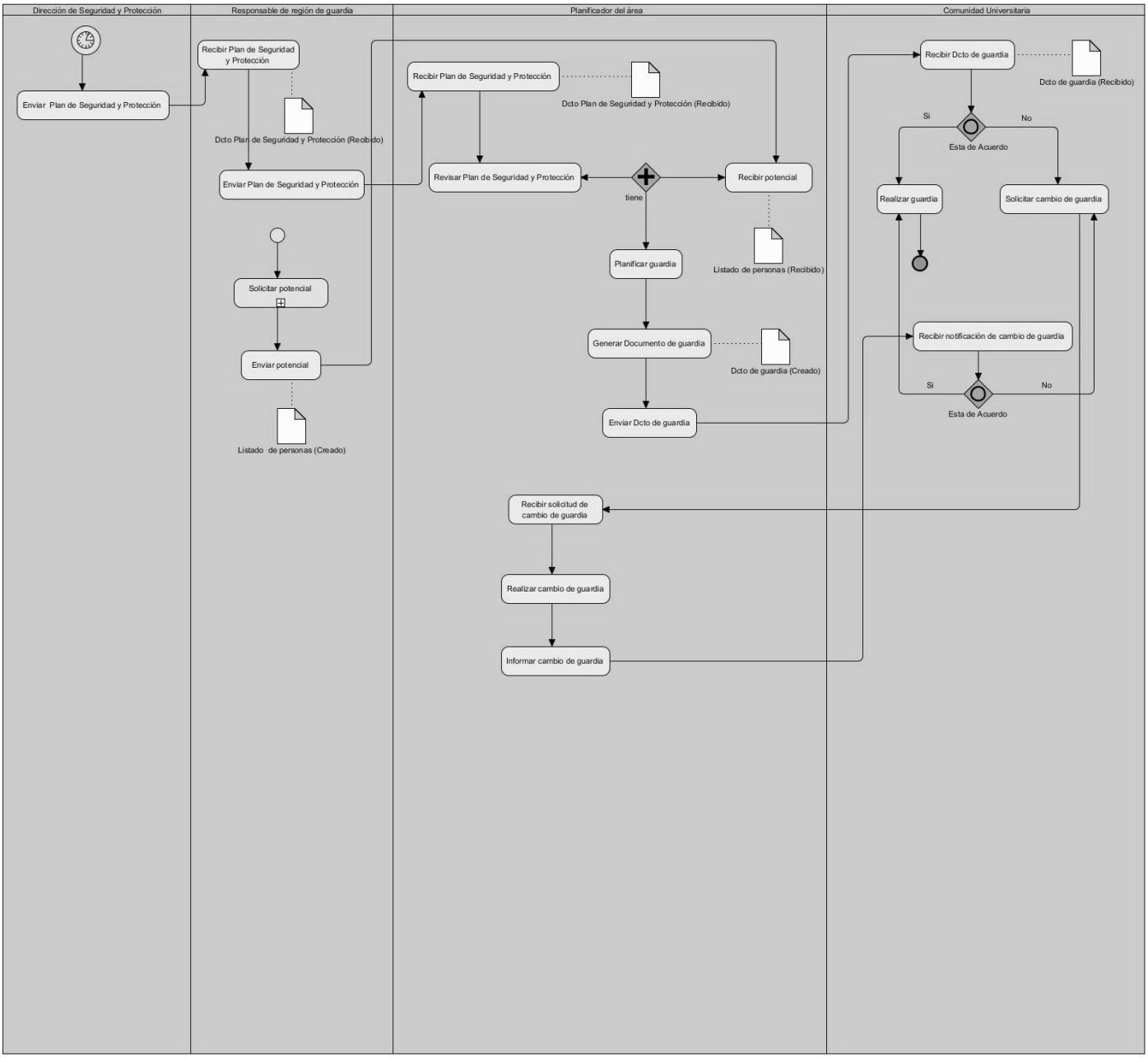


Figura 2: Proceso de planificación de la guardia.

En la *figura 3* se muestra el proceso de realización de la guardia.

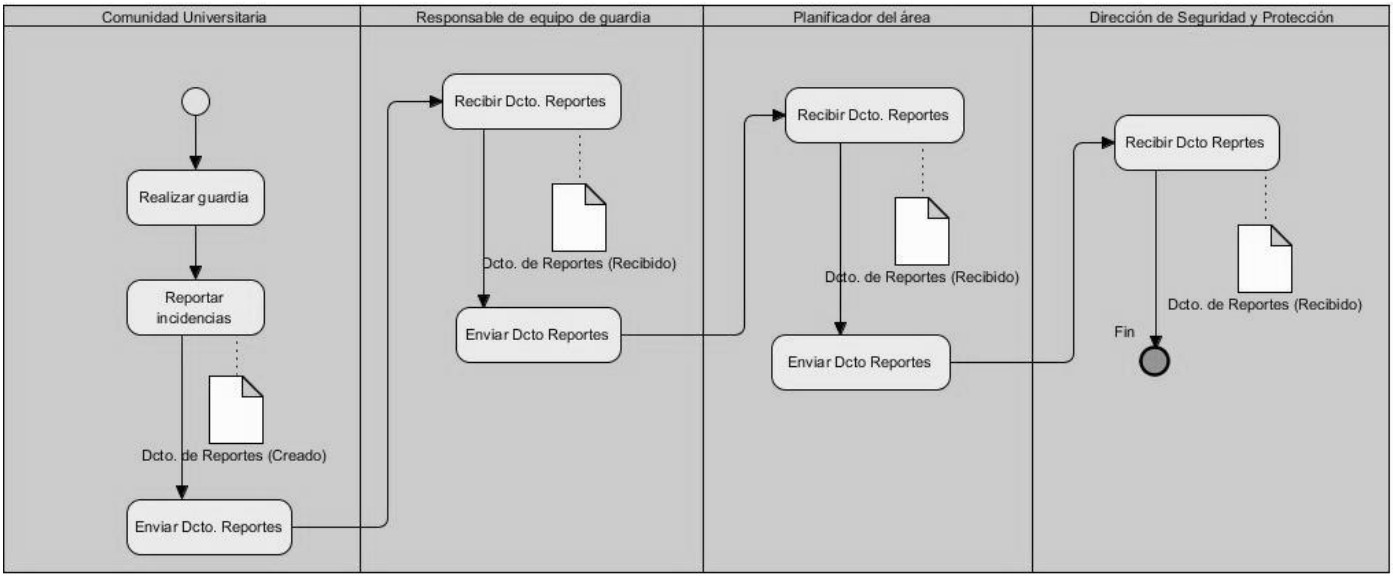


Figura 3: Proceso de realización de la guardia

2.3 Descripción de las reglas del negocio

Los procesos de negocio se rigen por reglas que garantizan su adecuada ejecución de acuerdo a las estrategias, objetivos y filosofía de la organización. Las reglas de negocio establecen los procedimientos que deben ser ejecutados y las condiciones que deben ser evaluadas y controladas en el flujo del proceso (27).En la *tabla 1* se describen las principales reglas del negocio asociadas a la solución propuesta.

Tabla 1: Reglas del negocio.

No.	Descripción
RN 1	Los días y turnos en que una persona puede hacer guardia se calcula a partir de las variables que tengan configuradas y la configuración de esas variables en el período planificado.
RN 2	Una región solo puede tener un horario sin calendario, por lo que quedaría ese horario por defecto para todos los días.
RN 3	Solo pueden registrar incidencias las personas que están de guardia en una fecha, posta y turno o el oficial de guardia superior de esa fecha.
RN 4	La configuración de una variable tiene que estar en el rango de fecha de la configuración.
RN 5	El potencial se muestra de acuerdo al área o zona del usuario autenticado.

RN 6	Una posta solo pertenece a una zona de guardia.
RN 7	Una persona solo pertenece a un grupo de guardia.
RN 8	A los planificadores solo se les mostrará las postas y turnos activos que pertenezcan a su zona.

2.4 Requisitos del sistema

La captura de requisitos es un proceso elemental que interviene en todo el ciclo de vida de un software. De ella depende desarrollar un sistema que incluya la mayor cantidad de funcionalidades posibles y esté a la altura de las expectativas reales del cliente. Para el desarrollo de la propuesta de solución se identificaron un conjunto de requisitos funcionales y no funcionales a partir de distintas técnicas.

- **Definición de las técnicas de obtención de requisitos**

Las técnicas de obtención de requisitos permitieron establecer una conexión directa entre el cliente del producto y el equipo de desarrollo, con el objetivo de definir cuáles son las prioridades y necesidades reales existentes. Estas técnicas se centran fundamentalmente en el descubrimiento de los requisitos del sistema, las aplicadas en la presente investigación fueron las siguientes:

- **Entrevista:** Se utilizó con el propósito de obtener diversos criterios siguiendo como método el intercambio de preguntas para esclarecer con exactitud el proceso de planificación de la guardia obrera estudiantil.
- **Tormenta de ideas (*brainstorming*):** Se realizaron reuniones donde estuvo presente el equipo de desarrollo con el objetivo de generar ideas con respecto a la solución del problema.
- **Prototipado:** Esta técnica se utilizó para la confección de las vistas del sistema, mediante la elaboración de prototipos de interfaz. Esto proporcionó que el cliente pudiera visualizar una primera versión de cómo pudieran quedar definidas las vistas y conllevó a la formalización de nuevas de ideas que ayudaron a mejorar la solución.

- **Requisitos funcionales**

Los requisitos funcionales de un sistema describen qué debe hacer el sistema. Estos requisitos dependen del tipo de software que se desarrolle, de los posibles usuarios y del enfoque general tomado por la organización al redactar el requisito. Los requisitos funcionales del sistema describen con detalle la función de este, sus entradas, salidas y excepciones (28). En la *tabla 2 se* muestran los requisitos funcionales definidos para el Componente para la Gestión de la Guardia Obrera Estudiantil.

Tabla 2: Requisitos funcionales del componente GOE.

No.	Requisitos Funcionales	Complejidad
RFGOE1[19]	Mostrar zona de guardia	Baja
RFGOE2	Crear zona de guardia	Media
RFGOE3	Modificar zona de guardia	Media
RFGOE4	Ver plan de la zona de guardia	Baja
RFGOE5	Ver detalles de zona de guardia	Baja
RFGOE6	Mostrar postas	Baja
RFGOE7	Crear posta	Media
RFGOE8	Modificar posta	Media
RFGOE9	Ver plan de guardia de una posta	Baja
RFGOE10	Ver detalles de posta	Baja
RFGOE11	Mostrar variables	Baja
RFGOE12	Crear variables	Baja
RFGOE13	Modificar variable	Baja
RFGOE14	Ver detalles de variable	Baja
RFGOE15	Mostrar configuración	Baja
RFGOE16	Crear configuración	Alta
RFGOE17	Modificar configuración	Alta
RFGOE18	Ver detalles de configuración	Baja
RFGOE19	Mostrar grupos	Baja
RFGOE20	Crear grupo	Media
RFGOE21	Modificar grupo	Media
RFGOE22	Asociar potencial a un grupo	Media
RFGOE23	Ver detalles de un grupo	Baja
RFGOE24	Mostrar potencial	Baja

[19] **RFGOE:** Requisito funcional Guardia Obrera Estudiantil.

RFGOE25	Configurar variable a persona	Media
RFGOE26	Configurar variable a grupos de personas	Media
RFGOE27	Ver detalles de la persona	Baja
RFGOE28	Mostrar horarios	Baja
RFGOE29	Crear horario	Baja
RFGOE32	Modificar horario	Baja
RFGOE33	Ver detalles de horario	Baja
RFGOE34	Definir horarios a días por zona.	Media
RFGOE35	Mostrar turnos	Baja
RFGOE36	Crear turno	Baja
RFGOE37	Modificar turno	Baja
RFGOE38	Asociar turnos a horarios en la zona	Media
RFGOE39	Ver detalles del turno	Baja
RFGOE40	Mostrar tipos de recurso	Baja
RFGOE41	Crear tipo de recurso	Baja
RFGOE42	Modificar tipo de recurso	Baja
RFGOE43	Ver detalles de tipo de recurso.	Baja
RFGOE44	Mostrar estados de planificación	Baja
RFGOE45	Crear estado de planificación	Baja
RFGOE46	Modificar estado de planificación	Baja
RFGOE47	Ver detalles de estado de planificación	Baja
RFGOE48	Mostrar planificación	Baja
RFGOE49	Crear planificación	Media
RFGOE50	Modificar planificación	Media
RFGOE51	Ver detalles de planificación	Baja
RFGOE52	Plan de guardia	Alta
RFGOE53	Mostrar tipos de incidencia	Baja
RFGOE54	Crear tipo de incidencia	Baja
RFGOE55	Modificar tipo de incidencia	Baja

RFGOE56	Ver detalles de tipo de incidencia	Baja
RFGOE57	Mostrar incidencias	Baja
RFGOE58	Crear incidencia	Media
RFGOE59	Modificar incidencia	Media
RFGOE60	Ver detalles de incidencia	Baja
RFGOE61	Mostrar asistencias	Baja
RFGOE62	Mostrar registros de asistencia	Media
RFGOE63	Pasar asistencia	Media
RFGOE64	Modificar asistencia	Media
RFGOE65	Eliminar asistencia	Baja
RFGOE66	Crear potencial	Media
RFGOE67	Reportes de potencial	Media
RFGOE68	Reportes de incidencia	Media
RFGOE69	Reportes de periodicidad	Media
RFGOE70	Reportes de planificación	Media

- **Descripción de requisitos**

Una descripción de requisitos consiste en determinar las necesidades, establecer las funciones que cubren los requerimientos del sistema, determinando las restricciones a las que va a ser sometida y la proposición de alternativas ante la posible inviabilidad de la solución idónea (29). En la *tabla 3* se muestra la descripción de requisito asociada a la funcionalidad Crear posta.

Tabla 3: Descripción del requisito Crear Posta.

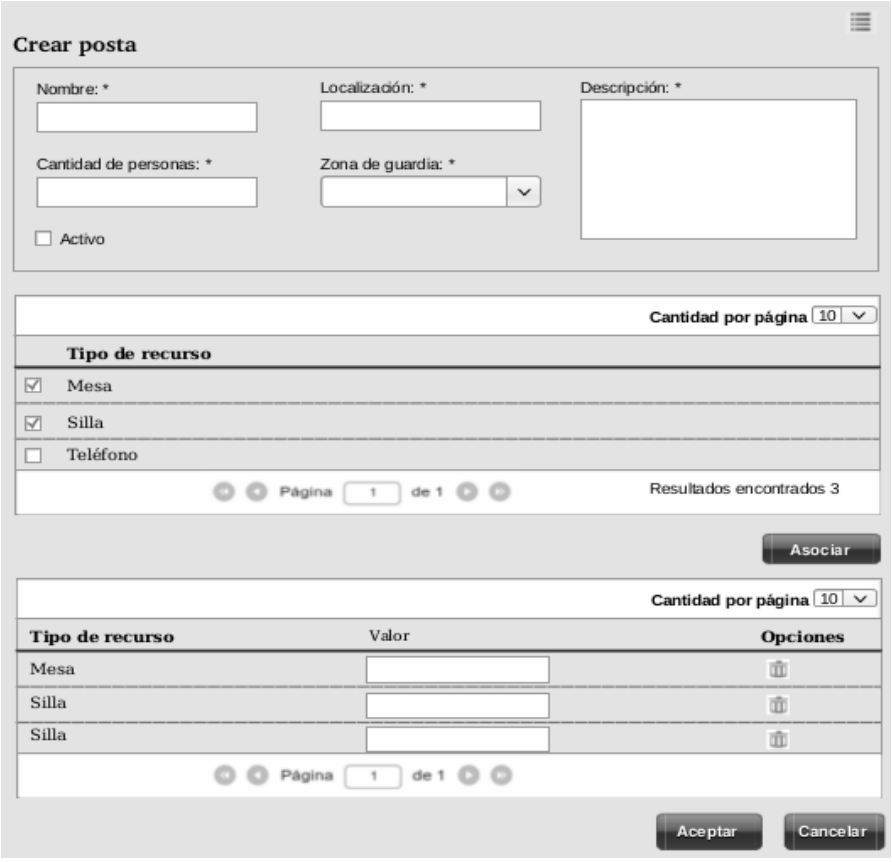
Precondiciones	El usuario debe estar autenticado en el sistema. Deben existir zonas en el sistema.
Flujo de eventos	
Flujo básico <<Crear Posta>>	
1	Seleccionar la opción Posta en el menú de la agrupación funcional Planificación.
2	Se muestra una interfaz con el listado de todas las postas de guardia que han sido creadas y las opciones Crear, Detalles y Modificar.
3	Se selecciona la opción Crear en el área de iconos flotantes.
4	Luego se muestra una interfaz con los campos: nombre, descripción, la opción activo (para indicar que la posta estará activa una vez creada), la localización, la cantidad de personas, la zona a la que pertenece la posta, el tipo de recurso y el valor del mismo. Además en el área de iconos flotantes se muestra la opción Listar.
5	Se llenan los campos. Cuando se asocian tipos de recursos aparece la opción Eliminar en el área de iconos internos.
6	Luego se selecciona la opción Aceptar.
Pos-condiciones	
1	Se crea la posta de guardia y se muestra el mensaje "El elemento ha sido creado satisfactoriamente".
Flujos alternativos	
Flujo alternativo 4.a Seleccionar la acción Listar en el área de iconos flotantes	
1	Se selecciona la acción Listar en el área de iconos flotantes.
2	Se muestra el listado con todas las postas existentes.
Pos-condiciones	
1	N/A
Flujo alternativo 5.a Dejar vacíos campos obligatorios	
1	Se dejan vacíos los campos marcados como obligatorios (campos que tienen el asterisco).
2	Se muestra un mensaje de error de color rojo encima del componente "Campo requerido" en el campo que debe ser llenado obligatorio.

Pos-condiciones	
1	N/A
Flujo alternativo 5.b Introducir datos incorrectos	
1	En caso de que introduzca datos incorrectos se muestra el mensaje de error "Entre sólo letras, números y espacio o guión bajo entre palabras".
2	Si se introducen más caracteres de los permitidos para una palabra el sistema muestra el mensaje "Ha excedido el número de caracteres permitidos para una palabra".
3	Si se trata de introducir más de los caracteres permitidos para los campos el sistema no permite seguir introduciéndolos o muestra el mensaje "No más de <cantidad> caracteres".
4	Si se trata de introducir menos de los caracteres permitidos para los campos el sistema muestra el mensaje "No menos de <cantidad> caracteres".
5	Si se introducen letras en el campo Cantidad de personas máximas se muestra el mensaje "Entre solo dígitos".
Pos-condiciones	
1	N/A
Flujo alternativo 5.c Seleccionar la acción Eliminar en el área de iconos internos	
1	Luego de asociados los tipos de recursos se presiona la acción Eliminar en el área de iconos internos.
2	Se elimina el tipo de recurso del listado de tipos de recursos asociados.
Pos-condiciones	
1	N/A
Flujo alternativo 6.a El elemento ya existe	
1	Se introducen los datos de un elemento existente y se presiona el botón Aceptar.
2	Se muestra el mensaje de error "El elemento ya existe".
Pos-condiciones	
1	N/A
Flujo alternativo 6.b Presionar el botón Cancelar	
1	Se muestra el mensaje de advertencia "¿Está seguro de realizar la acción? ".
2	Se selecciona la opción Aceptar o Cancelar.

Pos-condiciones		
1	N/A	
Validaciones		
1	**Nombre** • *Campo requerido.* • *Solo admite letras, números y espacio o guión bajo entre las palabras.* • *Cantidad de caracteres por palabra es 30.* • *Cantidad de caracteres máxima es 250 y mínima es 2.*	
2	**Descripción** • *Cantidad de caracteres por palabra es 30.* • *Cantidad de caracteres máxima es 1000.*	
3	**Localización** • *Campo requerido.* • *Solo admite letras, números y espacio o guión bajo entre las palabras.* • *Cantidad de caracteres por palabra es 30.* • *Cantidad de caracteres máxima es 250 y mínima es 2.*	
4	**Cantidad de personas** • *Campo requerido.* • *Solo admite números.* • *Cantidad de caracteres máxima es 5 y mínima es 1.*	
5	**Zona de guardia** • *Campo requerido.* • *Campo de selección.*	
6	**Tipo de recurso** • *Campo de selección.*	
Conceptos	Nombre	Indica el nombre con el que se crea la nueva posta de guardia.
	Descripción	Campo en el que se puede emitir una breve descripción de la nueva posta si es necesario.

	Activo	Indica que la posta está activa o no.
	Localización	Indica el lugar donde se encuentra la posta.
	Cantidad de personas	Indica la cantidad de personas que tendrá la posta.
	Zona de guardia	Indica la zona a la que pertenece la posta.
	Tipo de recurso	Indica el tipo de recurso que tiene asociada la posta.
Requisitos especiales		
Asuntos pendientes		

Prototipo elemental interfaz de usuario



❖ **Requisitos no funcionales**

Los requisitos no funcionales son aquellos que no se refieren directamente a las funciones específicas que proporciona el sistema, sino a las propiedades emergentes de este como la fiabilidad, el tiempo de respuesta y la capacidad de almacenamiento. Define las restricciones del sistema como la capacidad de los dispositivos de entrada/salida y las representaciones de datos que se utilizan en las interfaces del sistema (30). La *tabla 4* contiene los requisitos no funcionales definidos para el Componente para la Gestión de la Guardia Obrera Estudiantil.

Tabla 4: Requisitos no funcionales del componente GOE.

Usabilidad	
RNF 1[20]	El componente debe presentar un menú lateral y una barra de iconos flotantes que permitan el acceso rápido a la información por parte de los usuarios.
RNF 2	Las vistas del componente deben indicar en cada momento la acción que se está realizando, así como los íconos deben estar representados por una imagen acorde a la acción que se realiza.
RNF 3	Sólo se mostrarán a los usuarios aquellas acciones o informaciones del menú lateral a las que por su responsabilidad o rol dentro del negocio necesiten acceder.
Confiabilidad	
RNF 4	El tratamiento de las excepciones permitirá un seguimiento hasta guardar información, acerca del lugar donde se produjo el error y de los parámetros utilizados en el sistema que lo provocaron. Cuando ocurre una excepción el sistema mostrará un mensaje explicativo del error ocurrido.
RNF 5	El componente puede permanecer inactivo durante 10 minutos. Al cumplirse este término se cerrará la sesión teniendo que autenticarse el usuario nuevamente.
Eficiencia	
RNF 6	El componente deberá tener por cada transacción un tiempo de respuesta promedio de 20 segundos.
RNF 7	El componente soportará la conexión simultánea de todos los posibles usuarios, con un promedio de 1000 y un máximo de 3000.

[20] **RNF:** Requisito no funcional.

Soporte	
RNF 8	El componente cumplirá con las normas de codificación, conversiones para nomenclatura, bibliotecas de clase definidas para el Sistema de Gestión del Ciudadano en el documento SGC_Pautas de Configuración.
RNF9	El componente contará con toda la documentación definida en el expediente de proyecto asociada a su proceso de desarrollo para las actividades de soporte.
Restricciones de diseño	
RNF 10	El componente deberá ser desarrollado en su totalidad con tecnologías y componentes de código abierto.
RNF 11	El componente estará desarrollado con las herramientas definidas para el Sistema de Gestión del Ciudadano en el documento "Arquitectura de Software Vista de tecnología e infraestructura".
RNF 12	El componente cumplirá con la arquitectura de información definida para el Sistema de Gestión del Ciudadano en el documento de ''Arquitectura de Software Vista de Presentación DAC".
Interfaz	
RNF 13	La interfaz de usuario se guiará por la vista de arquitectura de presentación definidas en el documento "Pautas de diseño XABAL-SGC".
Software	
RNF 14	Para el despliegue del componente se debe contar en el servidor de aplicaciones web con: PHP versión 5.6 con las librerías php5-ldap, php5-gd, php5-mcrypt, php5-pgsql, php5-xsl, php5-*openssl* y Apache 2.2.
RNF 15	Para el despliegue del componente se debe contar en el servidor de bases de datos con PostgreSQL 9.4.
RNF 16	Para el uso del componente se requiere una PC cliente con cualquier sistema operativo, que se pueda instalar el navegador web Mozilla Firefox 26.0 o superior para el uso de la aplicación web.
Hardware	
RNF 17	Para la ejecución del componente se requiere que la PC cliente tenga los siguientes componentes de *hardware*: Pentium 4 o superior, 512 MB RAM y 200 MB disco duro disponible como mínimo.
RNF 18	La comunicación entre el cliente y el servidor de aplicaciones se realiza a través del protocolo HTTPS.

2.5 Descripción de la propuesta de solución

Se define como propuesta de solución el desarrollo del Componente para la Gestión de la Guardia Obrera Estudiantil perteneciente al Sistema de Gestión del Ciudadano. Dicho componente estará encargado de la planificación y control del proceso de la guardia obrera estudiantil en la UCI. Para ello se determinaron las siguientes agrupaciones funcionales: Configuración, Planificación, Control y Reportes.

La Configuración tiene como principal objetivo, la gestión de los elementos de configuración general del componente, permitiendo realizar modificaciones en caso de ser necesario. Esta agrupación funcional permite la creación de los diferentes estados que puede tener una planificación, los tipos de recursos con que cuentan las postas de guardia de la universidad y los tipos de incidencias, clasificadas de acuerdo al nivel de gravedad que pueda poseer las mismas. Por otra parte, posibilita la creación de los turnos de guardia necesarios para la ejecución de la planificación, así como los horarios a los cuales pertenecen dichos turnos de guardia. De igual manera permite crear todas las variables que pueden ser asociadas a una persona y a su vez definir un conjunto de restricciones en cuanto a días, turnos o períodos para cada variable creada.

En Planificación se encuentran todas las funcionalidades necesarias para ejecutar la planificación de la guardia obrera estudiantil, permitiendo la gestión de las zonas de guardia y las postas que componen cada zona. Del mismo modo posibilita la creación de grupos de guardia por áreas administrativas, la asignación de potencial a grupos de guardia y la asociación de variables a las personas comprometidas con el proceso de guardia. Finalmente permite crear ejecuciones de planificación donde son asignadas las personas a una posta y un turno de guardia en una fecha determinada, teniendo en cuenta siempre la configuración de las variables asociadas a estas personas.

La agrupación funcional Control tiene como principal objetivo garantizar el correcto funcionamiento de la guardia obrera estudiantil. Esta permite gestionar la asistencia de las personas que realizan la guardia, así como manejar las incidencias ocurridas durante la ejecución de la misma.

La agrupación funcional Reportes tiene como principal misión generar todos los documentos esenciales relacionados con el proceso de planificación y control de la guardia obrera estudiantil. Esto posibilitará que no ocurran errores a la hora de extraer datos y que exista una estandarización en los documentos. Además a través de los reportes se logrará homologar la frecuencia de guardia en las diferentes zonas de guardia.

En la *figura 4* se muestra el mapa de navegación de la agrupación funcional Planificación. Para consultar los restantes ver anexo 6.

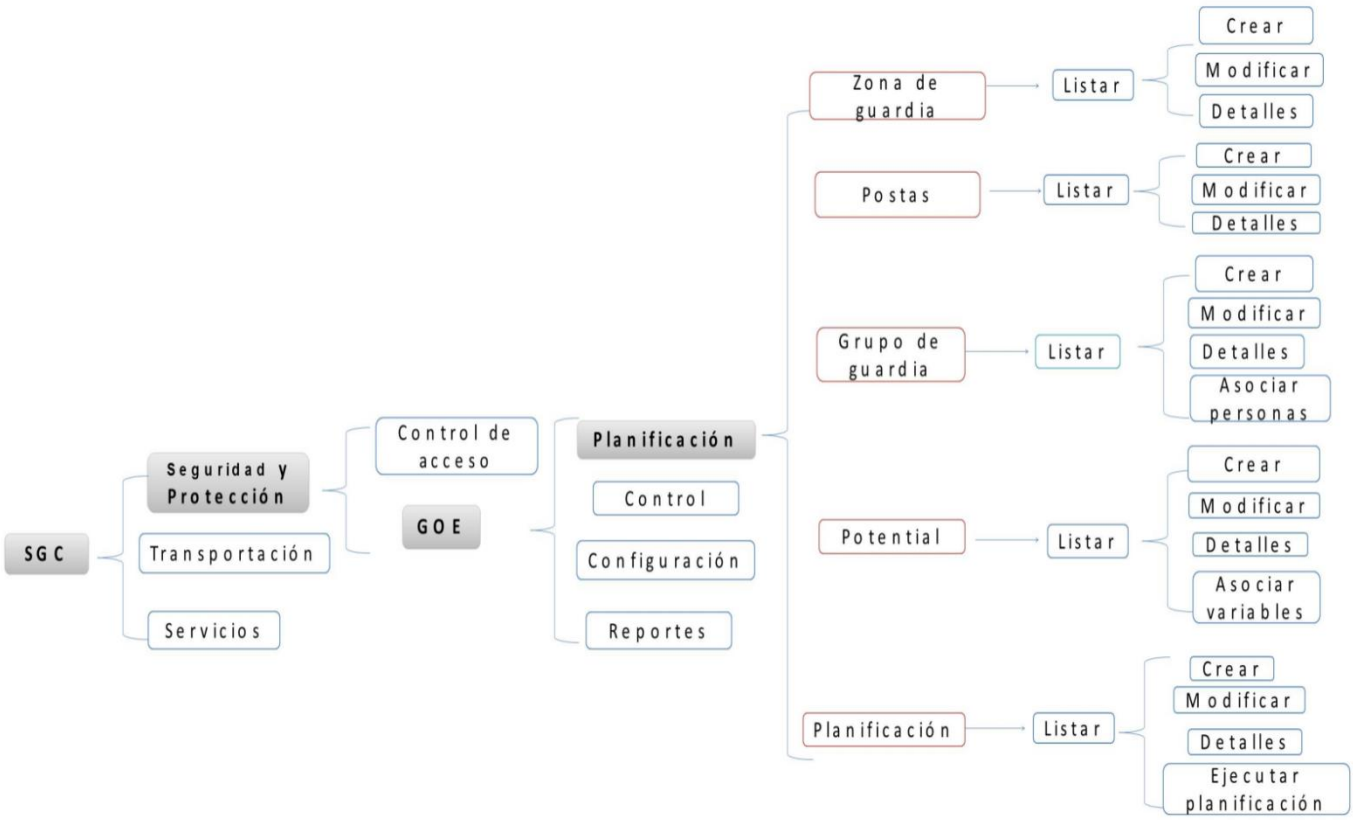


Figura 4: Mapa de navegación de la agrupación funcional Planificación.

2.6 Descripción de la arquitectura

La **arquitectura cliente-servidor** es un modelo de aplicación distribuida en el que las tareas se reparten entre los proveedores de recursos o servicios, llamados servidores, y los demandantes, llamados clientes. Un cliente realiza peticiones a otro programa y el servidor es quien le da respuesta. Esta idea también se puede aplicar a programas que se ejecutan sobre una sola computadora, aunque es más ventajosa en un sistema operativo multiusuario distribuido a través de una red de computadoras. La interacción cliente-servidor es el soporte de la mayor parte de la comunicación por redes. Ayuda a comprender las bases sobre las que están construidos los algoritmos distribuidos(31). En la *figura 5* se muestra el funcionamiento de la arquitectura cliente-servidor.

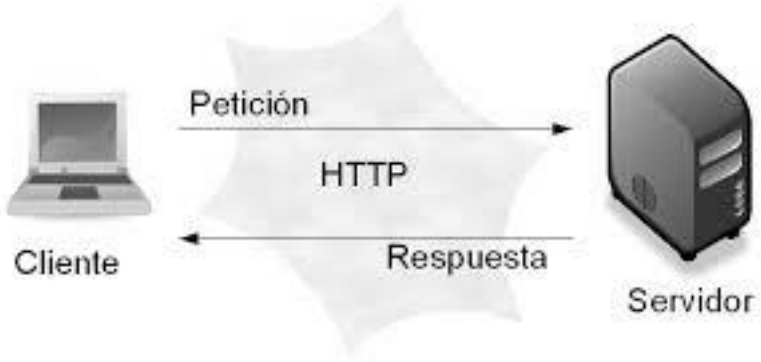


Figura 5: Arquitectura Cliente-Servidor (32).

Cliente: Programa ejecutable que participa activamente en el establecimiento de las conexiones. Envía una petición al servidor y se queda esperando por una respuesta. Su tiempo de vida es finito una vez que son servidas sus solicitudes, termina el trabajo (31).

Servidor: Es un programa que ofrece un servicio que se puede obtener en una red. Acepta la petición desde la red, realiza el servicio y devuelve el resultado al solicitante. Al ser posible implantarlo como aplicaciones de programas, puede ejecutarse en cualquier sistema donde exista TCP/IP y junto con otros programas de aplicación. El servidor empieza su ejecución antes de comenzar la interacción con el cliente. Su tiempo de vida o de interacción es interminable (31).

2.7 Patrones arquitectónicos

Los **patrones arquitectónicos** o **patrones de arquitectura** ofrecen soluciones a problemas de arquitectura de software en ingeniería de software. Dan una descripción de los elementos y el tipo de relación que tienen unido a un conjunto de restricciones sobre cómo pueden ser usados. Un patrón arquitectónico expresa un esquema de organización estructural esencial para un sistema de software, que consta de subsistemas, sus responsabilidades e interrelaciones. En comparación con los patrones de diseño, los patrones arquitectónicos tienen un nivel de abstracción mayor.

Dentro de los patrones arquitectónicos se encuentra el patrón Modelo-Vista-Controlador.

- **Modelo-Vista-Controlador (MVC)**

El patrón Modelo-Vista-Controlador separa los datos de una aplicación, la interfaz de usuario y la lógica de control en tres componentes distintos. Se ve frecuentemente en aplicaciones *web*, donde la vista es la

página HTML y el código que provee datos dinámicos a la página, el modelo es el sistema de gestión de base de datos y el controlador representa la lógica de negocio (33).

- **Modelo:** Es la capa donde se trabaja con los datos, por tanto, contendrá mecanismos para acceder a la información y también para actualizar su estado. Los datos se encuentran en una base de datos, por lo que en los modelos se localizan todas las funciones que accederán a las tablas.
- **Vista**: Las vistas contienen el código de la aplicación que va a producir la visualización de las interfaces de usuario, o sea, el código que permitirá renderizar los estados de nuestra aplicación en HTML. En las vistas nada más se tiene el código HTML y twig que permite mostrar la salida.
- **Controlador:** Contiene el código necesario para responder a las acciones que se solicitan en la aplicación, como visualizar o modificar un elemento, crear una posta o realizar una búsqueda de información.

  En la *figura 6* se muestra cómo funciona el patrón MVC.

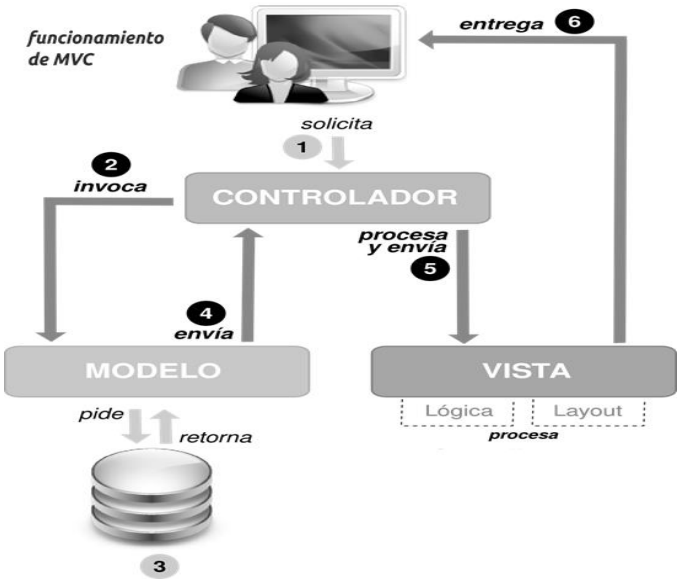


Figura 6: Patrón arquitectónico Modelo-Vista-Controlador (34).

❖ **Funcionamiento del patrón MVC**

1. El usuario realiza una solicitud al sistema. Generalmente estará desencadenada por acceder a una página del mismo. Esta solicitud le llega al controlador.

2. El controlador se comunica tanto con modelos como con vistas. A los modelos les solicita datos o les manda realizar actualizaciones de los datos. A las vistas les solicita la salida correspondiente, una vez se hayan realizado las operaciones pertinentes según la lógica del negocio.
3. Para producir la salida, en ocasiones las vistas pueden solicitar más información a los modelos. En ocasiones, el controlador será el responsable de solicitar todos los datos a los modelos y de enviarlos a las vistas, haciendo de puente entre unos y otros.
4. Las vistas envían al usuario la salida. Aunque en ocasiones esa salida puede ir de vuelta al controlador y sería éste el que hace el envío al cliente.

❖ **Funcionamiento del patrón arquitectónico MVC en el Componente para la Gestión de la Guardia Obrera Estudiantil**

Cuando el usuario accede a algún módulo para solicitar alguna información a través de una ruta, el sistema determina cuál Controlador es el que está asociado a la ruta especificada. En el Componente para la Gestión de la Guardia Obrera Estudiantil, los Controladores se encuentran en las carpetas Controller pertenecientes a cada bundle. Symfony 2 ejecuta el Controlador correspondiente a la ruta. El Controlador solicita al Modelo los datos de la clase. Una vez que el Modelo devuelve dichos datos, el Controlador solicita a la Vista que cree una página mediante una plantilla y que inserte los datos obtenidos por el Modelo. El Controlador entrega al servidor los datos de la página creada por la Vista, la cual se le muestra al usuario.

La *figura 7* muestra la forma en que se aplican los principios de la arquitectura MVC en el Componente para la Gestión de la Guardia Obrera Estudiantil.

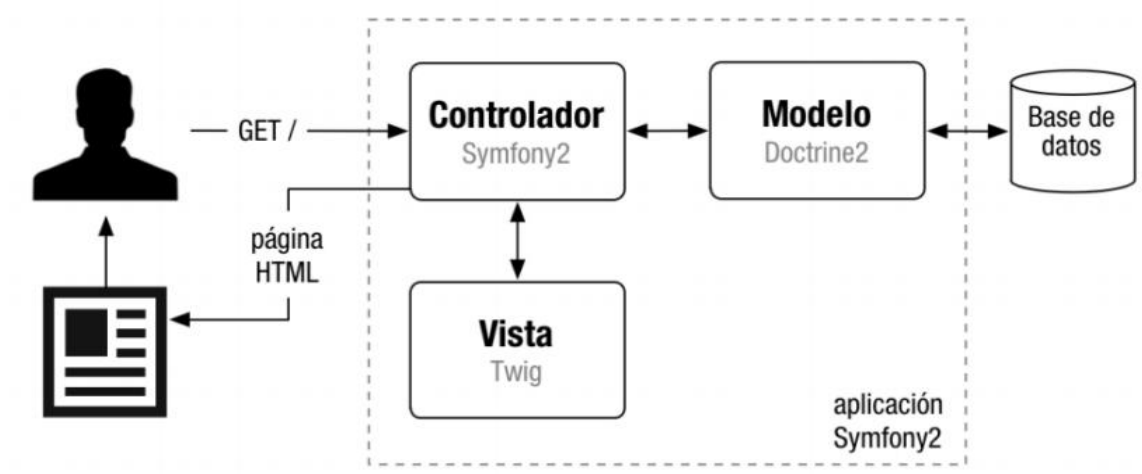


Figura 7: Patrón arquitectónico MVC con Symfony 2 (35).

2.8 Patrones de diseño

Un patrón de diseño es una descripción de clases y objetos comunicándose entre sí, adaptada para resolver un problema de diseño general en un contexto particular. El mismo identifica: clases, instancias, roles, colaboraciones y la distribución de responsabilidades. Estos modelos que se presentan como parejas de problema/solución con un nombre, codifican buenos principios y sugerencias relacionados con la asignación de responsabilidades, basados en la recopilación del conocimiento de los expertos en desarrollo de *software* (36) .

Entre los más usados se encuentran los patrones de Asignación de Responsabilidades (GRASP) y los patrones de la Pandilla de los Cuatro (GoF).

Los patrones de diseño tienen como características principales:

- Son soluciones concretas. Proponen soluciones a problemas concretos, no son teorías genéricas.
- Son soluciones técnicas. Indican resoluciones técnicas basadas en la Programación Orientada a Objetos. En ocasiones tienen más utilidad con algunos lenguajes de programación y en otras son aplicables a cualquier lenguaje.
- Se utilizan en situaciones frecuentes. Debido a que se basan en la experiencia acumulada al resolver problemas reiterativos.

- Favorecen la reutilización de código. Ayudan a construir software basado en la reutilización, a construir clases reutilizables. Los propios patrones se reutilizan cada vez que se vuelven a aplicar.

❖ **Patrones de diseño GRASP**

Los patrones GRASP representan los principios básicos de la asignación de responsabilidades a objetos, expresados en forma de patrones. GRASP es el acrónimo para General Responsibility Assignment Software Patterns (Patrones Generales de Software para Asignar Responsabilidades) (36).

■ **Experto**

Permite asignar una responsabilidad al experto en información, o sea, a la clase que cuenta con la información necesaria para cumplir con una determinada responsabilidad. Symfony2 utiliza este patrón con la inclusión de Doctrine para el mapeo de bases de datos. Se utiliza específicamente para crear una capa de abstracción en el modelo, encapsular toda la lógica de los datos y generar las clases entidades con todas las funcionalidades comunes (GET, SET y el constructor de la entidad); las clases de abstracción de datos poseen un grupo de funcionalidades que están relacionadas directamente con la entidad que representan y contienen la información necesaria de la tabla asociada.

Beneficios del patrón Experto:

- Se conserva el encapsulamiento, debido a que los objetos se valen de su propia información para realizar todas las peticiones. Esto posibilita tener sistemas de fácil mantenimiento.
- El comportamiento se distribuye entre las clases que cuentan con la información requerida para cumplir con la responsabilidad asignada.

En la *figura 8* se muestra un ejemplo de la utilización del patrón en la entidad *DPosta* donde se maneja la información referente a la tabla *sq_goe.tb_dposta.*

DPosta.php × | DAsigancion.php × | DPlanificacion.php × | NEstadoPlan.php × | DRegionGuardia.php × | RTipoRecursoPosta.php

I\DIN\SProt\GOE\GestionBundle\Entity\DPosta

```php
<?php
namespace UCI\DIN\SProt\GOE\GestionBundle\Entity;

use Doctrine\Common\Collections\ArrayCollection;
use Doctrine\ORM\Mapping as ORM;

/** Class DPosta ...*/
class DPosta
{
    /** @var integer ...*/
    private $idPosta;
    /** @var string ...*/
    private $nombrePosta;
    /** @var string ...*/
    private $localizacion;
    /** @var string ...*/
    private $descripcion;
    /** @var bool ...*/
    private $activo = true;
    /** @var integer ...*/
    private $cantidadPersonas;
    /** @var DRegionGuardia ...*/
    private $regionGuardia;

    /** @var \Doctrine\Common\Collections\Collection ...*/
    private $tipoRecursosPosta;

    public function __construct(){...}
    /** @return int ...*/
    public function getIdPosta(){...}
    /** @return string ...*/
    public function getNombrePosta(){...}
    /** @param string $nombrePosta ...*/
    public function setNombrePosta($nombrePosta){...}
```

Figura 8: Patrón Experto.

- **Creador**

Su implementación permite identificar quién debe ser el responsable de la creación o instanciación de nuevos objetos o clases. Dicho de otra manera, este patrón plantea que se debe asignar a una clase A la responsabilidad de crear una instancia de una clase B. Como la creación de objetos es una de las actividades más frecuentes en un sistema orientado a objetos, es importante el uso de este para guiar la asignación de responsabilidades relacionadas con la creación de objetos. Este patrón es utilizado en la implementación de las clases controladoras donde se encuentran las acciones definidas para el sistema. En dichas acciones se crean los objetos de las clases que representan las entidades.

Beneficios del patrón Creador:

- Se crean menos dependencias y existe mayor posibilidad de reutilización de código.

En la *figura 9* se muestra un ejemplo de cómo se utiliza este patrón la propuesta de solución.

```php
<?php
namespace UCI\DIN\SProt\GOE\GestionBundle\Controller;

use ...

class DPostaController extends AppController
{
    /**
     * Listar todos los registros de la entidad DPosta
     *
     */
    public function listarAction()
    {
        $filtros = DashBoardFilter::create($this->container, 'grid_dposta', array(), new Search('nombre dposta', 'app.term.buscar'));
        return $this->render('GOEGestionBundle:dposta:index.html.twig', array('filtros' => $filtros));
    }

    /** @param Request $request ...*/
    public function gridDataAction(Request $request){...}

    /**
     * Crear DPosta
     *
     */
    public function nuevoAction(){...}

    /** @param Request $request ...*/
    public function guardarNuevoAction(Request $request){...}
```

Figura 9: Patrón Creador.

- **Controlador**

El patrón controlador sirve como intermediario entre una determinada interfaz y el algoritmo que la implementa, de tal forma que es el que recibe los datos del usuario y los envía a las distintas clases según el método encargado de la ejecución. Se manifiesta en todo el sistema debido a que cada uno de los eventos generados por el usuario son atendidos por el archivo *routing.yml,* encargado de redirigir la petición al método de una clase controladora que realice las operaciones solicitadas, pero siempre manteniendo las clases controladoras sin sobrecarga, es decir, manteniendo siempre la alta cohesión. En la *figura 10* se muestra un ejemplo de la utilización del patrón en el archivo *posta.yml* de la propuesta de solución.

```
sprot_goe_posta_listar:
    path:     /listar
    defaults: { _controller: "GOEGestionBundle:DPosta:listar" }
sprot_goe_gestion_posta_grid:
    path:     /grid
    defaults: { _controller: "GOEGestionBundle:DPosta:gridData" }
    options:
        expose: true
sprot_goe_gestion_posta_detalles:
    path:     /{id}/show
    defaults: { _controller: "GOEGestionBundle:DPosta:detalles" }
    options:
        expose: true
sprot_goe_gestion_posta_nuevo:
    path:     /nuevo
    defaults: { _controller: "GOEGestionBundle:DPosta:nuevo" }
sprot_goe_gestion_posta_guardar_nuevo:
    path:     /guardarNuevo
    defaults: { _controller: "GOEGestionBundle:DPosta:guardarNuevo" }
    options:
        expose: true
sprot_goe_gestion_posta_tipo_recurso:
    path:     /gridRecursos
    defaults: { _controller: "GOEGestionBundle:DPosta:tipoRecursoGrid" }
    options:
        expose: true
sprot_goe_gestion_posta_tipo_recurso_valor:
    path:      /gridRecursosValor
    defaults: { _controller: "GOEGestionBundle:DPosta:tipoRecursoValorGrid" }
    options:
        expose: true
```

Figura 10: Patrón Controlador.

- **Alta Cohesión**

La alta cohesión hace referencia a cuanta responsabilidad tiene una determinada clase controladora, o sea, una clase debe tener responsabilidades moderadas. Una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme.

Symfony 2 permite la organización del trabajo en cuanto a la estructura del proyecto y la asignación de responsabilidades con alta cohesión. Se puede observar en el sistema, puesto que, cada clase controladora maneja solamente las responsabilidades correspondientes a las entidades con las que se relaciona, además para cada vista existe una página controladora encargada de manejar sus solicitudes.

Beneficios del patrón Alta cohesión:

- Mejora la claridad y la facilidad con que se entiende el diseño.
- Se simplifica el mantenimiento y las mejoras en funcionalidad.

- **Bajo Acoplamiento**

Este patrón plantea que se deben asignar las responsabilidades de forma tal que las clases dependan del menor número de clases que sea posible. Resuelve el problema de cómo dar soporte a una dependencia escasa y a un aumento de la reutilización. El acoplamiento es una medida de la fuerza con que una clase está conectada a otras, con que las conoce y con que recurre a ellas.

En Symfony2 las clases controladoras heredan de la clase Controller alcanzando de esta manera un bajo acoplamiento. Las clases que implementan la lógica del negocio y de acceso a datos se encuentran en el modelo y no tienen asociaciones con las de la vista o el controlador, lo que proporciona bajas dependencias.

Beneficios del patrón Bajo acoplamiento:

- Clases fáciles de entender por separado.
- Clases fáciles de reutilizar.

❖ **Patrones de diseño GoF**

Los patrones GoF (Gang of Four, en español Pandilla de los Cuatro, formada por Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides) se clasifican en 3 categorías basadas en su propósito: creacionales, estructurales y de comportamiento (36).

- **Decorador**

El decorador extiende la funcionalidad de un objeto dinámicamente de tal modo que es transparente a sus clientes, utilizando una instancia de una subclase de la clase original que delega las operaciones al objeto original. Este provee una alternativa muy flexible para agregar funcionalidad a una clase. En Symfony2 este patrón es fácilmente visible, ya que, la vista se separa por niveles, en hasta 3 niveles, una plantilla base y varias plantillas que heredan de esta. Normalmente, la plantilla base es global en toda la aplicación y contiene el código HTML que es común en la mayoría de las páginas.

2.9 Patrones de diseño de Base Datos.

Los patrones de diseño de una Base de Datos permiten al usuario crear una base de datos más fortalecida ya que constituyen una guía que especifica cómo debe ser la misma. El diseño y construcción de una base de datos requiere del mayor esfuerzo y análisis posible, ya que a partir de este diseño es que esta se crea y la calidad con que se obtenga determinará su comportamiento futuro (37).

- **Entidad-Atributo-Valor:** Es la representación de un modelo flexible donde se pueden representar objetos con sus atributos. Es un acercamiento al modelo orientado a objeto representado en el

modelo relacional, donde la entidad *Class* representa las clases, la entidad *Attribute* representa los atributos de las clases, por su parte la entidad *Object* representa las instancias de las clases, mientras que la entidad *Value* representa los valores de cada atributo para cada objeto dado (38).

- **Patrón llave subrogada:** Este patrón es muy utilizado porque facilita la interacción con la base de datos en un futuro. El mismo plantea que se genere una llave primaria única para cada entidad, en vez de usar un atributo identificador en el contexto dado. Normalmente se usan enteros en columnas autoincrementales o GUID[21] que están demostradas que no se repiten o con una probabilidad extremadamente baja. Esto permite que las tablas sean más fáciles de consultar a partir del identificador, pues todos tienen el mismo tipo en cada una de las tablas (39). La utilización del patrón está presente en la tabla *sq_goe.tb_rtiporecurso_posta*, en la cual se utiliza como llave un campo numérico auto incrementable llamado *id_rtiporecurso_posta.*

2.10 Modelo de Base de Datos.

Un modelo de datos es un conjunto de conceptos, reglas y convenciones que permiten definir los datos del universo de discurso, además de especificar y manipular los datos que se quieren almacenar en la base de datos (40) .

Así, un modelo de datos consiste:

- Objetos: Entidades que existen y se manipulan.
- Atributos: Características básicas de estos objetos.
- Relaciones: Forma en que enlazan los distintos objetos entre sí (40) .

Para consultar el modelo de datos de la propuesta de solución ver anexo 3.

[21] **GUID**: Por sus siglas en inglés significa Identificador Único Global, es un número pseudo-aleatorio empleado en aplicaciones de *software*. Aunque no se puede garantizar que cada GUID generado sea único, el número total de claves únicas es tan grande que la posibilidad de que se genere un mismo número dos veces puede considerarse nula en la práctica.

2.11 Modelo de despliegue

El diagrama de despliegue permite visualizar la estructura física del sistema, teniendo en cuenta la estrecha relación entre el *hardware* y el *software* que se despliega. Dicha relación se representa mediante los protocolos de comunicación que se utilizan para transferir información entre ellos. En la *figura 11* se muestra el diagrama de despliegue definido para la solución propuesta.

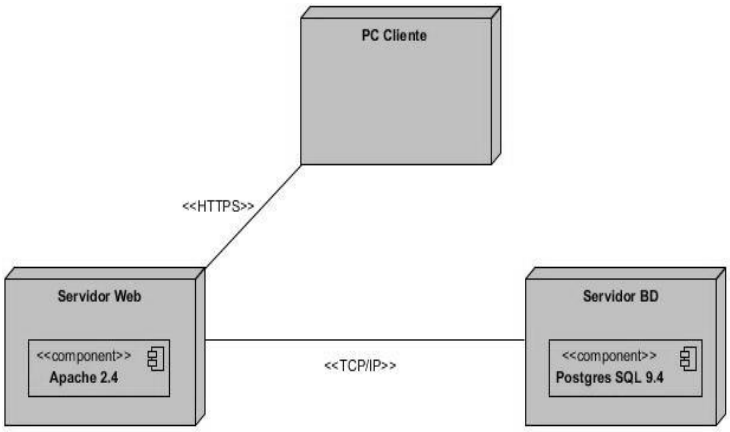


Figura 11: Diagrama de despliegue de la propuesta de solución.

El diagrama de despliegue representado muestra la siguiente distribución:

PC-Cliente: Dispositivo que permite a los usuarios acceder a la aplicación, de manera que estos puedan interactuar con el sistema de acuerdo a sus necesidades actuales.

Servidor web: Ordenador donde se localiza el servidor web Apache. Este constituye la zona donde se gestione todo el contenido de la aplicación y las máquinas clientes podrán acceder a través de un navegador web.

Servidor BD: Es el responsable de recopilar toda la información generada por el sistema. En él se encuentra situada la base de datos y de ella los servicios que brinda el componente, se nutren de información.

HTTPS: Protocolo de transferencia de hipertexto seguro, por sus siglas en inglés, *Hypertext Transfer Secure Protocol* (HTTPS), es un protocolo de red basado en HTTP por lo que está orientado a transacciones sin estado, es decir, que no guarda ninguna información sobre conexiones anteriores y sigue el esquema petición-respuesta entre un cliente y un servidor (41).

TCP/IP: Base de Internet y sirve para enlazar computadoras que utilizan diferentes sistemas operativos. Familia de protocolos utilizada para la conexión entre el servidor web y el servidor de datos donde se encuentra ubicada la base de datos (41).

2.12 Conclusiones parciales

Una vez realizado el análisis y diseño del componente propuesto se concluye que los requisitos funcionales y no funcionales identificados a partir del proceso de obtención de los requisitos y los artefactos generados, constituyeron la base principal para la construcción de la solución. Con la arquitectura Modelo-Vista-Controlador y el uso de los patrones de diseño y de base de datos, se garantiza una mejor organización de la arquitectura del sistema. Por otro lado, con la realización del modelo físico de datos se tiene definida la estructura de la base de datos a utilizar y con la realización del modelo de despliegue se representa la interacción de los diferentes componentes que intervienen en la solución.

Capítulo 3: Construcción y validación del componente

3.1 Introducción

En el presente capítulo se describe la implementación del software, fase donde finalmente se complementa el producto final y se cumple con los requisitos definidos al inicio de la investigación. Para ellos se definen los estándares de codificación que debe cumplir el equipo de desarrollo y se crea el diagrama de componentes, además de presentarse las principales interfaces de usuario de la solución. A partir del código resultante y su funcionamiento se ejecutan las pruebas unitarias, de validación y del sistema al componente desarrollado.

3.2 Estilos de programación

En la disciplina de implementación se lleva a cabo la producción del software. El equipo de desarrollo se encarga de implementar las funcionalidades, especificadas por el cliente en las descripciones de requisitos, utilizando el lenguaje de programación elegido. Para el desarrollo del componente se utilizaron dos estilos de programación, la programación orientada a objetos y la programación dirigida por eventos incluidos en el marco de trabajo Symfony. La programación orientada a objetos: es un paradigma de programación que usa objetos y sus interacciones para diseñar aplicaciones y programas de computadora. Está basado en varias técnicas, incluyendo herencia, modularidad, polimorfismo y encapsulamiento (42). El marco de trabajo Symfony al desarrollarse con la tecnología PHP, permite reutilizar el código utilizando el paradigma de programación orientado a objetos que ya tiene implementado. En él, cada objeto es responsable de inicializarse y destruirse de forma correcta, no existe la necesidad de llamar explícitamente al procedimiento de creación o de terminación debido a que el marco de trabajo se encarga de esto. Una de las principales características que proporciona es la herencia, lo que permite la reutilización del código y Symfony la utiliza entre los distintos tipos de clases como las modelos y las controladoras.

La programación dirigida por eventos: es un paradigma de programación en el que tanto la estructura como la ejecución de los programas van determinadas por los sucesos que ocurran en el sistema, definidos por el usuario o que el sistema mismo provoque (43). Tanto el marco de trabajo como la biblioteca JQuery implementan este paradigma. Esta librería es usada para el manejo de las interfaces de usuario y utiliza los principales eventos que provee el navegador para la interacción de sus componentes.

Estos pueden ser desencadenados por el usuario o por otros eventos en el sistema, permitiendo una mayor interoperabilidad usuario-sistema.

3.3 Estándares de codificación

Un estándar de codificación completo comprende todos los aspectos de la generación de código. Un código fuente completo debe quedar como si un único programador hubiera escrito cada línea de una sola vez (44). Es muy buena práctica, establecer estándares de codificación para garantizar que los programadores realicen un trabajo coherente. Emplear diferentes técnicas de codificación sólidas contribuye a mejorar la calidad del software y obtener mejores índices de rendimiento.

Los estándares de codificación son un complemento a la programación por pares, o sea, en equipo, y no sólo es importante usar un estándar, sino usar un buen estándar de codificación, de esta forma se deberá:

- Clarificar más que confundir.
- Promover la intención del código.
- Permitir que los programas se acerquen lo mejor posible al lenguaje natural.
- Incorporar las mejores prácticas de la codificación.

❖ **Indentación, llaves de apertura y cierre, y tamaño de las líneas**

Se debe utilizar una indentación[22] sin tabulaciones con un equivalente a cuatro espacios. En la *figura 12* se muestra que el uso de las llaves "{}" será en una nueva línea y la longitud de las líneas de código es aproximadamente de 75 y 80 caracteres para mantener la legibilidad del código.

```
public function indexAction()
{
    $var = $this->get('goe_configuracion.estado_plan')->obtenerVariablesListar();
    return $this->render('GOEConfiguracionBundle:Nomenclador:index.html.twig', array('path_js' => $var['path_js'], 'url' => $var['url'], 'url_edit' => $var['url_edit']
        /* , 'url_detail' => $var['url_detail'] */, 'campos_adicionales' => json_encode($var['campos_adicionales'], 7), 'nombre_view' => $var['nombre_view'],
        'path_create' => $var['path_create'], 'grid_title' => $var['grid_title']));
}
```

Figura 12: Estándar para indentación, llaves y tamaño de líneas.

[22] **Indentación:** En los lenguajes de programación es un tipo de notación secundaria utilizado para mejorar la legibilidad del código fuente por parte de los programadores.

❖ **Convención de nomenclatura**

Variables: se rigen por la nomenclatura camelCase[23]. Siempre comienzan con minúscula y en caso de nombres compuestos la primera letra de cada palabra comienza con mayúscula.

Clases: En la *figura 13* se muestra que las clases siempre comienzan con mayúscula, en caso de nombre compuesto las palabras no se separan y comienzan igualmente con mayúscula.

```
class TipoRecursoType extends BaseNomencladorType
{
    public function __construct(ContainerInterface $container){
        parent::__construct($container, 'UCI\DIN\SProt\GOE\ConfiguracionBundle\Entity\NTipoRecurso');
    }
}
```

Figura 13: Ejemplos de nomenclatura para clases.

Funciones: se rigen por la nomenclatura camelCase. Siempre comienzan con minúscula y en caso de nombres compuestos la primera letra de cada palabra comienza con mayúscula. Los parámetros son separados por una coma seguida de un espacio entre dos consecutivos. En la *figura 14* se muestran varios ejemplos de nomenclatura.

```
public function tipoRecursoGridAction(Request $request)
{

    $service = $this->get('base.util_grid');
    $post = $request->request->all();
    $filtros = isset($post['filtros']) ? $post['filtros'] : array();
    return new Response($service->grid('GOEConfiguracionBundle:NTipoRecurso', 'obtenerTipoRecursosActivos', $post, $filtros));
}
```

Figura 14: Ejemplos de nomenclatura para funciones.

Estructuras de control: se incluye un espacio entre las estructuras de control (if, for, foreach, while, switch) y los paréntesis. Se recomienda utilizar siempre llaves de apertura y cierre, incluso en situaciones en las que técnicamente son opcionales. Esto aumenta la legibilidad y disminuye la probabilidad de errores lógicos. Si las condiciones son muy largas que sobrepasan el tamaño de la línea, estas se dividen

[23] ***camelCase*:** Es un estilo de escritura que se aplica a frases o palabras compuestas. El nombre se debe a que las mayúsculas a lo largo de una palabra en *camelCase* se asemejan a las jorobas de un camello.

en varias líneas. Se emplea una declaración por línea lo que facilita los comentarios. Se añade un solo espacio alrededor de los operadores (==, &&) y una coma después de cada elemento del arreglo en un arreglo multilínea, incluso después del último.

- **Documentación**

Todos los archivos deben de tener la documentación asociada al mismo. Se debe de cumplir con el siguiente bloque al principio de cada clase. En la *figura 15* se muestra un ejemplo en la clase NTipoRecurso.

```
/**
 * @ParamConverter("tipo_recurso", class="GOEConfiguracionBundle:NTipoRecurso")
 * @author Raul Bach Eng
 * @param \Symfony\Component\HttpFoundation\Request $request
 * @return mixed
 */
public function crearAction(Request $request){...}
```

Figura 15: Documentación de las funciones.

- **Comentarios**

Los comentarios en el código, representan la documentación interna más precisa de un software, garantizan el entendimiento de lo que realmente realiza un determinado bloque de código, evitando confusiones y agilizando considerablemente las tareas de revisión y mantenimiento. Para la inclusión de comentarios es necesario respetar algunas reglas básicas, como son: abreviar el contenido de los comentarios, usar un lenguaje técnico y entendible, evitando el uso de vocablos rebuscados y emplear un estilo uniforme de comentarios estándares para todo el equipo. Para la implementación: deberá incluirse un espacio simple entre los caracteres "//" y el texto del comentario.

3.4 Validación de requisitos

La validación de requisitos examina las especificaciones para asegurar que todos los requisitos del sistema han sido establecidos sin ambigüedad, sin inconsistencias, sin omisiones, que los errores detectados hayan sido corregidos y que el resultado del trabajo se ajusta a los estándares establecidos para el proyecto y el producto (45).

- **Criterios para validar los requisitos**

Para validar los requisitos del sistema según Pressman[24] se pueden chequear mediante un cuestionario guiado por un conjunto de interrogantes con el objetivo de descubrir la mayor cantidad de errores posibles. A continuación, se muestra las preguntas utilizadas, algunas fueron agregadas a las planteadas por Pressman por ser consideradas necesarias para la validación.

Interrogantes para la validación de requisitos:

¿Está el requisito claramente definido? ¿Puede interpretarse mal?

¿Está identificado el origen del requisito (por ejemplo: persona, norma, documento)? ¿El planteamiento final del requisito ha sido contrastado con la fuente original?

¿El requisito está delimitado en términos cuantitativos?

¿Qué otros requisitos hacen referencia al requisito estudiado?

¿El requisito incumple alguna restricción definida?

¿El requisito es verificable? Si es así, ¿se pueden efectuar pruebas para verificar el requisito?

¿Está el requisito asociado con los rendimientos del sistema o con su comportamiento?

¿El requisito está implícitamente definido?

¿El requisito es modificable?

¿El requisito está completo?

¿El requisito puede ser implementado?

¿El requisito puede ser probado?

¿El resultado de la evaluación de impacto es positivo?

[24] **Roger Pressman:** es una autoridad internacionalmente reconocida en la mejora de procesos de software y en tecnologías de ingeniería de software. Por más de tres décadas, ha trabajado como ingeniero, gerente, profesor, autor y consultor de software en temas de ingeniería de software.

❖ **Técnicas de validación de requisitos**

- **Prototipado de interfaz:** Facilita al cliente la interacción con el sistema permitiendo que este se familiarice con la interfaz de usuario que tendrá la propuesta de solución. Existen dos tipos principales de prototipo de interfaz: *Desechables* se utilizan sólo para la validación de los requisitos y posteriormente se desechan. Pueden ser prototipos en papel o en *software*. *Evolutivos* una vez utilizados para la validación de los requisitos, se mejora su calidad y se convierten progresivamente en el producto final (46).
- **Revisiones de requisitos**: se realizan reuniones para analizar los requisitos, ayudando a obtener los posibles errores que pueden existir en la especificación de los requisitos del software o para confirmar que los requisitos poseen la calidad deseada.
- **Generación de casos de prueba:** la elaboración de casos de prueba basados en requisitos, permiten analizar el conjunto de entradas, condiciones de ejecución y resultados esperados a partir de los cuales se puede determinar si el requisito de una aplicación es parcial o completamente satisfactorio.

La evaluación de los requisitos permitió identificar deficiencias en las descripciones de requisitos, así como desechar requisitos que no se consideraban necesarios en la implementación de la solución.

Resultados de aplicar las técnicas de validación

1. Descripciones de requisitos muy generales.
2. Faltas de ortografía.
3. Errores en la construcción de los prototipos de interfaz.

3.5 Estrategias de pruebas

Una estrategia de prueba del software integra las técnicas de diseño de casos de prueba en una serie de pasos bien planificados que dan como resultado una correcta construcción del software. Debe ser suficientemente flexible para promover la creatividad y la adaptabilidad necesarias para adecuar la prueba a todos los grandes sistemas basados en software. Al mismo tiempo, la estrategia debe ser suficientemente rígida para promover un seguimiento razonable de la planificación y la gestión a medida que progresa el proyecto.

Para evaluar la calidad del componente que se está desarrollando y comprobar el cumplimiento de los objetivos propuestos, se aplicaron un conjunto de pruebas que verifican la eficacia del software.

Para comprobar que la solución implementada es válida y que cumple con los requisitos acordados con el cliente, se propone la siguiente estrategia de pruebas:

Tabla 5: Pruebas de software.

Nivel de prueba	Tipo de prueba	Método	Técnica
Integración	Estructural	Caja negra	Incremental
Sistema	Rendimiento y resistencia	Caja negra	Automática
	Funcional	Caja negra	Particiones equivalentes

Pruebas de caja blanca: se centran en la estructura de control del programa. Mediante los métodos de prueba de caja blanca, el ingeniero del software puede obtener casos de prueba que garanticen que se ejercita por lo menos una vez todos los caminos independientes de cada componente, ejerciten todas las decisiones lógicas en sus vertientes verdadera y falsa, ejecuten todos los bucles en sus límites y con sus límites operacionales y ejerciten las estructuras internas de datos para asegurar su validez (47).

Pruebas de caja negra: son diseñadas para validar los requisitos funcionales sin fijarse en el funcionamiento interno de un programa. La prueba de caja negra permite al ingeniero del software obtener conjuntos de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa. La prueba de caja negra no es una alternativa a las técnicas de prueba de caja blanca; se trata de un enfoque complementario que intenta descubrir diferentes tipos de errores que los métodos de caja blanca no verifican, como son errores de funciones incorrectas o ausentes, errores de interfaz, errores en estructuras de datos o en accesos a bases de datos externas, errores de rendimiento y errores de inicialización y de terminación. Las técnicas de prueba de caja negra se centran en el ámbito de información de un programa, de forma que se proporcione una cobertura completa de la prueba (47).

- **Pruebas de integración**

Las pruebas de integración son una técnica sistemática para construir la arquitectura de un software a la vez que se aplican las pruebas para encontrar errores asociados a las interfaces (47). Además, comprueban que los componentes integrados realmente funcionan juntos, que sean llamados correctamente y que transfieran los datos correctos en el tiempo preciso a través de sus interfaces. Existen dos tipos de integración: incremental y no incremental. La integración no incremental consiste en combinar los componentes como un todo y probar el sistema. La integración incremental se fundamenta

en dividir la integración en pequeños incrementos (componentes) y realizar la integración por separado, propiciando que sea más fácil aislar y corregir los errores.

Resultados de las pruebas de integración

En la validación de la solución se probó la integración del sistema con los componentes del SGC, en este caso con los componentes Seguridad y Base de este último se obtienen las personas y su estructura. La *Tabla 6* muestra el caso de prueba de integración realizado al componente de Seguridad; el caso de prueba al componente Base se puede observar en el *Anexo 6.* Al finalizar las pruebas de integración no se detectaron errores asociados a la interacción entre el Componente para la Gestión de Guardia Obrera Estudiantil con los componentes del SGC.

Tabla 6: Prueba de integración del componente Seguridad.

Caso de prueba: Integración con el componente Seguridad	
Componente al que se integra	Seguridad.
Condiciones de ejecución	El componente seguridad haya introducido los datos en la base de datos central y exista conexión con la misma.
Descripción de la prueba	Comprobar que el Componente para la Gestión de Guardia Obrera Estudiantil es capaz de realizar el acceso a funcionalidades y seguridad de negocio a partir de la información gestionada por el componente Seguridad.
Entradas/Pasos de ejecución	El componente de Seguridad introduce en la base de datos central los datos y el Componente para la Gestión de Guardia Obrera Estudiantil consulta estos datos y define la seguridad de este componente.
Resultado esperado	Los usuarios tienen acceso a las funcionalidades de acuerdo al rol y sus responsabilidades.
Evaluación	Prueba satisfactoria.

- **Pruebas de sistema**

La prueba del sistema, realmente, está constituida por una serie de pruebas diferentes cuyo propósito primordial es ejercitar profundamente el sistema basado en computadora. Aunque cada prueba tiene un propósito diferente, todas trabajan para verificar que se han integrado adecuadamente todos los elementos del sistema y que realizan las funciones apropiadas (48).

- Pruebas de resistencia

Las pruebas de resistencia están diseñadas para enfrentar a los programas con situaciones anormales. Estas pruebas se ejecutan en el sistema de forma que demande recursos en cantidad, frecuencia o volúmenes anormales. Por ejemplo: -1- diseñar pruebas especiales que generen diez interrupciones por segundo, cuando las normales son una o dos; -2- incrementar las frecuencias de datos de entrada en un orden de magnitud con el fin de comprobar cómo responden las funciones de entrada; -3- ejecutar casos de prueba que requieran el máximo de memoria o de otros recursos; -4- diseñar casos de prueba que puedan dar problemas en un sistema operativo virtual o -5- diseñar casos de prueba que produzcan excesivas búsquedas de datos residentes en disco. Esencialmente, el responsable de la prueba intenta romper el programa (49).

- Pruebas de rendimiento

La prueba de rendimiento está diseñada para probar el rendimiento del software en tiempo de ejecución dentro del contexto de un sistema integrado. La prueba de rendimiento se da durante todos los pasos del proceso de la prueba y a menudo, van emparejadas con las pruebas de resistencia (49).

Resultados

Se efectuaron las pruebas para observar el comportamiento de la aplicación bajo una cantidad de peticiones esperada. Para llevar a cabo las pruebas de rendimiento y resistencia se utilizó la herramienta JMeter 2.8.4 y se hizo uso de una copia local del sistema en una computadora con un microprocesador Dual Core, con 4GB de RAM a 1.40 GHz de velocidad . La prueba consistió en realizar a funcionalidades tres pruebas de 25, 50 y 100 hilos cada una, los cuales simulan 25, 50 y 100 accesos de usuarios respectivamente. En la *figura 16* se muestran las pruebas de desempeño realizadas en el navegador web Mozilla Firefox y en el Apache JMeter.

En el navegador web Mozilla Firefox se comprobaron varias funcionalidades. En la *figura 16* se muestra que se obtuvo un resultado satisfactorio.

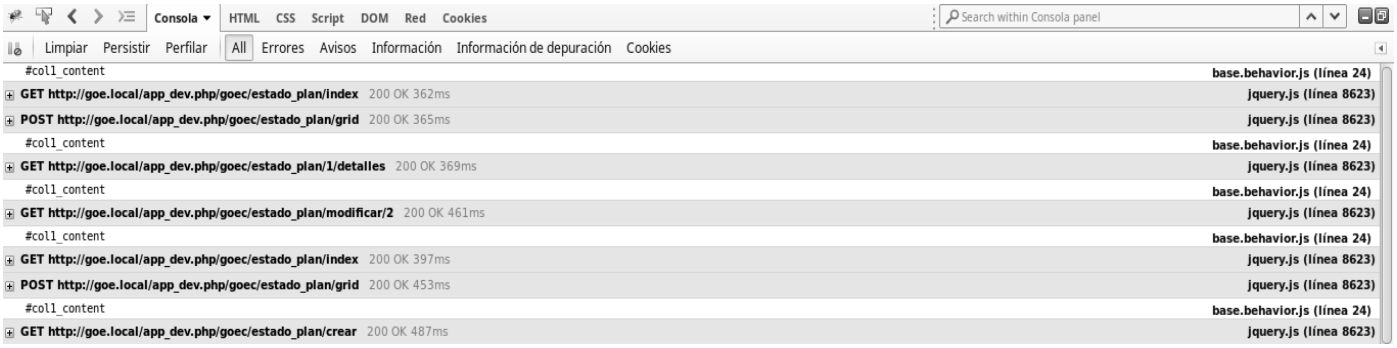


Figura 16: Prueba de desempeño a la funcionalidad Estado de planificación.

En la herramienta Apache JMeter se aplicaron las pruebas a varias funcionalidades, en la *tabla 7* se muestra el resultado alcanzado en una de estas funcionalidades. Para consultar los resultados de las demás pruebas realizadas ver anexo 7.

Tabla 7: Resultados de las pruebas de sistema.

Aplicado a	**Muestras**	**Tiempo de ejecución (ms)**			**Rendimiento**		
		Mín.	**Máx.**	**Media**	**% Error**	**Pet/seg**	**Kb/seg**
Crear Incidencia	25	2935	5487	4517	0.0%	3.9/sec	124.7
	50	2458	10758	8286	0.0%	4.2/sec	134.2
	100	2906	17153	13589	0.0%	4.6/sec	135.7

Columna mínima (Mín.): Indica el mínimo de tiempo de ejecución invertido para una petición con n (columna Muestras) usuarios haciendo peticiones de manera concurrente.

Columna máxima (Máx.): Indica el máximo de tiempo de ejecución invertido para una petición con n (columna Muestras) usuarios haciendo peticiones de manera concurrente.

Media: Representa el tiempo de ejecución promedio de una petición con n usuarios.

Error: Indica la relación entre el total de peticiones y el número de peticiones que originaron errores.

Rendimiento (Pet./seg): Hace referencia al número de peticiones que el servidor puede procesar en un segundo.

Rendimiento (Kb./seg): Hace referencia a la cantidad de datos que el servidor puede procesar en un segundo.

Interpretación de los resultados de las pruebas de rendimiento

Teniendo en cuenta que se realizaron pruebas para un máximo de 100 usuarios concurrentes, se puede observar que en las mismas la ocurrencia de errores se mantiene en 0.0%, lo que significa que todas las peticiones hechas se ejecutan satisfactoriamente. Se obtuvo un rendimiento de 4.6 peticiones por segundo y un tiempo promedio de ejecución de 13 segundos. Atendiendo a la cantidad de peticiones por segundo que se enviaron, la velocidad de respuesta de las peticiones enviadas y las prestaciones de hardware donde se realizaron las pruebas, se considera que los resultados alcanzados son buenos y considerablemente superables si se mejoran las características del entorno de prueba.

- **Pruebas funcionales**

Tienen como principal objetivo comprobar que los sistemas desarrollados cumplen con los requisitos funcionales del software. Los probadores o analistas de pruebas no enfocan su atención en cómo se generan las respuestas del sistema, sino en el funcionamiento de la interfaz del sistema (50).

- **Partición equivalente**

La partición equivalente es una técnica de prueba de caja negra que divide el campo de entrada en clases de datos que tienden a ejercitar determinadas funciones del software. Se dirige a la definición de casos de prueba que descubran clases de errores, reduciendo así el número total de casos de prueba que hay que desarrollar. Un caso de prueba ideal descubre de forma inmediata una clase de errores que, de otro modo, requerirían la ejecución de muchos casos antes de detectar el error genérico (50).En la *tabla 8* se muestra un ejemplo de diseño de caso de prueba.

Tabla 8: Diseño de caso de prueba Crear planificación.

Escenario	Descripción	Respuesta del sistema	Flujo central
EC 1.1 Insertar datos de forma correcta.	Mediante este escenario se inserta en el sistema una planificación.	El sistema actualiza el listado y muestra el mensaje: "La operación se ha realizado satisfactoriamente".	-El usuario una vez autenticado en el Sistema de Gestión del Ciudadano selecciona el Subsistema de Seguridad y protección y luego el componente Guardia Obrera Estudiantil. -El sistema muestra las opciones de menú.

			-El usuario selecciona en la agrupación funcional "Planificación" la funcionalidad "Planificación". -El sistema muestra las planificaciones registradas. -El usuario selecciona en el área de iconos flotantes la opción Crear. -El usuario llena los datos correctamente y presiona el botón Aceptar.
EC 1.2 Insertar elementos repetidos.	Mediante este escenario se introducen datos para insertar una planificación que ya existe en el sistema.	El sistema muestra un mensaje de error indicando: El elemento ya existe.	-El usuario una vez autenticado en el Sistema de Gestión del Ciudadano selecciona el Subsistema de Seguridad y protección y luego el componente Guardia Obrera Estudiantil. -El sistema muestra las opciones de menú. -El usuario selecciona en la agrupación funcional "Planificación" la funcionalidad "Planificación". -El sistema muestra las planificaciones registradas. -El usuario selecciona en el área de iconos flotantes la opción de Crear. -El usuario llena los datos de un elemento que ya esté en el sistema y presiona el botón Aceptar.
EC 1.3 Insertar datos Incompletos.	Mediante este escenario no se introducen todos los	El sistema muestra encima del componente un	-El usuario una vez autenticado en el Sistema de Gestión del Ciudadano selecciona el Subsistema de Seguridad

	datos para crear una planificación.	mensaje en color rojo indicando: Campo requerido.	y protección y luego el componente Guardia Obrera Estudiantil. -El sistema muestra las opciones de menú. -El usuario selecciona en la agrupación funcional "Planificación" la funcionalidad "Planificación". -El sistema muestra las planificaciones registradas. -El usuario selecciona en el área de iconos flotantes la opción de Crear. -El usuario llena los datos de forma incompleta y presiona el botón Aceptar.
EC 1.4 Insertar datos incorrectos.	Mediante este escenario se introducen datos incorrectos para crear una planificación.	El sistema muestra en rojo el mensaje: Entre solo letras, números y espacio o guión bajo entre palabras. El sistema muestra en color rojo encima del componente el mensaje: Entre al menos 2 caracteres. El sistema muestra en color rojo encima del componente el mensaje: No más de 250 caracteres. El sistema muestra	-El usuario una vez autenticado en el Sistema de Gestión del Ciudadano selecciona el Subsistema de Seguridad y protección y luego el componente Guardia Obrera Estudiantil. -El sistema muestra las opciones de menú. -El usuario selecciona en la agrupación funcional "Planificación" la funcionalidad "Planificación". -El sistema muestra las planificaciones registradas. -El usuario selecciona en el área de iconos flotantes la opción de Crear. -El usuario llena los datos de forma incorrecta y presiona el botón Aceptar.

		en rojo encima del componente: Ha excedido el número de caracteres permitidos para una palabra.	
EC 1.5 Cancelar operación.	Mediante este escenario se cancela la operación de crear una planificación.	Si presiona Aceptar cancela la creación y muestra el listado de planificaciones, si presiona Cancelar se mantiene en la misma interfaz del crear.	-El usuario una vez autenticado en el Sistema de Gestión del Ciudadano selecciona el Subsistema de Seguridad y protección y luego el componente Guardia Obrera Estudiantil. -El sistema muestra las opciones de menú. -El usuario selecciona en la agrupación funcional "Planificación" la funcionalidad "Planificación". -El sistema muestra las planificaciones registradas. -El usuario selecciona en el área de iconos flotantes la opción Crear. -El usuario llena o no los datos y presiona el botón Cancelar. -El sistema muestra el mensaje de confirmación ¿Está seguro de realizar la acción?

Resultados de las pruebas funcionales

En la *tabla 9* se muestra la relación de no conformidades por cada iteración efectuada durante la realización de las pruebas funcionales a los requisitos implementados.

Tabla 9: Relación de no conformidades por iteraciones.

Iteraciones	Cantidad de no conformidades	Asociadas
1ra	61	Errores de interfaz, funcionalidades incorrectas o ausentes.
2da	49	Errores de validación, de interfaz de usuario y ortografía.
3ra	0	-

En la *figura 17* se muestra una representación gráfica de la relación de las no conformidades encontradas durante la realización de las pruebas funcionales.

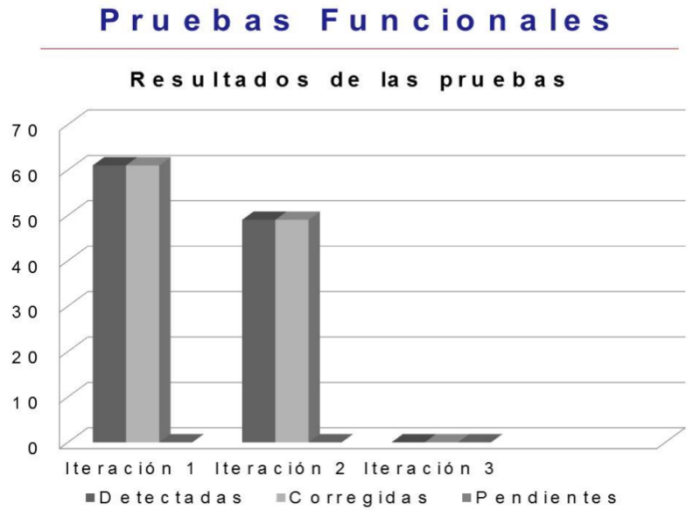


Figura 17: Relación de no conformidades por iteración.

3.6 Conclusiones parciales

Como parte de este capítulo se describieron los estilos de programación y los estándares de codificación empleados en el desarrollo del Componente para la Gestión de la Guardia Obrera Estudiantil; esto permitió mantener la uniformidad y organización en el código para lograr una mejor comprensión y futuro soporte del mismo. Además, se aplicaron un conjunto de pruebas que permitieron evaluar el componente en cuanto a resistencia, rendimiento, funcionalidad e integración mostrando resultados satisfactorios y garantizando la calidad requerida.

Conclusiones generales

Como resultado de la investigación se arribó a las siguientes conclusiones:

- El estudio de los sistemas homólogos permitió identificar un conjunto de funcionalidades que pueden ser implementadas en la nueva solución teniendo en cuenta las necesidades actuales que se presentan.
- La puesta en práctica de la metodología de desarrollo de *software* DAC permitió obtener la documentación asociada al componente.
- Con la implementación del Componente para la Gestión de la Guardia Obrera Estudiantil se logró cumplir satisfactoriamente con los objetivos propuestos al inicio de la investigación, presentando como resultado una solución informática capaz de agilizar y mejorar la gestión del proceso de guardia obrera estudiantil en la UCI.
- La aplicación de pruebas de software permitió encontrar y corregir no conformidades identificadas durante el período de desarrollo, asegurando el cumplimiento de los requisitos de software.

Recomendaciones

Dada la solución propuesta se recomienda para investigaciones futuras:

- Implementar un algoritmo inteligente que permita generar la planificación automática de la guardia obrera estudiantil.
- Diseñar patrones que puedan ser asignados a las personas en el momento de planificar la guardia.
  Un ejemplo de patrón sería:
  Realizar la guardia todos los días primeros de cada mes.
- Implementar las descripciones de requisitos asociadas a la agrupación funcional Reportes.

Referencias bibliográficas

1. EcuRed. *EcuRed.* [En línea] [Citado el: 25 de octubre de 2015.] http://www.ecured.cu/La_Guardia_Pretoriana.

2. Gaceta Oficial de la República de Cuba. [En línea] [Citado el: 18 de octubre de 2015.] http://www.facetaoficial.cu.

3. Blog de Sap. [En línea] [Citado el: 25 de noviembre de 2015.] http://www.blogdesap.com/2011/03/definicion-de-turnos-y-horarios-en.html.

4. Robotics. [En línea] [Citado el: 20 de noviembre de 2015.] http://www.robotics.es/control-horario-personal.

5. EMR software. [En línea] [Citado el: 20 de noviembre de 2015.] http://www.emrsoftware.es/jano-planificacion-hospitalaria.html.

6. aTurnos. [En línea] [Citado el: 20 de noviembre de 2015.] http://www.aturnos.com.

7. *Proceso de Desarrollo de Software,Metodología DAC.* Méndez, Alelí Sánchez. 2013.

8. Mendez, Alelí Sanchez. *Proceso de desarrollo de software.Metodología DAC.* Universidad de las Ciencias Informáticas,Habana : s.n., 2013.

9. Heredia, Herminio Santos. Maestros Del Web. [En línea] [Citado el: 20 de noviembre de 2015.] http://www.maestrosdelweb.com/phpintro.

10. PHP. [En línea] [Citado el: 20 de noviembre de 2015.] http://www.php.net/...

11. Lemay, Laura. *Teach Yourself Web Publishing with HTML 4 in a Week.* USA: Sams Indianapolis, 2012. 1575213362.

12. W3C. [En línea] [Citado el: 25 de noviembre de 2015.] http://www.w3.org/TR/2006/REC-xml11-20060816.

13. Escofet, Carme Martín. *El lenguaje SQL.* 2010.

14. Gauchat, Juan Diego. *El gran libro de HTML5, CSS3 y Javascript.* Barcelona : Marcombo, 2012.

15. Visual Paradigm. [En línea] [Citado el: 25 de noviembre de 2015.] http://www.visual-paradigm.com.

16. Baeza, Pablo Nicolás. Visual Paradigm DB Visual Architect SQL. [En línea] [Citado el: 25 de noviembre de 2015.] http://www.docstoc.com/docs/96492173/Visual.

17. GNU Linux Android,software libre,tecnología y mucho más. [En línea] 2010. [Citado el: 10 de octubre de 2015.] www.nosolounix.com.

18. Guía-Ubuntu. Guía Documentada para Ubuntu. [En línea] [Citado el: 27 de noviembre de 2015.] http://www.guia-ubuntu.com/index.php/PgAdmin_III.

19. Richard, Matthew Neily. *Beginning Databases with PostgreSQL.* 2005.

20. POSTGRESQL. *Sobre PostgreSQL.* [En línea] [Citado el: 25 de noviembre de 2015.] http://www.postgresql.org.es/sobre_postgresql.

21. The Apache Software Fundation. [En línea] [Citado el: 27 de noviembre de 2015.] http://www.apache.org.

22. Pencil Project. [En línea] [Citado el: 27 de noviembre de 2015.] http://pencil.evolus.vn.

23. Subversion. *The Apache Software Fundation.* [En línea] [Citado el: 29 de noviembre de 2015.] http://subversion.apache.org.

24. NetBeans. [En línea] [Citado el: 28 de noviembre de 2015.] https://netbeans.org/.

25. Symfony. [En línea] [Citado el: 27 de noviembre de 2015.] http://symfony.com.

26. Álvarez, Miguel Angel. Manual de JQuery. [En línea] [Citado el: 29 de noviembre de 2015.] http://www.desarrolloweb.com/manuales/manual-jquery.html.

27. Bizagi. [En línea] [Citado el: 10 de enero de 2016.] http://help.bizagi.com/bpmsuite/es/index.html?definir_reglas_de_negocio.htm.

28. Sommerville, Ian. *Ingeniería de software.* s.l.: Prentice-Hall.6ta Edición, 2002. 970-26-0206-8.

29. Sampalo de la Torre, Maria de los Angeles, y otros, y otros. *Informática Volumen III.* ALCALA DE GUADAIRA (Sevilla): EDITORIAL MAC, S.L, 2003. 84-665-1151-2.

30. Romo, Miguel Martin. *Ingeniería de software.* Madrid: PEARSON EDUCASION,SA:7ma edición, 2005. 84-7829-074-5.

31. Ecured. [En línea] [Citado el: 5 de enero de 2016.] http://www.ecured.cu/index.php/Cliente-Servidor.

32. emaze. [En línea] [Citado el: 20 de 5 de 2016.] https://www.emaze.com/@ACIRRQRI/ARQUITECTURA-cliente-servidor.

33. Lema Mouzo, Juan y Makedonsky, Mariano. *Flash: desarrollo profesional.* Buenos Aires : fox Andina SA, 2011. 978-987-1857-00-5.

34. Undercode. [En línea] [Citado el: 20 de 5 de 2016.] https://underc0de.org/foro/java/modelo-vista-controlador/.

35. WikiSalud. [En línea] [Citado el: 20 de 5 de 2016.] http://wiki.salud.gob.sv/wiki/Desarrollo_web_Symfony2.

36. Ecured. [En línea] [Citado el: 5 de enero de 2016.] http://www.ecured.cu/patrones_de_diseno_y_arquitectura.

37. Entorno Virtual de Aprendizaje (EVA).Patrones de diseño de base de datos. [En línea] [Citado el: 5 de enero de 2016.] http://eva.uci.cu/file.php/180/2._Clases/Tema_1/Materiales_basicos/4.Patrones_de_ diseno_de_BD.pdf.

38. CLEMENTS, P. *"A survey of Architecture Description Languages". Proceedings of the International WorkShop on Software Specification an Design.* 1996.

39. Patrones de diseño de base de datos. Eva. [En línea] [Citado el: 5 de enero de 2016.] http://eva.uci.cu/file.php/180/2._Clases/Tema_1/Materiales_basicos/4.Patrones_de_diseno_de_BD.pdf.

40. Bello Pena, David. *Modelización de datos. Un enfoque práctico.* España: Lulu.com, 2009. 978-1-4092-1640-7.

41. Stallings, William. *Comunicaciones y redes de computadoras.* s.l. : 6ta edición.

42. Alvarez, Miguel Angel. Desarrollo web.com. [En línea] [Citado el: 3 de febrero de 2016.] http://www.desarrolloweb.com.

43. Garcia, Victor. Programación en Java. [En línea] [Citado el: 4 de febrero de 2016.] http://programarjava.wordpress.com/2011/12/13/programacion-orientada-aeventos/..

44. Arias Calleja, Manuel. *Estándares de codificación.*

45. Pressman, Roger S. *Ingeniería de Software. Un enfoque práctico.Capítulo 10.Ingeniería de sistema. Validación de requisitos, 174.* s.l.: Quinta edición.

46. Marco de Desarrollo de la Junta de Andalucía. [En línea] [Citado el: 10 de febrero de 2016.] http://www.juntadeandalucia.es/servicios/madeja/sites/default/files/historico/1.4.0/contenido-recurso-419.html.

47. Pressman, Roger. *Ingeniería de Software, un enfoque práctico.* New York: McGraw-Hill Companies : Séptima Edición, 2007. 978-0-07-337597.

48. *Ingeniería de Software. Capítulo 18.Estrategia de pruebas de software. Prueba del sistema. Pag 317.*

49. *Ingeniería de Software. Capítulo 18.Estrategia de prueba de software .Pruebas de resistencia. Pág 318.*

50. PRESSMAN, Roger S. *Ingeniería de Software. Un enfoque práctico.* Madrid: s.n, Quinta edición, 2001.

51. Real Academia Española. [En línea] [Citado el: 11 de noviembre de 2015.] http://dle.rae.es/?id=TJyphVT.

52. Definiciones ABC. *Definiciones ABC.* [En línea] [Citado el: 11 de noviembre de 2015.] http://www.definicionabc.com/general/planificacion.php.

53. Española, Diccionario de la Real Academia. Diccionario de la Real Academia Española. [En línea] [Citado el: 11 de noviembre de 2015.] http://dle.rae.es/?id=AeYZ09V.

54. Anzil, Federico. Zona Económica. [En línea] [Citado el: 11 de noviembre de 2015.] http://www.zonaeconomica.com/control..

55. Definición de. [En línea] [Citado el: 11 de noviembre de 2015.] http://definicion.de/gestion.

56. Szyperski, Clemens. *Component Software Component Software Beyond Object – Oriented Programming.*

57. Española, Real Academia. Real Academia Española. [En línea] [Citado el: 11 de noviembre de 2015.] http://dle.rae.es/?id=JjpuE9J.

58. Ruz, Fidel Castro. Gaceta Oficial de la República de Cuba. [En línea] 14 de junio de 1998. [Citado el: 15 de noviembre de 2015.] http://www.gacetaoficial.cu/.

59. [En línea] [Citado el: 18 de octubre de 2015.] http://www.blogdesap.com/2011/03/definicion-de-turnos-y-horarios-en.html.

60. S.Pressman, Roger. Modelado de diseño de aplicaciones web. *INGENIERÍA DE SOFTWARE. Un enfoque práctico.* s.l. : Mc Graw Hill, 2002.

61. Real Academia Española. [En línea] [Citado el: 21 de noviembre de 2015.] http://dle.rae.es/?id=TIvEXgq.

Bibliografía consultada

1. González Castellanos, Roberto A. 2003, Diciembre. Metodología de la investigación científica. Parte I.
2. González Castellanos, Roberto A. 2003, Diciembre. Metodología de la investigación científica. Parte II.
3. PARADIGM, V. UML tool, business process modeler and database designer for *software* development team. [En línea] 2015. http://www.visual-paradigm.com.
4. GAMMA, E., HELM, R. y JOHNSON, R. y VLISSIDES, J. Patrones de diseño. 2000.
5. Cooper, James W. The Design Patterns. Java Companion. octubre 2, 1998.
6. Pressman, Roger S. *Pressman_Cap_10_Diseno_Arquitectonico_Parte_1.* Sexta Edición.
7. Apache JMeter -Apache JMeter™. 2013. Apache JMeter -Apache JMeter™. *Apache JMeter - Apache JMeter™.* [En línea] 2013. http://jmeter.apache.org/index.html.
8. Home -Pencil Project. 2012. Home -Pencil Project. *Home -Pencil Project.* [En línea] 2012. http://pencil.evolus.vn/.
9. Calleja., Manuel Arias. *Estándares de codificación.*
10. NetBeans. [En línea] [Citado el: 28 de noviembre de 2015.] https://netbeans.org/.
11. Symfony. [En línea] [Citado el: 27 de noviembre de 2015.] http://symfony.com.
12. Álvarez, Miguel Angel. Manual de JQuery. [En línea] [Citado el: 29 de noviembre de 2015.] http://www.desarrolloweb.com/manuales/manual-jquery.html.
13. Ruz, Fidel Castro. Gaceta Oficial de la República de Cuba. [En línea] 14 de junio de 1998. [Citado el: 15 de noviembre de 2015.] http://www.gacetaoficial.cu/.
14. Robotics. [En línea] [Citado el: 20 de noviembre de 2015.] http://www.robotics.es/control-horario-personal.
15. EMR software. [En línea] [Citado el: 20 de noviembre de 2015.] http://www.emrsoftware.es/jano-planificacion-hospitalaria.html.
16. aTurnos. [En línea] [Citado el: 20 de noviembre de 2015.] http://www.aturnos.com.

17. Proceso de Desarrollo de Software, Metodología DAC. Méndez, Alelí Sánchez. 2013.

18. Ingeniería de Software. Capítulo 18.Estrategia de pruebas de software. Prueba del sistema.Pag 317.

19. Ingeniería de Software. Capítulo 18.Estrategia de prueba de software. Pruebas de resistencia.Pág 318.

20. Clemens Szyperski, "Component Software Component Software Beyond Object – Oriented Programming". P 20 -22. 1998.