Universidad de las Ciencias Informáticas

Facultad 4

SweetSalt: herramienta para la planificación, control y seguimiento de la administración centralizada de computadoras

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

Autores:

- *Manuel Román Sosa*
- *Manuel García Bouza*

Tutores:

- *Ing. Michel Miranda Cairo*
- *Ing. Rosalina Ibarra González*

Declaración de autoría

Declaramos ser los autores del presente trabajo de diploma y otorgamos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo. Para que así conste firmamos la presente a los ____ días del mes de _____ del año _____.

_____

Firma del autor
Manuel García Bouza

_____

Firma del autor
Manuel Román Sosa

_____

Firma del tutor
Ing. Michel Miranda Cairo

_____

Firma del tutor
Ing. Rosalina Ibarra González

Resumen

Este trabajo documenta la creación de la herramienta de administración centralizada de computadoras SweetSalt, la cual implementa las mejores prácticas de su homóloga SaltStack y propone una interfaz gráfica capaz de mediar entre el usuario y las diversas tareas de configuración remota a través de comandos por consola. Esta provee también, simultaneidad en la configuración remota de las estaciones de trabajo de la Facultad 4 y reduce el desplazamiento de los especialistas del Departamento de tecnología de dicha facultad, así como el tiempo empleado y la aparición de errores humanos en el proceso.

Tales beneficios se obtuvieron después de construir la fundamentación teórica de la solución, sustentada en los conceptos y definiciones relacionadas con el tema en cuestión, así como en el estudio de diversas herramientas homólogas. De igual manera se estudiaron y emplearon tecnologías como el Lenguaje Unificado de Modelado (UML), la herramienta de ingeniería de software asistida por computadoras Visual Paradigm, los lenguajes de programación PHP, CSS3, HTML y JavaScript, así como los marcos de trabajo Symfony2 y Bootstrap, Apache como servidor web y NetBeans como entorno integrado de desarrollo.

Para la base de datos se empleó MySQL como sistema gestor y todo la construcción de la propuesta estuvo regida por la metodología Extreme Programming XP, lo cual indica fundamentalmente una orientación a la implementación empleando el estándar de codificación de Symfony2 con el estilo de escritura lowerCamelCase. Para validar el producto de la investigación se aplicaron métricas al diseño y pruebas al software.

Palabras claves: SweetSalt, SaltStack, administración centralizada de computadoras, configuración remota de estaciones de trabajo.

Contenido

Índice de ilustraciones

Índice de tablas

Introducción

La administración de recursos ha sido un área de constante evolución e innovación para el hombre; así como su especialización en las diferentes esferas donde el capital humano y financiero, los activos fijos tangibles, las soluciones de software y el control de los diferentes productos comercializables son de vital importancia.

Dado que no resulta sencillo administrar todos los recursos anteriormente mencionados, algunas nuevas herramientas y soluciones de diversos tipos han surgido para apoyar al hombre en tales tareas. Entre ellas, se encuentran las aplicaciones de software, que han ido reclamando su espacio en la sociedad debido al desarrollo científico-tecnológico y a la facilidad de cómputo, organización, extracción de reportes, transmisión de datos, monitorización de transferencias, manejo remoto de recursos, control y supervisión de procedimientos, entre otras actividades que pueden realizarse si se emplean productos informáticos.

Ante tales beneficios, Cuba con el propósito de proveer nuevas vías de desarrollo a su pueblo, ha visto en la informatización de la sociedad una forma progresiva y favorable para propiciar el crecimiento económico-social. Como materialización de tal estrategia, fue creada la Universidad de las Ciencias Informáticas con el propósito de vincular prácticamente el precepto de estudio-trabajo, donde la formación y la producción de software estuvieran estrechamente vinculadas. Una de las áreas académicas de dicha universidad es la Facultad 4, que cuenta con más de 150 computadoras distribuidas en oficinas, departamentos docentes, aulas, salones de conferencias y laboratorios docentes y de producción. Lo que implica –evidentemente- que se deba llevar un control minucioso de todos esos recursos para evitar desperfectos de funcionamiento y errores de transferencia de datos. Todas esas máquinas están conectadas en una red de área local, requiriendo la supervisión personal de su estado en las locaciones anteriormente expuestas.

La gestión de tantos equipos constituye un reto para el grupo de trabajadores del Departamento de Tecnología de dicha facultad, debido a la necesidad de realizar un conjunto de tareas de configuración de forma manual a través de comandos en la terminal de Linux. Los especialistas y técnicos de este departamento, tenían que desplazarse constantemente por las áreas elevando su carga de trabajo, el

tiempo empleado en la configuración de las computadoras o en los controles rutinarios, lo que podía incurrir en la introducción de errores humanos en el proceso.

Con el objetivo de disminuir los problemas previamente descritos, se desplegó una herramienta de administración de recursos centralizada conocida como SaltStack. La misma consiste en un software de código abierto para la gestión remota de computadoras. Con el uso de esta herramienta, se posibilitó la comunicación con cualquier servidor y la administración de las computadoras en un menor tiempo, sin necesidad de desplazarse por cada una de las estaciones de trabajo. Sin embargo, aún persisten dificultades en cuanto al uso de los numerosos comandos de la terminal de Linux y no provee una interfaz sencilla y rápida para la ejecución de los diferentes comandos de manera múltiple, es decir aún no es posible la realización de una misma tarea a un grupo de computadoras simultáneamente.

Por lo anteriormente expuesto se plantea el siguiente **problema a resolver**: ¿Cómo disminuir el tiempo de ejecución del proceso de administración centralizada de las computadoras en las estaciones de trabajo de la Facultad 4 sobre la consola de SaltStack?

La investigación enmarca su **objeto de estudio** en la administración centralizada de las computadoras, tomando como **campo de acción** el proceso de administración centralizada de las computadoras sobre la consola de SaltStack.

Para dar solución al problema planteado se propone como **objetivo general**: desarrollar una solución que disminuya el tiempo de ejecución del proceso de administración centralizada de las computadoras en las estaciones de trabajo de la Facultad 4 sobre la consola de SaltStack.

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

- Estudiar los aspectos teóricos fundamentales que sustentan este trabajo mediante consultas, extracción y recopilación de información relevante, para establecer precedentes, aspectos medulares y contrapartes de la investigación.
- Definir una solución de software siguiendo las pautas de una metodología de desarrollo determinada, para establecer el punto de partida desde la propuesta de la investigación.
- Diseñar la propuesta de solución para ilustrar las relaciones, dependencias y responsabilidades de cada concepto del negocio.

- Implementar la solución de software con el uso de las tecnologías seleccionadas, para concretar la propuesta como aporte práctico de la investigación.
- Realizar pruebas al software para verificar y validar la propuesta de solución de la presente investigación.

Con el fin de corroborar el cumplimiento del objetivo general de este trabajo, se tienen las siguientes **preguntas de la investigación**:

¿Cuáles elementos de la problemática resultan críticos para los umbrales actuales de tiempo en la administración centralizada de computadoras a través de la consola de SaltStack?

¿Qué características funcionales son determinantes para disminuir el tiempo de ejecución del proceso de administración centralizada de computadoras?

En cuanto al desarrollo documental de este trabajo, se utilizan los siguientes métodos científicos para conducir y registrar los resultados de la investigación:

Métodos empíricos:

- **Entrevista**: para obtener los problemas y soluciones presentes en el departamento de tecnología de la facultad 4.
- **Observación:** la observación fue otro de los métodos utilizados en el desarrollo de la investigación, específicamente en la observación del comportamiento de la consola de SaltStack.

Métodos teóricos:

- **Analítico–sintético:** para el estudio de las fuentes bibliográficas y materiales relacionados con el tema, identificando elementos esenciales para darle solución al problema planteado.
- **Análisis histórico–lógico:** permite analizar la trayectoria completa acerca del control en la administración centralizada de computadoras a través de herramientas informáticas, así como el estudio histórico de las mismas.
- **Inductivo-deductivo:** posibilita determinar las características de los sistemas encargados de la administración centralizada de computadoras, que puedan ser aplicables a la investigación en curso y además permitirá la descripción e implementación de las funcionalidades identificadas.

Finalmente, para lograr una descripción cabal del proceso de creación de una solución para el problema planteado y la subsecuente investigación inherente al mismo, se estructura este documento en 3 capítulos agrupando los elementos imprescindibles para la concepción, el diseño, la implementación y validación de la propuesta. A continuación se describen dichos capítulos:

Capítulo I: Fundamentación teórica

En este capítulo se hace referencia a los elementos teóricos que sustentan la investigación, mediante el estudio de la evolución y situación actual de las herramientas cuyo objetivo se enfoca en la administración centralizada de computadoras. Para ello, se describen las soluciones similares y se realizan comparaciones, de manera que se pueda conceptualizar una propuesta de solución empleando las mejores características comunes de sus antecesores e incorporar nuevas funcionalidades que deban ser parte del aporte práctico de este trabajo. Se presentan además, los lenguajes de programación y modelado, así como la metodología de desarrollo software y demás tecnologías que se utilizarán fundamentando su selección con el estudio realizado.

Capítulo II: Descripción de la propuesta de solución

Luego de identificar el precedente teórico y de definir la base tecnológica, se describen las características del sistema a desarrollar, listándose por tanto, las funcionalidades y restricciones con las que este debe cumplir. De igual modo se asocian los conceptos del negocio a través de las entidades de la base de datos, así como las relaciones, responsabilidades y colaboraciones entre clases y demás elementos que responden al uso de la metodología de desarrollo seleccionada en el capítulo anterior.

Capítulo III: Implementación y pruebas

Al haber concebido y diseñado la propuesta de solución se procede a su implementación y comprobación. Para ello, en el presente capítulo se documenta el estándar de codificación empleado, la representación de las clases del sistema por capas de acuerdo al modelo definido en el capítulo anterior y toda la instrumentación de pruebas de software para la verificación y validación de la solución.

Capítulo 1: Fundamentación teórica

Introducción

El primer paso para comenzar una investigación es plantearse las interrogantes que cuestionan algo de la sociedad o del ámbito científico. Para hallar una solución es preciso realizar la fundamentación teórica basada en los conceptos asociados al dominio del problema, definición, características y ventajas de los mismos. En este caso, las herramientas para la administración centralizada y específicamente, las herramientas para la administración centralizada de computadoras, tanto fuera como dentro del ámbito nacional resultan vitales para este trabajo.

1.1. Conceptos y definiciones relacionados con la administración centralizada

Antes de hacer referencia al concepto de administración centralizada es necesario precisar que la gestión aplicada a la administración de computadoras, obliga a que la misma cumpla con varias funciones específicas como: controlar, estandarizar, proteger, auditar y ejecutar tareas.

1.1.1. Administración

Existen dos formas con marcadas diferencias para definir este concepto:

La administración como disciplina: *es el conjunto de principios, ideas y conceptos que deben tenerse en cuenta para guiar a un grupo.* (Stoner, 1992)

La administración como técnica: *como técnica, la administración constituye un conjunto de procedimientos validados por el conocimiento y por la experiencia, y de aplicación general o particular. Estos procedimientos y recursos que se moldean sobre la base de la profesión de administrador, requieren de un ejercicio constante para adquirir mayor pericia y habilidad al utilizarlos en la solución de problemas prácticos.* (Stoner, 1992)

De acuerdo a una definición más general dada por el mismo autor James A. Stoner: *"la administración es el proceso de planear, organizar, liderar y controlar los esfuerzos de los miembros de la organización, y el empleo de todos los demás recursos organizacionales para lograr objetivos establecidos."*(Stoner, 1992).

Por su parte, la planeación efectiva, la toma de decisiones y el control dependen del manejo efectivo de la información a través de los diferentes sistemas. De manera que los que están basados en la computación facilitan obtener, almacenar, organizar y distribuir información útil para los administradores.

1.1.2. Control

Se define el **control** como un mecanismo preventivo y correctivo adoptado por la administración de una organización o entidad. El control permite la oportuna detección y corrección de desviaciones, ineficiencias o incongruencias en el curso de la instrumentación, ejecución y evaluación de las acciones. Se hace además con el propósito de procurar el cumplimiento de normativas, estrategias y políticas.

Stoner plantea en la 5ta edición del libro de *Administración* que el control administrativo se trata de un proceso para garantizar que las actividades reales se ajusten a las actividades planeadas. No obstante, el control es un concepto más general, uno que también ayuda a los administradores a realizar el seguimiento de la eficacia de su planeación, su organización y su dirección y a tomar medidas correctivas cuando se necesitan. Por tanto, aunque el control sea la cuarta etapa del proceso administrativo, sin duda, por orden de importancia, no es la última. (Stoner, 1992)

Para la presente investigación, *el control* es el proceso o función administrativa que permite verificar, constatar y medir, si la actividad, unidad, elemento o sistema seleccionado está produciendo o no los resultados que se esperan, con el objetivo de asegurar que las actividades reales correspondan a las actividades proyectadas.

Es evidente que un motivo por el cual se requiere tener control es vigilar el avance y corregir errores. Pero no solo eso, el control también ayuda a los administradores a hacer el seguimiento de los cambios ambientales y las repercusiones que estos producen en el avance de la organización. Algunos de los cambios más apremiantes del entorno son el cambio en la naturaleza de la competencia, la necesidad de acelerar el ciclo oferta-demanda, la importancia de "agregar valor" a los productos y los servicios como manera para crear demanda por parte de los clientes, los cambios de las culturas organizacionales y de los trabajadores, así como la necesidad, cada vez mayor, de delegar y trabajar en equipo en las organizaciones.

1.1.3. Centralización

La computarización ha tenido un gran impacto en las organizaciones debido al aumento de la capacidad de gestión y procesamiento de datos con mayor precisión y velocidad. Sin embargo, la computarización no cambia las estructuras existentes en una organización, sencillamente permite realizar ciertas actividades aunque para el usuario final represente una vía hacia la descentralización. (Stoner, 1992)

En realidad, la disposición sobre los recursos, su gestión y control, se mantiene centrado en un reducido número de personas. Por lo que a pesar de lo anteriormente expuesto, la *centralización* se entiende como la tendencia a concentrar la autoridad para la toma de decisiones, dándole a este grupo específico de administradores, gerentes o directores -según corresponda- la iniciativa y poder de decisión, así como la potestad de ejercer o disponer sobre las funciones, recursos y medios.

En el caso de la administración de computadoras, la centralización se entiende como la concentración o agrupación de tareas de configuración en una única consola capaz de administrar desde una estación de trabajo, todas las demás computadoras conectadas en una red de área local.

1.1.4. Administración centralizada de computadoras

La administración centralizada de computadoras para la presente investigación se entiende como el proceso de controlar la información, configuración y funcionamiento de las estaciones de trabajo de forma central, simultánea y remota, con el objetivo de disminuir el costo y el riesgo en dicho proceso.

1.1.5. Eficiencia

Algunos expertos como **Koontz** y **Weihrich** aseguran que la eficiencia consiste en el logro de aquellas metas que se ha propuesto una empresa utilizando para ello la menor cantidad posible de recursos. (Koontz and Weihrich, 2011)

Por su parte, **Robbins** y **Coulter**, dicen que es obtener resultados de una magnitud importante invirtiendo la mínima cantidad posible en ella. (Robbins and Coulter, 2013)

Eficiencia: de acuerdo al concepto de administración centralizada de computadoras, la eficiencia se entiende como la gestión simultánea de todos los recursos conectados en red, requiriendo la realización de la menor cantidad de tareas de configuración y control posibles.

1.2. Herramientas de administración centralizada

Una solución de administración centralizada permite que los administradores configuren, gestionen y controlen en forma remota y automatizada todos los equipos de la red corporativa desde una ubicación central.

A continuación se describen las herramientas estudiadas en este trabajo, para trazar el precedente necesario para valorar objetivamente a SaltStack en cuanto a sus beneficios y carencias.

1.2.1. Comparación de las herramientas de administración centralizada.

Chef

Chef fue originalmente lanzado en 2009, y fue desarrollado con Ruby usando un modelo master-agente. En adición al servidor master, la instalación de Chef requiere una estación de trabajo para que controle las funciones del master. Desde esa misma estación de trabajo pueden ser instalados los agentes, donde los nodos administrados deberán autenticarse con el master a través de certificados. Pero en ese periodo, los agentes deben configurarse para registrarse periódicamente con el master, pues la gestión de cambios instantánea del master a los agentes es realmente imposible. Además el software puede ser ejecutado en modo cliente-servidor o en modo individual denominado Chef-solo.

Chef es un producto viejo, por lo que su documentación está bien organizada y es abundante, sin embargo no es muy popular por la complejidad de su administración y aprendizaje. Este ofrece soporte para Linux, Unix y Windows. Su interfaz gráfica basada en un buscador es bastante funcional, sin embargo carece de funcionalidades como gestión de reportes y opciones avanzadas de configuración. Realmente, los principales clientes de Chef son las corporaciones debido a su madurez tecnológica y estabilidad, pero los clientes individuales apuestan por opciones más actualizadas y con mayor número de opciones y funcionalidades.

Ansible

La herramienta Ansible de gestión de la configuración es bastante diferente de Chef y Puppet, y semejante a SaltStack. Este fue liberado a inicios de 2012 y su código se desarrolló en Python, requiriendo solamente las librearías de este lenguaje para ser configurado en casi todos los servidores de Linux. Entre sus principales beneficios están su ligereza, facilidad de uso relativa y la velocidad de

despliegue en comparación con otras herramientas. Es por ello que no es preciso conocer todos los comandos del paquete de Ansible-Ruby en los módulos de YAML mientras el lenguaje predeterminado que se emplee pueda mostrar los módulos de JSON. Además gestiona toda la comunicación master-agente a través de comandos SSH (Secure Shell) estándares o del módulo Paramiko el cual provee una interfaz de Python para SSH2.

En cuanto a la documentación de Ansible, se puede decir que este es su punto débil en relación a su reciente desarrollo y actualización. Sin embargo, dicha deficiencia es suplida con su facilidad de uso debido al fácil aprendizaje de administración que requiere. Actualmente está disponible solo para Linux y Unix, con una interfaz gráfica aun desprovista de todas las opciones que los clientes desean. Su arquitectura ligera sin agentes, le permite tener una comunicación y despliegue más rápidos aunque no es muy flexible debido a la ausencia de agentes. (*ScriptRock,* 2014)

Puppet Enterprise

Puppet es la herramienta de gestión de la configuración más completa en términos de funcionalidades disponibles, módulos e interfaces de usuario. Esta herramienta representa una nueva forma de dirección de centros de datos, acoplándose con cualquier sistema operativo y ofreciendo opciones robustas para los principales sistemas operativos del mercado. La configuración inicial es relativamente simple, requiriendo la instalación de un servidor master y los agentes de tipo cliente en cada sistema que va a administrarse. De ahí que con el CLI sea sencillo descargar módulos e instalarlos mediante los comandos Puppet. Entonces, es necesario realizar cambios a los archivos de configuración para ajustar el módulo a la tarea requerida. Por su parte los clientes que deberían recibir las instrucciones tendrán que realizar lo antes descrito cuando se registren con el master.

Tiene además, módulos que pueden proveer y configurar servidores en la nube e instancias de servidores virtuales. Todos los módulos y configuraciones son construidos con un lenguaje Puppet específico, basado en Ruby o con el propio lenguaje Ruby, por lo que para su manejo se requerirá amplia experiencia en programación y habilidades de administración. (Philips, 2013)

DameWare Remote Support

DameWare Remote Support permite administrar servidores y computadoras portátiles de forma remota para asistir rápidamente a los usuarios finales. Esta herramienta incluye además control de escritorio a

remoto y la capacidad de completar de forma remota tareas de administración de Windows directamente desde su consola. Esta es compatible con Mac OS X, Linux y Windows, permitiendo administrar entornos de Windows distribuido. En adición, permite administrar y actualizar el directorio activo de Microsoft.
Dicha herramienta exporta de forma remota información desde las computadoras con Windows, ya sea desde unidades de disco, impresoras, servicios o recursos compartidos de red en los formatos comunes disponibles que incluyen XML y CSV. Entre sus principales características se encuentran:

- Da soporte a miles de usuarios finales sin dejar su escritorio.
- Ahorra tiempo al completar de forma remota tareas de administración de Windows.
- Resuelve problemas rápidamente usando el Control remoto.
- Los gastos giran en torno a la administración de las estaciones de trabajo y no en función de las computadoras gestionadas (DameWare, 2015).

Desktop Central

Desktop Central (DC) ofrece amplias prestaciones para la administración, estandarización y aseguramiento de los puestos de trabajo y servidores de una empresa. Esta permite automatizar las tareas de gestión desde una consola centralizada, ahorrando tiempo y recursos, y permitiendo la optimización de estos últimos. Entre las muchas funcionalidades que ofrece DC destacan la instalación de software, parches y service-packs, la estandarización del puesto de trabajo aplicando configuraciones comunes, las políticas de seguridad y control de dispositivos USB, las políticas de ahorro energético, la configuración de aspectos de Windows incluyendo opciones de Internet Explorer, Outlook y Office y la auditoría con un completo inventario de hardware y software, detección de aplicaciones prohibidas y control de utilización de las licencias adquiridas.
Con su arquitectura de red neutral, permite administrar dispositivos en diferentes entornos de red como Directorio Activo, Grupos de Trabajo y Novell eDirectory ahorrando considerablemente tanto en costo como en recursos. Los dispositivos administrados pueden estar en la LAN[1] o distribuidos a través de una WAN[2], incluso en múltiples dominios o usuarios móviles. El módulo de Gestión de Dispositivos Móviles (MDM) permite configurar y asegurar los teléfonos inteligentes y tabletas iOS y Android desde una consola

[1] Local Area Network: Red de área local.
[2] Wide Area Network: Red de área amplia.

central, permitiendo aplicar criterios diferenciados según sean los dispositivos de propiedad corporativa o personal.
DC reduce los gastos de mantenimiento y soporte de los puestos de trabajo y dispositivos móviles y proporciona un rápido retorno de la inversión mediante una completa funcionalidad a un precio realmente asequible (Desktop Central, 2012).

TeamViewer

TeamViewer es una solución funcional para distintos escenarios como: mantenimiento remoto, reuniones, videoconferencias, presentaciones y acceso a ordenadores o servidores remotos. Una herramienta de software para tareas de soporte, administración, ventas, trabajo en equipo, oficina en casa y formación en tiempo real. Compatible con Windows, Mac, Linux, iOS, Android y BlackBerry OS (RIM[3]), incluidas las conexiones entre plataformas. Además funciona sin configuración incluso a través de firewalls y servidores proxy. Permite escuchar y ver de forma remota audio y video en el sistema del ordenador al que está conectado, así como la grabación de sesiones de control remoto o reuniones (incluso con imágenes de cámara web) con audio y voz sobre IP, incluida la conversión a formato de vídeo AVI. Asimismo la grabación de audio de alta definición para las transmisiones de voz sobre IP con reducción de ruido y cancelación de eco automáticos.
Esta herramienta provee una fácil gestión de los dispositivos propios sin necesidad de contraseña, así como provee funciones de mensajería instantánea en la lista de ordenadores y contactos, incluidos chat en grupo y mensajería sin conexión. Finalmente ofrece la posibilidad de habilitar cualquier computadora que esté en la red de área local a través de Wake-on-LAN por el enrutador (TeamViewer, 2015)

SaltStack

Herramientas como SaltStack, permiten publicar comandos por consola para múltiples computadoras a la vez e instalar y configurar el software también. Además se le considera la plataforma más rápida y escalable para sistemas de administración remota, siendo esta última, al igual que la gestión de la configuración, sus características esenciales. (SaltStack Inc., 2014)

La columna vertebral de esta herramienta es el motor de administración remota, lo que crea una alta velocidad y seguridad en la red de comunicación bi-direccional para grupos de sistemas. Esta se

[3] Research in Motion: antiguo nombre del OS de Blackberry.

distribuye bajo licencia GNU y es compatible con distribuciones de Linux. Sin embargo carece de una interfaz visual lo cual establece su uso y ejecución a través de una consola de comandos.

Entre los beneficios más señalados en la bibliografía consultada se tienen:

Velocidad:

- Automatización en tiempo real.
- Administración remota paralela en milisegundos.
- Despliegue continuo de código.
- Control predictivo y monitoreo instantáneo de infraestructura.
- Rápida configuración y predicción del servidor.

Escalabilidad:

- Construida para grandes infraestructuras.
- Compatible para soporte de cualquier tipo de computación en la nube, infraestructura, sistema operativo, virtualización, aplicación de software, contenedor y código.
- Desde pequeña a gran escala para implementaciones ligeras sin agentes o con agentes desde la arquitectura Master-Minion para escalabilidad y velocidad extremas.

Flexibilidad:

- Comprende un sin fin de casos de uso para empresas diversas o específicas.
- Puede orientarse a una arquitectura regida por agentes o ligera (sin agentes).
- Administración de la configuración declarativa o impositiva.
- No solo permite la gestión de la configuración sino también orquestación, predicción, administración remota, automatización y control.

Eficiencia:

- Bajo costo administrativo y operacional.
- Una única plataforma interna.
- La configuración y automatización más eficientes de conjunto.
- Administración sin pago de licencia para requisitos, código o lenguajes.
- Sin restricciones de especialización para uso, cualquier usuario puede administrar la plataforma.
- Implementación y mantenimiento fáciles y rápidos. (*SaltStack Inc.,* 2015)

Herramientas	Beneficios	Desventajas
Chef	• Solución más Madura. • Mayor documentación de la herramienta. • Mayor comunidad, con una colección amplia de módulos y adaptaciones de configuración. • Soporte para Linux, Unix y Windows.	• Lenguaje de programación Ruby sin compatibilidad de CLI con otros lenguajes. • Depende de JSON el cual no es tan fácil como YAML. • No es fácil de aprender y desplegar. • Pobre interfaz gráfica + consola.
Ansible	• Herramienta rápida, con despliegue y comunicación sin agentes. • CLI soporta casi todos los lenguajes de programación. • Usa Python, el cual es omnipresente en todas las distribuciones de Linux. • Excelente seguridad usando SSH /SSH2.	• Aún muy reciente, sin suficiente uso y pruebas de estabilidad. • Desarrollo insuficiente de su interfaz gráfica. • No es compatible con Windows/ Mac OS.
Puppet	• Amplia disponibilidad de funciones, módulos e interfaces de usuario. • Soporte para todos los Linux, Unix, Windows. • Módulos de configuración de servidores en la nube e instancias de servidores virtuales.	• Lenguaje de programación Ruby sin compatibilidad de CLI con otros lenguajes. • Se requiere de amplia experiencia de programación y administración. • Pobre interfaz gráfica + consola.

DameWare Remote Support	• Soporta Windows NT 4.0 c/Service Pack 1 o superior (incluso Windows 2000, XP, 2003, Vista, 2008, Windows 7, o Windows 8). • Soporta también MAC OS y Linux. • Los gastos son por conceptos de administración y no de computadora gestionada.	• Es una herramienta privativa de código “cerrado”. • Con un alto costo de adquisición y licencia. • Existe poca información sobre la herramienta para su consulta y estudio. • Está desarrollado en Visual C++ convirtiéndola en una solución compleja y de escritorio.
Desktop Central	• Permite la instalación de software, parches y service-packs. • Permite la configuración de aspectos de Windows incluyendo opciones de Internet Explorer, Outlook y Office y la auditoría con un completo inventario de hardware y software. • Los dispositivos administrados pueden estar en una LAN o distribuidos a través de una WAN, incluso en múltiples dominios o usuarios móviles. • Permite realizar búsquedas en los lenguajes Java, .Net y Ruby.	• Es una herramienta privativa de código “cerrado”. • Con un alto costo de adquisición y licencia. • Existe poca información sobre la herramienta para su consulta y estudio. • Es una solución de escritorio.
Teamviewer	• Permite el mantenimiento remoto, reuniones, videoconferencias, presentaciones y acceso a ordenadores o servidores remotos. • Es compatible con Windows, Mac, Linux, iOS, Android y BlackBerry OS. • Ofrece la posibilidad de habilitar cualquier computadora que esté en la red de área local a través de Wake-on-LAN por el enrutador.	• Es una herramienta privativa de código “cerrado”. • Con un alto costo de adquisición y licencia. • Es una solución de escritorio. • Para cada uno de sus componentes y actualizaciones es necesario adquirir una licencia.
SaltStack	• Rapidez, escalabilidad, flexibilidad, eficiencia. • Motor de administración remota. • Alta velocidad y seguridad en la red de	• No tiene interfaz gráfica.

	comunicación bi-direccional. • Administración remota paralela en milisegundos. • Administración de la configuración basada en Master-Minion. • Soporte para cualquier tipo de computación en la nube, infraestructura, sistema operativo, virtualización, aplicación de software, contenedor y código.	

Tabla 1 Comparativa entre herramientas de administración centralizada.

Después de haber comparado las herramientas homólogas y haber establecido las diferencias que hacen que SaltStack sea la mejor opción a tener en cuenta para desarrollar la nueva solución, es preciso describir cuáles tecnologías y herramientas serán utilizadas para su concepción.

1.3. Tecnologías y herramientas actuales

En la actualidad existen numerosas empresas e instituciones desarrolladoras de *software* las cuales han dado lugar a nuevas tecnologías para el desarrollo web, así como a herramientas, sistemas gestores de base de datos y lenguajes de programación.

1.3.1. Metodología para el desarrollo de software

Todo producto de software ha sido diseñado e implementado siguiendo una metodología de desarrollo de software. Básicamente, dichas metodologías suelen dividirse en dos grupos: metodologías pesadas (tradicionales) y metodologías ágiles. (Piattini, 2013), para seleccionar qué metodología es la más adecuada en el desarrollo del proceso de construcción de software en la presente investigación, se realizó un estudio de ambas.

De forma general las metodologías tradicionales, aunque también denominadas pesadas, clásicas o robustas, no resultan ser el enfoque más adecuado para muchos proyectos de software actuales, donde el entorno del sistema es muy cambiante, y en donde se exige reducir drásticamente los tiempos de desarrollo pero manteniendo una alta calidad.

Sin embargo las metodologías ágiles cumplen muy bien el rol de ser muy adaptables a los cambios que puedan suceder a lo largo del proceso de producción de software, en lugar de ser predictivos, ya que se

encuentran especialmente orientadas a proyectos pequeños y constituyen una solución a medida para ese entorno, aportando una elevada simplificación, que a pesar de ello no renuncia a las prácticas esenciales para asegurar la calidad del producto.

Las robustas se caracterizan por ser muy rígidas, algunos autores las catalogan como “camisas de fuerza”, y dirigidas por la gran cantidad de documentación que se genera en cada una de las actividades desarrolladas. Las ágiles por su parte, intentan adaptarse y crecer con el cambio, produciendo solo la documentación necesaria para solucionar el problema en cuestión. Estas últimas se encuentran, en gran medida, orientadas a las personas y no tanto al proceso; y por su parte, las tradicionales, tienden a intentar planear una gran parte del proceso del software en gran detalle para un plazo largo de tiempo, con el objetivo de definir un proceso que funcionará bien independientemente de quien lo utilice. (Torres Hernández, 2013).

Teniendo en cuenta que el tiempo para el desarrollo y entrega de los artefactos es corto y que se cuenta con solo dos desarrolladores, se propone el uso de una metodología ágil para el desarrollo de una nueva solución.

Para esta investigación se analizaron las metodologías ágiles SCRUM y Extreme Programming debido a que ambas promueven el trabajo en equipo, fomentan la interacción sistemática entre el cliente y el equipo de desarrollo y están suficientemente documentadas**.**

SCRUM

Scrum propone la ejecución de manera regular de un conjunto de buenas prácticas para trabajar colaborativamente en equipo y obtener el mejor resultado posible de un proyecto. Estas prácticas se apoyan unas a otras y su selección tiene origen en un estudio de la manera de trabajar de equipos altamente productivos.

En Scrum se realizan entregas parciales y regulares del producto final, priorizadas por el beneficio que aportan al receptor del proyecto. Por ello, Scrum está especialmente indicado para proyectos en entornos complejos, donde se necesita obtener resultados pronto, donde los requisitos son cambiantes o poco definidos, donde la innovación, la competitividad, la flexibilidad y la productividad son fundamentales.

Scrum también se utiliza para resolver situaciones en que no se está entregando al cliente lo que necesita, cuando las entregas se alargan demasiado, los costes se disparan o la calidad no es aceptable, cuando se necesita capacidad de reacción ante la competencia, cuando la moral de los equipos es baja y la rotación alta, cuando es necesario identificar y solucionar ineficiencias sistemáticamente o cuando se quiere trabajar utilizando un proceso especializado en el desarrollo de producto.

Programación extrema (Extreme Programming o XP).

XP por su parte, es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. Se basa en la retroalimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y flexibilidad para enfrentar los cambios. Se define como una metodología adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico. Los principios y prácticas son de sentido común pero llevadas al extremo, de ahí proviene su nombre. (Joskowicz, 2008)

La metodología XP tiene un conjunto importante de reglas y prácticas. En forma genérica, se pueden agrupar en: reglas y prácticas para la Planificación, para el Diseño, para el Desarrollo y para las Pruebas. Se diferencia de las demás metodologías en que exige un alto nivel de disciplina de las personas que participan en el proyecto. Se basa sobre todo en la programación y usualmente se utiliza el paradigma orientado a objetos, con un fuerte uso de patrones de diseño. Las características esenciales de XP se organizan en los tres apartados siguientes: historias de usuario, roles, procesos y prácticas. (Joskowicz, 2008)

1.3.2. Metodología seleccionada

Se decide utilizar XP debido a que se adapta en gran medida tanto al tipo de proyecto a desarrollar como a las condiciones de trabajo. A continuación se exponen varias de las razones que llevaron al uso de esta metodología.

- **El proyecto es pequeño:** la presente investigación se desarrolla a pequeña escala como solución para un problema tecnológico de la Facultad 4.

- **No existe un contrato previo especificando tiempo, recursos y alcance:** para el desarrollo de la propuesta de solución no se dispone de un contrato con un presupuesto previamente definido, puesto que es un proyecto para el uso interno de la facultad 4.
- **Los requisitos del sistema cambian frecuentemente:** al no existir un precedente del negocio en la Facultad 4, los requisitos cambian a medida que se va concibiendo y adaptando la solución.
- **El cliente forma parte del equipo de desarrollo:** a través de todo el desarrollo de la propuesta, el cliente está integrado constantemente ya que es el especialista en seguridad informática del Dpto. de Tecnología de la Facultad 4.
- El **riesgo de desarrollo** es elevado debido al corto tiempo de entrega planteado y a los continuos cambios de requisitos.
- **Poca disponibilidad de personal:** solo se cuenta con dos programadores, quienes tienen la posibilidad de intercambiar responsabilidades debido a la flexibilidad de XP, aunque no puedan especializarse en un rol en particular.
- Se emplearán las buenas **prácticas de XP** en relación a la Integración Continua y el Desarrollo Guiado por Pruebas.
- **Propiedad colectiva del código:** para el desarrollo, ambos programadores trabajarán de conjunto pudiendo codificar, revisar y corregir el trabajo del compañero, facilitando por tanto mayor dominio de la solución y la ejecución de las prácticas anteriormente descritas.

En general, XP maneja documentación fundamentalmente oral, relacionada con el código inherente al desarrollo y la transmisión de conocimiento tácito en vez de generar documentación y artefactos escritos. Sin embargo, mientras que la comunicación oral puede funcionar en pequeños grupos de desarrollo, no es recomendado emplear tal modalidad con grandes sistemas o con aquellos que requieren auditoría por razones legales o de confiabilidad ingenieril. En tales casos, es necesario emplear artefactos que permitan una gestión más formal, así como una mayor organización, registro y revisión como parte de un proceso de desarrollo más trazable. (Williams, 2007)

Los siguientes artefactos de XP, se describen para su mejor comprensión:

- **Historias de usuario:** fichas que muestran una breve descripción de requisitos funcionales. Estas contendrán los datos capturados a partir de las conversaciones entre el desarrollador y el cliente, donde se llegará a un acuerdo sobre aquellas restricciones que deben tenerse en cuenta para que

el requisito pueda ser completado. Asimismo se indicará la prioridad establecida por el propio cliente y la estimación para el desarrollo calculada por el programador, la cual no debe exceder el tiempo acordado para la iteración que contendrá tal funcionalidad.

- **Lista de tareas:** lista de tareas en correspondencia con las historias de usuario para completar una iteración. Cada tarea representa un aspecto concreto de cada historia de usuario y debe ser elegida por el programador, en vez de ser asignada al mismo.
- **Tarjetas CRC (Responsabilidad-colaboración por clases)**: aunque es opcional, estas tarjetas registran las responsabilidades por clases así como los colaboradores de las mismas, lo que constituye la base del diseño del sistema. Las clases, responsabilidades y colaboradores son identificados durante una tormenta de ideas para el diseño involucrando a diversos desarrolladores.
- **Artefactos de prueba:** son las pruebas de aceptación, las descripciones textuales y los casos de pruebas automatizados que son desarrollados por el cliente. El equipo de desarrollo demostrará la completitud de una historia de usuario y la validación de los requisitos del cliente empleando tales casos de prueba.
- **Gráficos y posters:** permiten facilitar la comunicación y la contabilización del progreso del desarrollo, ilustrando la cantidad de historias de usuario que han sido completadas o cuánto casos de prueba están siendo aplicados.

Como se mencionara anteriormente, también se seguirán las buenas prácticas recomendadas para el desarrollo con XP, a continuación se describen brevemente las más apropiadas para el equipo de desarrollo de este trabajo:

- Aplicar la técnica de programación en parejas, donde dos desarrolladores trabajan en la misma computadora colaborando en el mismo diseño, algoritmo, código o prueba.
- Integrar continuamente, para poder evidenciar el código completado y las pruebas asociadas a los mismos.
- Diseñar de manera incremental en vez de realizar un diseño detallado anticipando la implementación, de manera que cada día se pueda aportar algo nuevo en base a la experiencia acumulada.

- Desarrollar de manera incremental, para desplegar gradualmente las funcionalidades en un entorno real y así evitar los riesgos de los despliegues a gran escala.
- Involucrar al cliente real para evitar confusiones en el negocio, desviaciones teórico-prácticas y favorecer la toma de decisiones en cuanto a la prioridad de los requisitos funcionales. Además de que es el cliente quien escribe las pruebas de aceptación.
- Reducir el tamaño del equipo de desarrollo a medida que este se especializa y crece en cuanto a la experiencia adquirida.
- Realizar el análisis de los errores y deficiencias detectadas, a través de las pruebas de aceptación y las de unidad.
- Compartir el código, de manera que pueda ser modificado o actualizado por cualquier miembro del equipo. La propiedad colectiva sobre el código garantiza que no existan cuellos de botella en un componente específico si es necesario realizar algún cambio.
- Mantener como artefactos necesarios el código y las pruebas. Dependiendo de las necesidades del cliente y el entorno de su negocio, otros artefactos podrían ser desarrollados para facilitar retroalimentación, consulta bibliográfica, capacitación, mantenimiento o la modificación de elementos específicos del negocio que no afecten el propósito funcional de la aplicación. (Williams, 2007)

1.3.3. Lenguaje de modelado

La metodología seleccionada -XP- no plantea una gran cantidad de artefactos referentes a modelos y diseños para representar los sistemas, simplemente se deben generar los necesarios para que se entienda el completo funcionamiento del mismo. Por tanto, el lenguaje de modelado que se utilizará para representar los conceptos del negocio a través del modelo de dominio será el Lenguaje Unificado de Modelado (UML, del inglés Unified Modeling Language) en su versión V2.4.1.

Lenguaje Unificado de Modelado (UML)

UML es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema de software. Es importante resaltar que UML es un "lenguaje" para especificar y no para describir métodos o procesos. Se utiliza para definir un sistema de software, para detallar los artefactos en el sistema y para documentar y construir.

El objetivo de UML es proporcionar una arquitectura de sistema, ingeniería del software, y desarrollo de software con herramientas de análisis, diseño e implementación basada en los sistemas de software tanto para modelado de procesos de negocio, como para procesos similares (Magaña Orue, 2009).

Herramienta CASE (Ingeniería de Software Asistida por Computadora) para el modelado

Una vez seleccionado el lenguaje de modelado, elegir una herramienta CASE es de vital importancia porque es la que permitirá la representación gráfica de los elementos significativos de la solución para facilitar su comprensión.

Visual Paradigm

Visual Paradigm es una potente herramienta CASE que utiliza como lenguaje de modelado UML. Esta herramienta contribuye a una rápida construcción de aplicaciones de mayor calidad, a un menor coste. Además, permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. Soporta el ciclo de vida completo del desarrollo de *software*: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Facilita la interoperabilidad con otras herramientas CASE y puede integrarse con Eclipse/IBM WebSphere, Jbuilder, NetBeans IDE, Oracle Jdeveloper, BEA Weblogic. Además, está disponible en varias ediciones: Enterprise, Professional, Community, Standard, Modeler y Personal. (Saavedra, 2013)

Esta herramienta permite aumentar la velocidad en el análisis, captura, plan, desarrollo, comprobación y despliegue de los requisitos. Es multiplataforma y cuenta con una versión libre para la comunidad (Community Edition). Genera la documentación del sistema en los formatos PDF, HTML y el formato de documentos de Microsoft Word. Además, permite importar proyectos de otras herramientas de modelado como Rational Rose, Erwin y Microsoft Visio.

Entre sus características más importantes se destacan:

- Soporte UML hasta la versión 2.2.
- Modelado de Caso de uso.
- Generación de reportes.
- Ingeniería inversa de código Java, .NET, PHP5, Python.
- Ingeniería inversa - Código a modelo, código a diagrama.

- Integración con Visio - Dibujo de diagramas UML con plantillas (stencils) de MS Visio. (Cabrera, 2012)

Por todo lo anteriormente expuesto, se propone Visual Paradigm 8.0 ya que es una herramienta multiplataforma que cuenta con versiones gratuitas y provee fácil integración con el resto de las herramientas de desarrollo.

1.3.4. Lenguaje de programación

Se debe entender por lenguaje de programación al lenguaje codificado usado por los programadores para escribir instrucciones que la computadora pueda interpretar para realizar la operación que se le implementa. El lenguaje de computadoras de más bajo nivel es el lenguaje binario de máquinas de ceros y unos, el cual es ejecutado rápidamente sin programas intérpretes, pero es tedioso y complejo.

Sin embargo, los lenguajes de alto nivel como Basic, C y Java, son mucho más sencillos de usar y tienen una sintaxis más parecida al lenguaje humano, en idioma inglés; aunque deben emplear un programa compilador o intérprete para convertir el lenguaje de alto nivel en código de máquina. Existen por tanto, muchos lenguajes de programación y constantemente se crean nuevos con mejores y mayores beneficios. (Business Dictionary, 2015)

Lenguajes del lado del cliente

Lenguaje de Marcado de Hipertexto

El lenguaje de marcado de hipertexto es un lenguaje de computadoras basado en texto estandarizado para crear documentos electrónicos (hipertexto). Al ser un lenguaje de marcado, el valor de HTML no reside precisamente en el diseño de una estructura visual -fuente, espaciado de línea, capas- de un documento electrónico, sino en la formulación de su lógica estructural. Dicha estructura lógica permite el procesamiento de información, su indexación, comunicación y descubrimiento en la web. HTML soporta además, la inclusión de audio, video y animación en un documento electrónico a través de software asistentes como ActiveX, Java applets y Quick-Time. (Business Dictionary, 2015)

Hojas de Estilos en Cascada

Las hojas de estilos en cascada son una característica de HTML que habilita la aplicación de elementos de estilo tales como encabezados, fuentes y vínculos a cualquier página web, independientemente de su

contenido. Esto permite proveer páginas web relacionadas con estilos similares y además permitir modificaciones generales al cambiar el estilo de las plantillas en vez de las páginas individualmente. (Business Dictionary, 2015)

JavaScript

JavaScript es un lenguaje de programación que fue originalmente creado por la empresa Netscape con el nombre original de LiveScript y soportando gran cantidad de las instrucciones que tiene en la actualidad, con el propósito de añadir interactividad a las páginas web vistas con su navegador de Internet. Actualmente JavaScript está integrado en otras aplicaciones y otros navegadores de Internet, y es uno de los lenguajes más utilizados en la red de redes para añadir interactividad a las páginas web (Calvo, 1999).

JavaScript se diseñó teniendo Java en mente; pero, a pesar de que sus sintaxis es muy similar, son lenguajes muy distintos. A diferencia de Java, JavaScript no dispone de elementos para crear interfaces de usuario propias para los programas y tiene que utilizar para ello los formularios de HTML a través de los denominados manejadores de eventos. El código JavaScript se embebe en el código HTML de las páginas web añadiendo cierta “inteligencia” e interactividad a las mismas. La mayor parte de las páginas web modernas incluyen algo de código JavaScript, bien para obtener ciertos efectos estéticos (cambiar una imagen al pasarle por encima, gráfico por la pantalla, etc.), bien para validar una entrada de datos, hacer cálculos, cargar dinámicamente valores en listas desplegables entre otros componentes.

Los programas en JavaScript no generan ningún tipo de código compilado, sino que éste se interpreta en el navegador de Internet una vez se descarga la página que lo contiene. A este tipo de lenguajes se les denomina lenguajes interpretados. No se necesita ninguna herramienta especial para programar en JavaScript. Simplemente usando el bloc de notas de Windows se puede escribir código dentro de una página web, como por ejemplo el entorno integrado de desarrollo (IDE) de Visual Studio o el Dreamweaver (Oliva, 2003).

Las principales características de este lenguaje son:

- Es un lenguaje interpretado.
- No necesita compilación.
- Multiplataforma.
- Lenguaje de alto nivel.

- Admite programación estructurada.
- Basado en objetos.
- Maneja la mayoría de los eventos que se pueden producir sobre la página web.
- No se necesita ningún kit o entorno de desarrollo.

Lenguajes del lado del servidor

Procesador de Hipertexto

El Procesador de Hipertexto (PHP, del inglés HyperText Preprocessor), es un lenguaje de programación que se caracteriza por la fluidez y rapidez de sus scripts. Diseñado originalmente para el desarrollo web de contenido dinámico, es gratuito, multiplataforma, cuenta con una gran biblioteca de funciones y detallada documentación. Ofrece integración con numerosas bibliotecas externas y es compatible con varios Gestores de Bases de Datos (SGBD) como: MySQL y PostgreSQL (Baukes, 2013).

Marco de trabajo (Framework)

El término framework (marco de trabajo) se refiere a una estructura de software compuesta de componentes personalizables e intercambiables para el desarrollo de una aplicación. En otras palabras un framework se puede considerar como una aplicación genérica y configurable a la que se le puede añadir las últimas piezas para construir una aplicación concreta. Los objetivos principales que persigue este tipo de tecnología son: acelerar el proceso de desarrollo, reutilizar el código ya existente y promover buenas prácticas de desarrollo con el uso de patrones (Calderín, 2012).

Symfony v2.3.7

Symfony2 es un framework de desarrollo web PHP, pero también es una filosofía de trabajo y una comunidad próspera. Symfony2 facilita a los desarrolladores una forma de trabajo estructurada que añade valor a su trabajo, dando la posibilidad de definirse en un perfil de movilidad más amplia (Martínez, 2013).

El framework Symfony2 presenta las siguientes características, las cuales conllevaron a utilizarlo como marco de trabajo de esta solución: implementa el patrón Modelo Vista Controlador (MVC, del inglés Model View Controller), proporciona una estructura al código fuente, forzando al desarrollador a crear código más legible para un futuro mantenimiento, encapsula operaciones complejas en instrucciones sencillas, es un framework multiplataforma, usa Programación Orientada a Objetos (POO), posibilita soportar varios

gestores de base de datos como MySQL, Oracle y PostgreSQL, siendo importante por si en el futuro se decide cambiar el gestor y soporta integración con el ORM Doctrine y con los framework de la capa de presentación jQuery y Bootstrap. (Potencier, 2011)

Bootstrap

Bootstrap es un marco de trabajo de JavaScript de código abierto desarrollado por el equipo que creó Twitter. Este es una combinación de HTML, CSS y código de JavaScript diseñado para ayudar a construir componentes de la interfaz de usuario. Como Bootstrap fue programado para soportar HTML5 y CSS3, provee una amplia gama de plantillas para tipografías, formularios, botones, paneles de navegación y otros componentes de la interfaz, al igual que otros componentes opcionales de JavaScript. (SANDEEP, 2014)

1.3.5. Sistema gestor de base de datos (SGBD)

Un sistema de gestor de bases de datos (en inglés Database Management System, DBMS) es un tipo de software muy específico, dedicado a servir de interfaz entre la base de datos, el usuario y las aplicaciones que la utilizan. Permiten describir los elementos de datos con su estructura, sus interrelaciones y sus validaciones.

Se compone de un lenguaje de definición de datos, de un lenguaje de manipulación de datos y de un lenguaje de consulta. Un SGBD permite definir los datos a distintos niveles de abstracción y manipular dichos datos, garantizando la seguridad e integridad de los mismos (Romero y Valiente, 2011)

MySQL

MYSQL es un sistema de administración de bases de datos relacionales rápido, sólido y flexible. Es ideal para crear bases de datos con acceso desde páginas web dinámicas, para la creación de sistemas de transacciones on-line o para cualquier otra solución profesional que implique almacenar datos, teniendo la posibilidad de realizar múltiples y rápidas consultas (Welling y Thomson, 2001).

MySQL ofrece varias ventajas respecto a otros sistemas gestores de base de datos: Tiene licencia pública, permitiendo no solo la utilización del programa si no también la consulta y modificación de su código fuente. Resulta por tanto fácil de personalizar y adaptar a las necesidades concretas.

El programa está desarrollado en C y C++, lo que facilita su integración en otras aplicaciones desarrolladas igualmente en esos lenguajes. Puede ser descargada gratuitamente de internet y para aquellos que deseen que sus desarrollos en MySQL no sean “código abierto” existe también una licencia comercial.

MySQL utiliza el lenguaje SQL (Structured Query Language–Lenguaje de Consulta Estructurado) que es el lenguaje de consulta más usado y estandarizado para acceder a bases de datos relacionales. Soportan las sintaxis estándar del lenguaje SQL para la realización de consultas de manipulación, creación y de selección de datos (Converse, Park, y Morgan, 2006).

Es un sistema cliente/servidor, permitiendo trabajar como servidor multiusuario y de subprocesamiento múltiple, es decir, cada vez que se establece una conexión con el servidor, el programa servidor crea un subproceso para manejar la solicitud del cliente controlando el acceso simultáneo de un gran número de usuarios a los datos y asegurando el acceso solo a usuarios autorizados (Gilmore, 2006).

Es portable, es decir, puede ser llevado a cualquier plataforma informática. MySQL está disponible en más de veinte plataformas diferentes incluyendo las distribuciones más usadas del sistema operativo Linux, Mac OS X, UNIX y *Windows*.

Es disponible encontrar gran cantidad de software desarrollado sobre MySQL o que soporte MySQL. En concreto son de destacar diferentes aplicaciones *open source* para la administración de la base de datos a través de un servidor web.

Servidor web

Un servidor Web es un programa que implementa el protocolo HTTP (Protocolo de transferencia de hipertexto, por sus siglas en inglés). Este protocolo está diseñado para transferir lo que se llama hipertextos, páginas Web o páginas HTML, textos complejos con enlaces, figuras, formularios, botones y objetos incrustados como animaciones o reproducciones de sonidos. A continuación se describen importantes características del servidor Web Apache en su versión 2.4.7.

Apache v2.4.7

Apache es un servidor HyperText Transfer Protocol (HTTP) de código abierto que por su facilidad de configuración, robustez y estabilidad se encuentra desplegado en millones de servidores. Es multiplataforma, haciéndolo prácticamente universal, tecnología gratuita de código abierto, patentado a través de la licencia Apache, además es extensible. (apache.org, 2013).

Constituye el servidor HTTP más popular y usado, lo que permite que sea fácil de adquirir y brinda una amplia fuente de ayuda. Muestra, entre otras características, mensajes de errores altamente configurables, posee bases de datos de autenticación y negociado de contenido, facilita la utilización de Perl, PHP, Java y otros lenguajes script. (apache.org, 2013).

1.3.6. Entorno de desarrollo integrado (IDE)

Un entorno de desarrollo integrado o en inglés Integrated Development Environment (IDE) es un programa compuesto por un conjunto de herramientas para un programador. Puede dedicarse en exclusiva a un solo lenguaje de programación o bien, poder utilizarse para varios.

Un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (GUI). Los IDE pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes. Estos proveen un marco de trabajo amigable para la mayoría de los lenguajes de programación tales como C++, Java, C#, Visual Basic, Object Pascal, etcétera.

NetBeans IDE v7.4

Este IDE permite a los desarrolladores escribir, compilar, depurar y ejecutar programas informáticos. Es un IDE de código abierto escrito completamente en Java que permite crear aplicaciones de escritorio, web y aplicaciones para móviles utilizando los lenguajes Java, JavaFX, PHP, Java Script, Ruby y Ruby onRails, Groovy y Grails, y C/C++, además de presentar soporte para la tecnología Ajax. Está disponible para múltiples plataformas como son Windows, Mac, Linux y Solaris

Se seleccionó NetBeans IDE como entorno de desarrollo, ya que los desarrolladores tienen pleno conocimiento acerca de las funciones del mismo, además por las siguientes características definidas por (Maldonado, 2007):

- Creación de proyectos PHP: NetBeans provee a los desarrolladores de una estructura para los proyectos que se puedan crear junto a este IDE. Propone un esqueleto para organizar el código fuente, el editor conjuntamente integra los lenguajes como HTML, JavaScript y CSS. Además, NetBeans posee un sistema para examinar todos los directorios de cada proyecto, haciendo reconocimiento y carga de clases, métodos y objetos, para acelerar la programación.
- Editor de código fuente: mejora en su editor, sobre todo en el editor de PHP, es mucho más rápido y a la vez robusto, contiene más ayuda en línea.
- Depuración de PHP: NetBeans integra muy bien la utilización Xdebug, permitiendo inspeccionar y examinar cada variable local, establecer puntos de interrupción y evaluar el código en nuestra lógica.
- Integración con sistemas de control de versiones: esta es una de las condiciones necesarias para los proyectos y es la posibilidad de contar con la integración de sistemas de control de versiones, tales como SVN, CVS, Mercurial y Git. Desde el editor es posible realizar la administración de estos sistemas versionados.

1.3.7. Sistema de Control de Versiones

Una de las herramientas que se utiliza para el desarrollo del presente trabajo es el Sistema de Control de Versiones del inglés Versions Control System, puesta a disposición a la comunidad universitaria para su uso bajo el nombre de GitLab. Esta es una herramienta que ayuda mucho a los programadores a tener control sobre su código, además de ser un sistema distribuido. Permite tener repositorios locales y remotos, brindando la posibilidad de ser accedidos por varios usuarios. Se presenta como aplicación *opensource*[4], y la administración de los repositorios se realiza mediante una interfaz web. Una de las características más notables de esta herramienta es que permite ver el código y editarlo en la propia

[4] Se refiere a herramientas de código abierto o licencia libre.

interfaz web que provee. Además brinda la opción de poder descargar el código del proyecto, ya sea a través de descarga directa o a través de *commits*[5] mediante la consola (Kotov, 2013).

[5] Envío de datos que se ejecuta para actualizar información en un determinado contexto.

1.4. Conclusiones parciales del capítulo

- Se estableció un marco conceptual con los principales conceptos del dominio del negocio a través de la definición de administración, control, centralización, administración centralizada de computadoras y eficiencia, que permitió comprender cabalmente las particularidades de esos términos.
- Se consolidó el basamento teórico mediante un estudio de las aplicaciones de administración centralizada de computadoras, evidenciándose las dificultades que las privaban de ser la solución al problema planteado.
- Se seleccionaron las tecnologías y la metodología de desarrollo de software a emplear a través de un estudio y descripción de estas para ir documentando sus características y funcionalidades.

Capítulo 2: Descripción de la propuesta de solución

Introducción

Una vez concluido el estudio del estado del arte, la descripción del precedente teórico de la investigación y definido el ambiente de desarrollo de software, es necesario describir la propuesta de solución. Para ello se registrarán el modelo de dominio, la lista de reserva del producto, los roles involucrados, las historias de usuario, el plan de iteraciones, los prototipos de interfaz de usuario, las tarjetas CRC, el modelo de datos y algunos de los patrones de diseño a emplear.

2.1. Descripción de la solución

En el capítulo anterior se estudiaron las herramientas Chef, Ansible, Puppet y SaltStack que constituyen un precedente de la solución que se propone en esta investigación. A partir de la *Tabla 1 Comparativa entre herramientas de administración centralizada,* se pudieron definir aquellos elementos que tributan a la construcción de la solución teniendo en cuenta las funcionalidades que se deben implementar y las especificidades de administración centralizada con las que debe cumplir. Debe entenderse entonces, que se pretende realizar una administración asistida por una interfaz gráfica que permita la asignación simultánea de tareas de configuración a un conjunto de máquinas de un área específica seleccionable en dicha interfaz.

Además la nueva solución debe poder incluir las mejores prácticas de cada una de sus predecesoras, por ejemplo debe lograr una automatización en tiempo real, la administración remota paralela en milisegundos, debe ser compatible con diversos navegadores web, soportar la arquitectura Master-Minion de SaltStack, debe tener un bajo costo administrativo y operacional, una única plataforma interna, implementación y mantenimiento fáciles y rápidos, así como lograr la simultaneidad de asignación de tareas a un conjunto específico de computadoras, crear áreas para una mejor organización e identificación de las computadoras y simplificar las tareas de configuración al sustituir la escritura de los comandos en la consola de SaltStack por instrucciones implementadas en botones de la interfaz.

Para una mejor comprensión de la solución se deben explicar las cuatro secciones principales a ser desarrolladas en la solución SweetSalt:

Áreas: esta sección es relativa a las áreas físicas en las que están ubicadas las computadoras conectadas en una red local, por ejemplo los departamentos docentes, los laboratorios, las oficinas y aulas.

Computadoras: relativo a las máquinas computadoras, la cuales se registran en el sistema realizando peticiones al servidor, el cual puede aceptar o rechazar las llaves de las mismas.

Comandos: se entiende por las instrucciones escritas en la consola de SaltStack y que en la presente propuesta se gestionan a través de *ítems* que pueden ser guardados, modificados, eliminados o ejecutados individualmente.

Usuarios: son los clientes de la aplicación que emplean el rol de administrador convirtiéndolos en usuarios del sistema. Estos realizan todas las operaciones sobre SweetSalt y al mismo tiempo tienen la posibilidad de crear, listar, modificar y eliminar usuarios.

La selección del tipo de aplicación e interfaz gráfica se sustentó en la necesidad concreta planteada en la problemática de la investigación y en las ventajas de los principios del desarrollo web:

- Las aplicaciones web son más sencillas y económicas de costear que las aplicaciones de escritorio porque para su desarrollo no hace falta instalar ningún programa en los ordenadores de los usuarios, solamente deben tener un navegador web (Chrome, Firefox, Opera) y además poseer conexión a la red.
- Pueden ejecutarse en cualquier sistema operativo como Windows o Linux sin restricciones.
- Su costo de mantenimiento no es elevado porque solo se necesita actualizar la aplicación en el servidor (Mora, 2012).

Entonces a partir de entender a SweetSalt como una aplicación web, para contribuir con la comprensión de la misma, en la *Ilustración 1. Arquitectura de SweetSalt* se muestran las especificidades de su arquitectura que se encuentra en correspondencia con la de SaltStack basada en agentes.

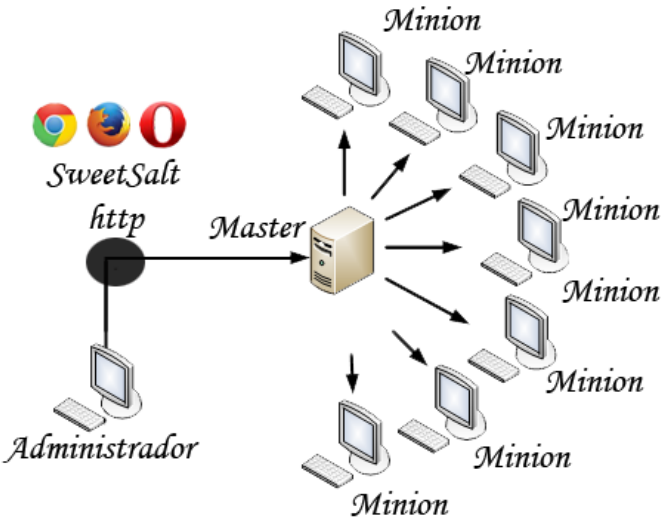


Ilustración 1. Arquitectura de SweetSalt

2.2. Modelo de dominio

XP como metodología para el desarrollo de software gira más su atención a los procesos que guiarán la implementación del sistema. Entre los artefactos que genera no es visible el diagrama "Modelo de Dominio", pero se utiliza con el objetivo de lograr un mejor entendimiento de los principales conceptos que se manejan en el negocio. A continuación se puede observar en la *Ilustración 2. Modelo de Dominio* la representación de la necesidad de desarrollar la solución propuesta, en un diagrama de dominio.

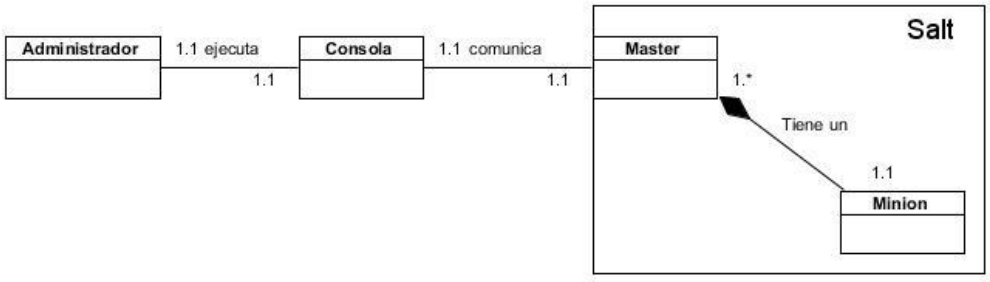


Ilustración 2. Modelo de dominio

- **Administrador:** es el usuario encargado de ejecutar la consola para comunicarse con el master.
- **Consola:** es el medio a través del cual se efectúa la comunicación entre el administrador y el master.

- **SaltStack:** herramienta de administración centralizada que permite la administración de las estaciones de trabajo, supeditando las estaciones Minions al Master.
- **Master:** servidor de la herramienta SaltStack donde se guarda toda la información referente a los Minions y de igual modo se les controla desde este.
- **Minion:** computadoras sobre las que se realiza la administración centralizada a través de la herramienta SaltStack.

2.3. Técnicas empleadas para la captura de los requisitos

Para poder identificar las funcionalidades o requisitos que debe integrar la propuesta de solución es necesario desarrollar alguna forma de captación de esa información. Para ello el equipo de desarrollo comienza por extraer datos de cualquier fuente disponible y confiable. Realizar esa actividad puede resultar complicado, principalmente si el entorno de trabajo es desconocido para el equipo de analistas, y depende mucho de las personas que participen en él. Teniendo en cuenta todo ello, la ingeniería de requisitos propone el uso de técnicas que permitan hacer estas actividades más sencillas pero precisas, entre ellas se encuentra la Entrevista que por su tipología y facilidades fue la seleccionada para este trabajo.

Entrevista

Es una técnica para recopilar información a partir de un intercambio directo entre individuos o grupos de personas. Puede realizarse a partir del uso de cuestionarios, listas de preguntas, conversación libre, entre otras formas y de manera general permite sostener un diálogo diáfano y simple con el cliente para facilitar la comunicación y el mutuo entendimiento de los intereses de ambas partes involucradas. A partir de la aplicación de esta técnica se pudieron obtener 18 requisitos funcionales y 14 no funcionales.

2.4. Lista de reserva del producto

En esta sección se define la lista de reserva del producto (LRP), la cual es una lista priorizada que define el trabajo a realizar en el proyecto. En ella se incluyen los requisitos sobre el producto, por lo que puede crecer y modificarse a medida que se obtiene más conocimiento acerca del producto y del cliente. En la LRP se incluyen los requisitos que demanda el cliente del sistema, separados por su prioridad (muy alta, alta, media o baja). Este sistema de clasificaciones por su prioridad indica la necesidad que tiene el sistema de dicha funcionalidad. Las muy altas, son funcionalidades indispensables para cubrir las

necesidades del cliente; mientras que las bajas son aquellas que se desean incluir, pero sin las que el sistema aún podría ser funcional y útil. Además, se enuncian los requisitos no funcionales especificados por el cliente como imprescindibles.

En función de lograr una mayor organización de tales funcionalidades, se decide aplicar un principio de agrupación sustentado en la 8va Conferencia de Ingeniería y Tecnología de Latinoamérica y el Caribe, realizada en Arequipa, Perú, en junio del 2010, donde se explican las diferentes formas de agrupar los requisitos:

- Si la aplicación a desarrollar tiene distintas categorías de usuarios (roles), los requisitos del sistema pueden ser especificados por roles. (Dependiente, Gerente, Administrador) (IEEE 830, 1998).
- Otra forma de agrupar los requisitos es atendiendo a los objetivos o servicios que se desea que ofrezca el sistema y que requiere una determinada entrada para obtener su resultado. Para cada objetivo o sub-objetivo requerido del sistema, se detallarán las funciones que permitan llevarlo a cabo (IEEE 830, 1998).
- Los requisitos también pueden ser descritos por jerarquía funcional. La funcionalidad del sistema se especifica como una jerarquía de funciones que comparten entradas, salidas o datos del propio sistema. Para cada función y sub-función del mismo se detallará la entrada, el proceso en el que interviene y la salida. Normalmente este tipo de análisis implica que el diseño siga el paradigma de diseño estructurado. Por lo general este sistema se utiliza cuando ninguno de los anteriores se puede aplicar (IEEE 830, 1998).

Se decide agrupar y especificar los requisitos de acuerdo a la segunda opción, quedando los mismos contenidos en Agrupaciones de Requisitos Funcionales (ARF) para concentrar por objetivos las funcionalidades en cuestión y simplificar por tanto su administración y desarrollo.

A continuación se presenta la LRP mediante la Tabla 2: Lista de reserva del producto.

ARF	Descripción	Tareas	Prioridad	Estimación	Complejidad	Responsable
1	Administrar usuario	Crear usuario	Muy alta	1 semana	Baja	Manuel García
		Autenticar usuario	Muy alta		Media	
		Editar usuario	Alta		Media	

		Eliminar usuario	Alta		Baja	
2	Administrar computadoras	Listar llaves	Muy alta	3 semanas	Baja	Manuel García
		Aceptar llave	Muy alta		Baja	
		Rechazar llave	Alta		Baja	
		Crear computadora	Muy alta		Media	
3	Administrar áreas	Listar áreas	Muy alta	3 semanas	Baja	Manuel Román Manuel García
		Crear área	Muy alta		Baja	
		Asignar computadora	Muy alta		Media	
		Editar área	Alta		Baja	
		Eliminar área	Media		Baja	
4	Administrar comandos	Listar comandos	Muy alta	4 semanas	Alta	Manuel García Manuel Román
		Crear comando	Muy alta		Baja	
		Editar comando	Alta			
		Ejecutar comando	Muy Alta		Alta	
		Eliminar comando	Media		Baja	

Características del sistema

Usabilidad

- **RNF 1.** Los elementos gráficos como los íconos deberán contar con un *tooltype* o mensaje flotante que señalen el tipo de recurso al que se refiere.
- **RNF 2.** La aplicación deberá contar con una interfaz y navegación funcional, asequible a todo tipo de usuario.
- **RNF 3**. El sistema debe mantener al usuario informado de las operaciones que este realiza a través de la interacción con el mismo.

Soporte

- **RNF 4.** La aplicación debe ser implementada bajo tecnología web, puesto que la misma será accedida a través de la Web.
- **RNF 5.** Ejecutarse sobre cualquier navegador, siendo como mínimo compatible con:
  - Mozilla Firefox 7.0 y superior.
  - Opera 10.0.0 y superior.
  - Chrome 7.0 y superior.
- **RNF 6.** Los usuarios finales deberán contar como mínimo con:
  - Procesador Pentium II o superior.
  - 512 MB de RAM.
  - 15 GB de HDD.
  - Si no cuentan con un servidor local: se necesita una conexión de banda ancha de 256Kbps como mínimo.

Restricciones de diseño e implementación

- **RNF 7.** Se aplicará la programación orientada a objetos.
- **RNF 8.** El marco de trabajo de desarrollo que se utilizará es: Symfony v2.3.7.
- **RNF 9.** Como IDE se empleará NetBeans v7.4.
- **RNF 10.** Se empleará como SGBD MySQL
- **RNF 11.** Como servidor web se empleará Apache v2.4.7.
- **RNF 12.** Se utilizará el estándar de codificación de Symfony2.

Seguridad

- **RNF 13.** Cuando una sesión permanece inactiva durante un tiempo especificado, el sistema debe cerrar dicha sesión de forma automática.

Apariencia o Interfaz externa

- **RNF 14.** Mantendrá en todo momento un diseño sencillo, con pocas imágenes y gráficos, con el objetivo de elevar el nivel de respuesta del sistema ante las peticiones del usuario.

Tabla 2 Lista de reserva del producto.

2.5. Descripción de los roles del sistema

Los roles identificados son aquellas personas que van a interactuar con la aplicación y en el caso específico de esta investigación, se trata del administrador.

Rol	Descripción
Administrador	Este es el único rol registrado en el sistema y tiene todos los privilegios para hacer uso total de la aplicación.

Tabla 3 Descripción de los roles del sistema.

2.6. Exploración

Es la fase en la que se define el alcance general del proyecto. En esta fase, el cliente define lo que necesita mediante la redacción de sencillas “historias de usuarios”. Los programadores estiman los tiempos de desarrollo en base a esta información. Debe quedar claro que las estimaciones realizadas en esta fase son primarias (ya que estarán basadas en datos de muy alto nivel) y podrían variar cuando se analicen más en detalle en cada iteración (JOSKOWICZ, 2008).

El propósito de la exploración es llegar a un entendimiento global entre desarrolladores y el cliente acerca de qué es lo que el futuro sistema debe hacer. Para ello se toman los requisitos funcionales que se han definido y comienza el proceso de descripciones de las historias de usuario, los análisis en grupo y las

tormentas de ideas. Se esclarecen las dudas sobre procesos que deben ser automatizados y que no han sido bien entendidos (Hidalgo Urbino, 2010).

2.6.1. Historias de usuario

Las historias de usuario (en lo posterior HU), son la forma en que se especifican en XP los requisitos del sistema, las mismas no deben ser descritas en más de tres líneas e idealmente es el cliente quien las redacta y prioriza. Por tanto serán descripciones cortas y escritas en el lenguaje del usuario, sin terminología técnica (Cruz Castro, 2011).

Las HU pueden agrupar uno o varios requisitos funcionales identificados. XP recomienda que las HU sean escritas por el propio cliente. Es por ello que debe existir una buena relación entre los clientes y el grupo de desarrollo, la falta de comunicación entre ambos factores influye directamente en el fracaso del proyecto. Las HU son priorizadas y colocadas de acuerdo a su prioridad en las iteraciones (Hidalgo Urbino, 2010). A continuación se muestra la HU relacionada con el ARF 1. Administrar usuario. En el Anexo 1: Especificaciones Historias de Usuario se encuentran todas las HU para su consulta.

Especificación de HU

HU1 ARF1. Administrar usuario

RF relacionados:

- **RF 1.1** Crear usuario en el sistema.
- **RF 1.2** Editar usuario.
- **RF 1.3** Eliminar usuario del sistema.

Historia de usuario	
No.:1	**Nombre:** Crear usuario
Usuario: Administrador	
Prioridad en el negocio: Muy alta	**Nivel de complejidad:** Bajo
Estimación: 2d	**Iteración asignada:** 1
Descripción: se crea un usuario de tipo administrador en el sistema.	
Información adicional (observaciones): No aplica.	
No.:2	**Nombre:** Autenticar usuario
Usuario: Administrador	
Prioridad en el negocio: Muy alta	**Nivel de complejidad:** Medio

Estimación: 1d	**Iteración asignada:** 1
Descripción: se autentican los usuarios especificando nombre de usuario y contraseña.	
Información adicional (observaciones): No aplica.	
No.:3	**Nombre:** Editar usuario
Usuario: Administrador	
Prioridad en el negocio: Alta	**Nivel de complejidad:** Medio
Estimación: 1d	**Iteración asignada:** 1
Descripción: se editan los usuarios creados por concepto de cambio de nombre o cambio de contraseña.	
Información adicional (observaciones): se ejecuta cuando es necesario modificar la información de acceso de un usuario específico.	
No.:4	**Nombre:** Eliminar usuario del sistema
Usuario: Administrador	
Prioridad en el negocio: Alta	**Nivel de complejidad:** Baja
Estimación: 1d	**Iteración asignada:** 1
Descripción: se elimina un usuario específico del sistema borrando de igual manera todos sus datos asociados y los registros de navegación e interacción con el mismo.	
Información adicional (observaciones): No aplica.	

Tabla 4 HU ARF1. Administrar usuario.

2.7. Planificación

La planificación es una fase corta, en la que el cliente, los gerentes y el grupo de desarrolladores acuerdan el orden en que deberán implementarse las HU y asociadas a estas, las entregas. Típicamente esta fase consiste en una o varias reuniones grupales de planificación. El resultado de esta fase es un Plan de Entregas (JOSKOWICZ, 2008).

Esta fase de planificación se muestra como una de las más importantes dentro de la metodología XP, ya que se estiman los esfuerzos, por parte de los desarrolladores, para implementar las funcionalidades de la aplicación. Se necesita un constante intercambio con el cliente para suplir cualquier duda que surja durante el desarrollo del sistema y de existir algún cambio, que todas las partes involucradas sean informadas del mismo y de las consecuencias reales que arraigará tal evento. Además se hace necesario planificar la utilización de recursos humanos y materiales para su utilización en cualquier circunstancia del proceso.

Historias de usuario	Tareas por cada ARF	Estimación (días)
ARF1. Administrar usuario	Crear usuario	2

	Autenticar Usuario	1
	Editar usuario	1
	Eliminar usuario	1
ARF2. Administrar computadoras	Listar llaves	4
	Aceptar llave	4
	Rechazar llave	2
	Crear computadora	3
ARF3. Administrar áreas	Listar áreas	4
	Crear área	4
	Asignar computadora	2
	Editar área	4
	Eliminar área	3
ARF4. Administrar comandos	Crear evento	5
	Listar eventos	4
	Ejecutar evento	5
	Editar evento	4
	Eliminar evento	2

Tabla 5 Estimación de esfuerzos por HU.

2.7.1. Plan de iteraciones

Cuando se concluye el proceso de definir las Historias de usuario en la fase de planificación para guiar la etapa de implementación de la solución se procede a desarrollar el plan de iteraciones. Este plan se construye en la fase de Iteración y consiste en un documento que muestra la cantidad de Historias de Usuario definidas por el cliente en cada iteración y el tiempo estimado para completar las iteraciones, con el objetivo de planificar el periodo de implementación del sistema.

La implementación de las HU se planificó para que fueran realizadas, en primer orden, las de mayor interés para el negocio. Además se tuvo en cuenta que, aquellas funcionalidades que dependían de otra para su implementación, se colocaran en la misma iteración, permitiendo que el flujo de trabajo se realizara de forma correcta y de acuerdo a lo establecido por el cliente. Para cada iteración se destinó una duración de 3 semanas, cada semana con 5 días laborables de 8 horas cada uno, cumpliendo con las 40 horas semanales que plantea la metodología XP.

Para el desarrollo de la primera versión del sistema se estiman 6 semanas para la implementación. En la *Tabla 6. Plan de iteraciones* que se muestra a continuación se evidencia el plan de iteraciones propuesto:

Iteraciones	Orden de las HU la implementar	Prioridad	Días	Total de días de la iteración
Iteración 1	HU1. Crear usuario en el sistema	Alta	2	18
	HU2. Autenticar Usuario	Alta	1	
	HU3. Editar usuario	Alta	1	
	HU4. Eliminar usuario del sistema	Alta	1	
	HU5. Listar llaves	Alta	4	
	HU6. Aceptar llave	Alta	4	
	HU7. Rechazar llave	Alta	2	
	HU8. Crear computadora	Alta	3	
Iteración 2	HU9. Listar áreas	Alta	4	37
	HU10. Crear área	Alta	4	
	HU11. Asignar computadora	Alta	2	
	HU12. Editar área	Media	4	
	HU13. Eliminar área	Baja	3	
	HU14. Crear evento	Muy alta	5	
	HU15. Listar evento	Muy alta	4	
	HU16. Ejecutar evento	Muy alta	5	
	HU17. Editar evento	Alta	4	
	HU18. Eliminar evento	Media	2	

Tabla 6 Plan de iteraciones.

2.8. Prototipos de interfaz de usuario

A continuación se muestra en la *Ilustración 3. Prototipo de Interfaz de usuario: Administrar usuario* un prototipo de baja fidelidad que permite validar la captura de información con el cliente, mejorar la comunicación en el equipo de desarrollo y representar gráficamente una funcionalidad específica de la herramienta.

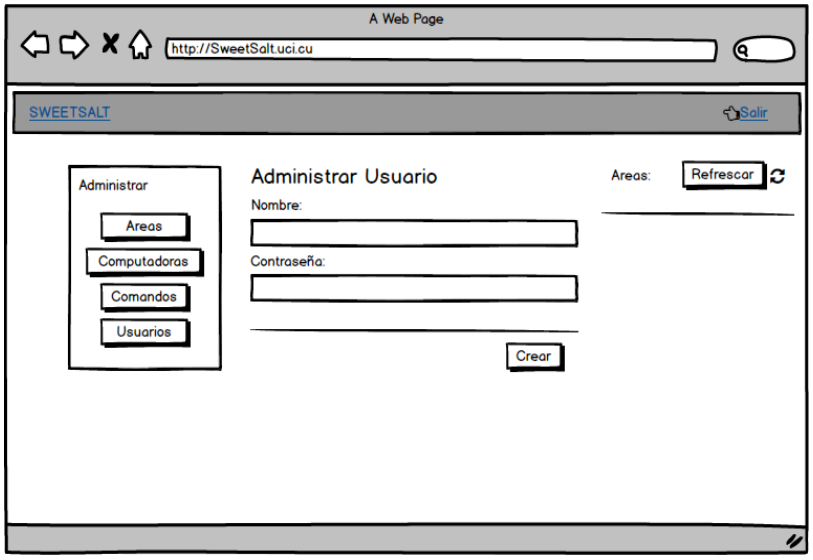


Ilustración 3. Prototipo de Interfaz de usuario: Administrar usuario

2.9. Tarjetas Clases – Responsabilidades – Colaboraciones

Las tarjetas CRC se definen para identificar las principales clases por las cuales está compuesta la aplicación informática y para definir, por cada una de las clases identificadas, la función que ejerce en el flujo de negocio y las relaciones que mantienen con otras clases del sistema. Este artefacto se especifica con el objetivo de obtener un diseño simple y evitar la implementación de funcionalidades innecesarias, además de proveer mejoría a las clases analizadas. Las tarjetas CRC pueden ser consultadas en el Anexo 2: Especificación de las tarjetas CRC.

2.10. Modelo de datos

La construcción de la base de datos es una de las tareas principales en el diseño de una aplicación, en esta se ponen de manifiesto los datos necesarios para el correcto funcionamiento de la misma.

2.10.1. Diseño formal de una base de datos

Una base de datos relacional es aquella que sigue el llamado modelo relacional formulado inicialmente por E. F. Codd, un investigador de IBM. El modelo relacional describe las bases de datos, tablas, registros, campos y operadores como *select*, *project* y *join* de un modo formal, matemático. Uno de los puntos

fuertes del modelo relacional es que es matemáticamente completo, no contiene incoherencias ni le faltan vínculos. (Buyens, 2001)

Al proceso de organizar los datos de una base de datos se le conoce como normalización, en el cual se incluye la creación de tablas y el establecimiento de relaciones entre ellas según reglas diseñadas tanto para proteger los datos como para hacer que la base de datos sea más flexible al eliminar la redundancia y las dependencias incoherentes. (Support Microsoft, 2013)

En una base de datos normalizada lo habitual es que exista un mayor número de tablas que en otra que no lo está. En consecuencia, las tablas normalizadas suelen ser más pequeñas y con menos atributos que las tablas no normalizadas.

A continuación se describen brevemente las 6 formas de normalización para las bases de datos, partiendo desde la primera forma hasta la tercera, incluyendo la forma especial de Boyce-Codd y las dos últimas formas.

- **Primera forma normal:** un campo dado de un registro dado, solo puede contener un valor, para evitar la redundancia de grupos dentro de un único registro.
- **Segunda forma normal:** debe cumplir con las restricciones de la primera forma normal, así como cada campo no clave debe depender de la clave principal de la tabla. Además dos o más tablas no pueden tener la misma clave principal, pues en caso de que esto suceda, para normalizar se combinan dichas tablas especializándolas en una sola, con una única clave principal.
- **Tercera forma normal:** debe cumplir con las restricciones de la segunda forma normal y no deben existir dependencias funcionales transitivas.
- **Forma normal de Boyce-Codd:** debe cumplir con las restricciones de la tercera forma normal y cada atributo determinante debe ser una clave candidata.
- **Cuarta forma normal:** debe cumplir con todas las restricciones de la 3era forma normal y no debe tener dependencias multievaluadas.
- **Quinta forma normal:** debe cumplir con las restricciones de la cuarta forma normal y cada entidad debe supeditar sus dependencias con las claves candidatas. (Oracle, 2010)

A partir del estudio documentado previamente, se puede concluir que la base de datos para la propuesta de solución inherente a este trabajo se encuentra en 3FN. Ello implica que no existan relaciones demasiado complejas entre tablas que provoquen dificultades a la hora de recuperar información en forma de consultas difíciles de expresar por parte del programador o el usuario. En la *Ilustración 3. Modelo de datos*, puede evidenciarse gráficamente dichos elementos.

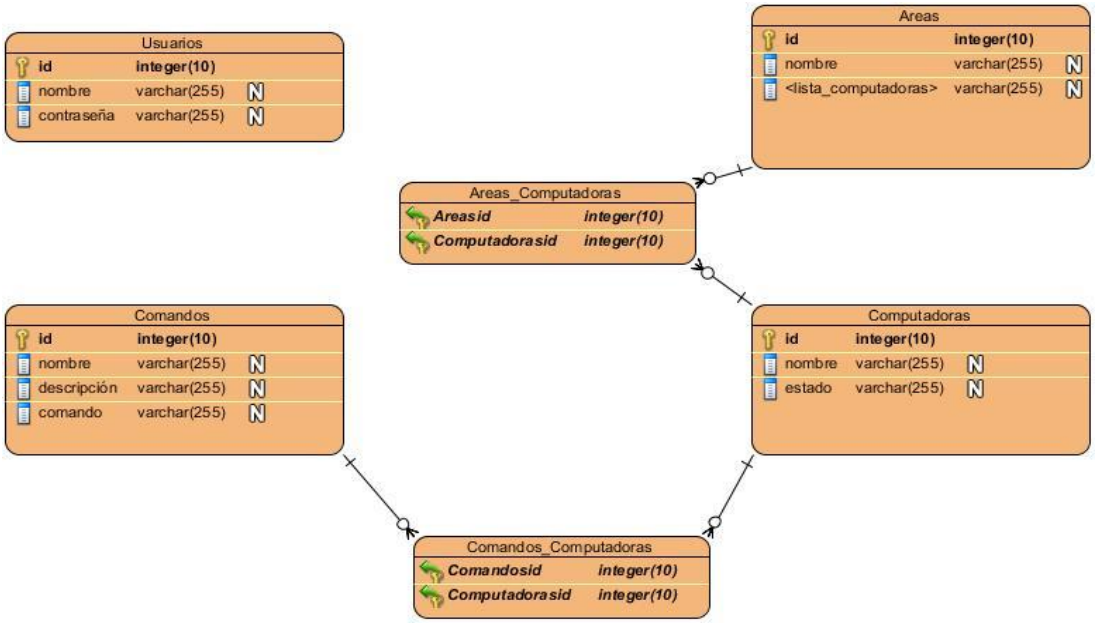


Ilustración 4. Modelo de datos

2.11. Modelo arquitectónico

En la construcción de software existen diversos componentes que proporcionan una filosofía de trabajo para los equipos de desarrollo, creando una visión que unifica la forma en que los miembros del equipo ven el sistema y además impone una transformación del mismo. Entre los mencionados componentes se encuentran los estilos arquitectónicos, patrones arquitectónicos y patrones de diseño, los cuales se encuentran estrechamente relacionados, conformando la base de la arquitectura de un software.

Existen diversas definiciones en lo que respecta a arquitectura de software, pero todas se ven reflejadas en las siguientes ideas fundamentales:

- La arquitectura de software como un proceso particular dentro del ciclo de vida.

- La arquitectura de software como la topología o la forma de articular distintos componentes en una solución.
- La disciplina académica y profesional de la arquitectura de software.

La arquitectura de software de un programa o sistema de computación es la estructura o estructuras del Sistema, que comprenden componentes de software, las propiedades externamente visibles de esos componentes, y las relaciones entre ellos. (Pressman 2010).

2.11.1. Patrón arquitectónico Modelo-Vista-Controlador

Una de las razones por las que se utiliza el patrón Modelo-Vista-Controlador (MVC) es por el nivel de organización que provee en las aplicaciones informáticas que, conjuntamente con la estructura interna que plantea el framework Symfony2, hace posible que la envergadura con que se desarrolla un sistema informático fluya de forma organizada. Este patrón separa la arquitectura de una aplicación en tres capas: modelo, vista y controlador. Cada una de estas capas tiene su funcionamiento específico, y una depende de otra para cumplir con sus responsabilidades dentro de la arquitectura.

El **Modelo** representa la información con la que trabaja la aplicación web, representando un intermediario entre la base de datos y el resto del sistema. Es el responsable del acceso directo a los datos, a través de funciones y restricciones que especifica la lógica del negocio. Esta capa hace uso del ORM Doctrine dentro de la arquitectura que plantea el framework Symfony2, el cual define funcionalidades y estructura para realizar las consultas y extracción de datos.

La **Vista** es la representación visual de los datos que provee el sistema, permitiendo que el usuario interactúe con el mismo de forma dinámica. Symfony2 plantea una estructura denominada Tres Capas[6], para el sistema de interfaces de usuario, utilizando Twig, el cual se presenta como un lenguaje de plantillas moderno, seguro, rápido y con el que se puede crear plantillas concisas y muy fáciles de mantener (Eguiluz 2015).

El **Controlador** es quien ejecuta las acciones dentro de la lógica del sistema. Funge como un intermediario entre las peticiones del usuario y lo que el sistema debe mostrar. Esta capa basa su

[6] Representa una buena práctica en el trabajo con el framework Symfony2 y define que la aplicación debe estar estructurada en tres capas de plantillas fundamentales: plantilla para representar el maquetado de toda la aplicación, plantilla para representar la estructura general de las interfaces y las plantillas específicas para cada funcionalidad de la lógica del negocio.

funcionamiento en las solicitudes del usuario, obteniendo datos del modelo y visualizándolos nuevamente al usuario en una plantilla perteneciente a la vista.

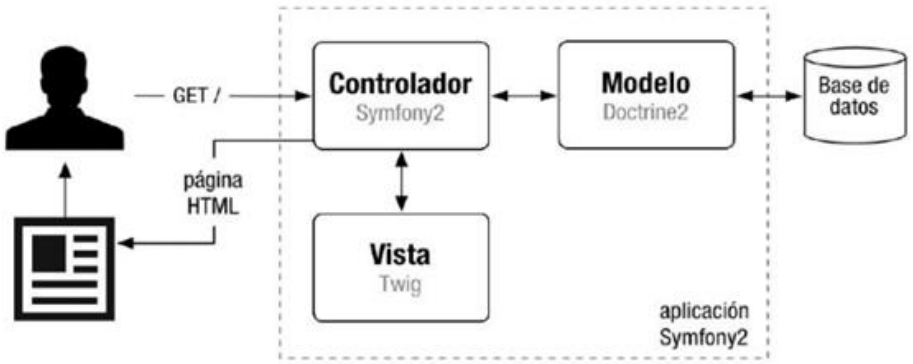


Ilustración 5. Esquema de la arquitectura interna de Symfony2 utilizando el patrón MVC.

Dicho patrón puede ser evidenciado a través del caso de estudio Administrar usuario, donde cada elemento está separado de acuerdo a la función que realice e interconectado siguiendo las pautas del diseño de clases con estereotipos web.

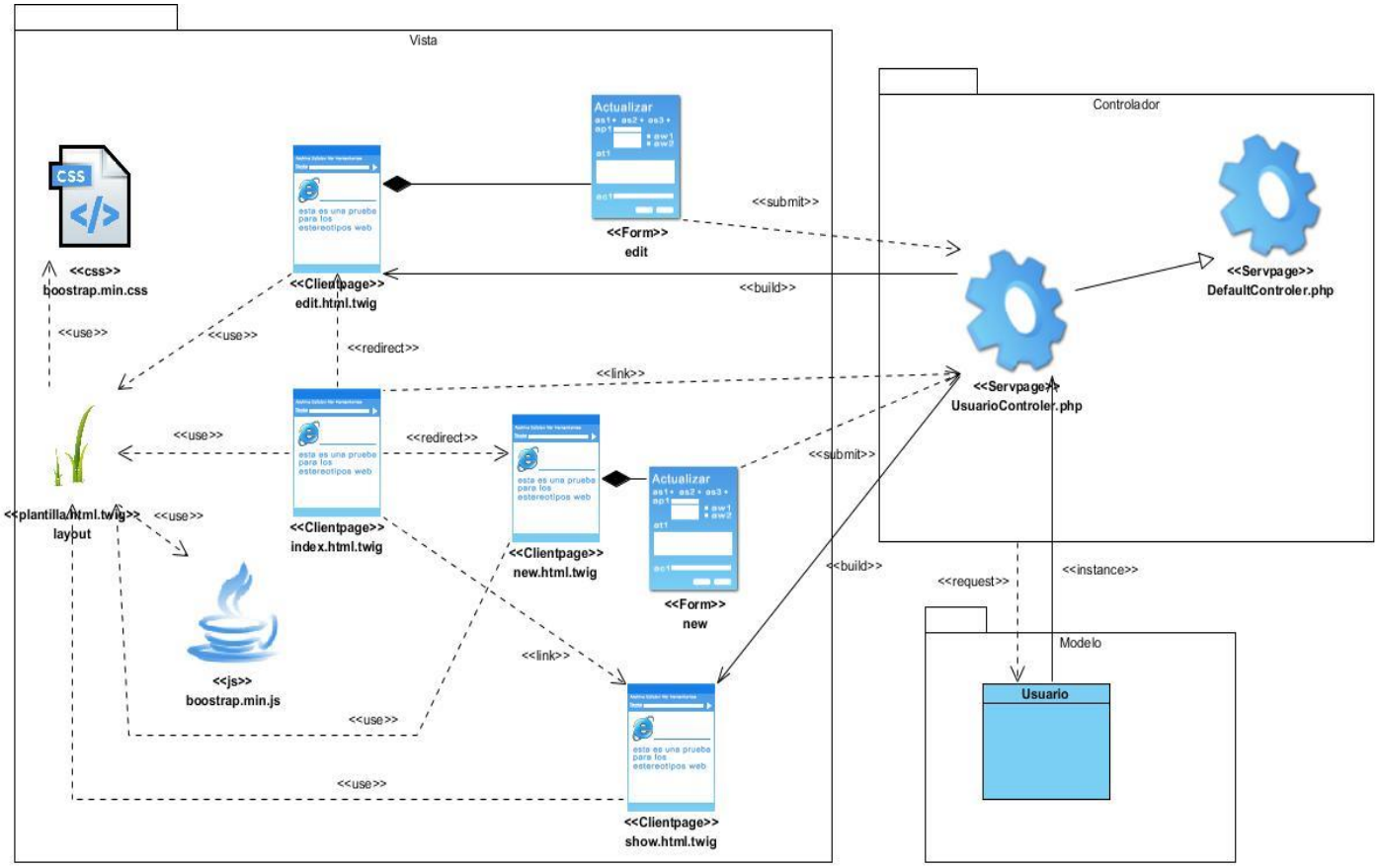


Ilustración 6. Diagrama de clases con estereotipos web. Administrar usuario.

2.11.2. Patrones de diseño

Una de las principales ventajas de utilizar un framework de desarrollo es que están basados en patrones de diseño, característica que hace posible la gran usabilidad que tienen, casi siempre independientes del tipo de aplicación web que se desee desarrollar (Pérez, Ramírez, González y Vargas, 2011).

El framework Symfony2 define una serie de patrones de diseño que se encuentran embebidos en su arquitectura concebidos para que cualquier desarrollador los use en pos de lograr un exitoso diseño en su sistema.

Los patrones **GRASP**[7], describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones (Larman, 2004). En la siguiente sección se explican los patrones de diseño utilizados directamente en la solución.

Experto: dentro de la arquitectura que presenta el framework Symfony2, este patrón se muestra muy evidenciado en la capa perteneciente al modelo de datos. En la misma se encuentran, las clases encargadas de interactuar directamente con la base de datos a través del ORM Doctrine, las cuales generalmente se encuentran bajo el nombre de name_entityRepository, presentando la responsabilidad de realizar directamente las acciones de consultas debido a que contienen los atributos necesarios para realizar dichas operaciones.

Creador: este patrón guía la asignación de responsabilidades relacionadas con la creación de objetos. El propósito fundamental de este patrón es encontrar un creador que debemos conectar con el objeto producido en cualquier evento (Larman, 2004).

Bajo Acoplamiento: este patrón plantea que la dependencia entre las clases que conforman la arquitectura del sistema debe ser mínima, debido a que a la hora de realizar modificaciones en una clase, no afecte la estructura de las restantes. En la aplicación se concibió mantener separadas las clases pertenecientes al modelo de datos, las vistas de usuario y las clases controladoras.

Alta Cohesión: este patrón plantea que las clases se deben apoyar del funcionamiento de otras, para lograr su objetivo; y no incluir en ella misma todo el proceso del negocio, ya que sería difícil de comprender, de reutilizar y le afectarían constantemente los cambios.

Controlador: el framework Symfony2 incluye un *bundle*[8], denominado FrameworkBundle, el cual presenta una serie de clases controladoras, encargadas de las principales operaciones dentro del funcionamiento de Symfony2. Además se crean aquellas clases controladoras que darán solución a los principales requisitos pertenecientes al flujo del negocio en cuestión, facilitando la centralización de las actividades.

Patrones conductuales

[7] Acrónimo que significa General Responsibility Assignment Software Patterns (Patrones Generales de Software para Asignar Responsabilidades).
[8] Filosofía de desarrollo a partir de Symfony2 de denominar a la estructura de un componente.

Strategy: este patrón plantea la utilización de varias estrategias para representar comportamientos distintos, utilizando una misma interfaz. Estos comportamientos pueden ser modificados e intercambiados en la aplicación, sin que afecte la lógica del sistema.

Command: este patrón se puede observar en el sistema de routing de la aplicación, el cual parsea las direcciones URL[9] con el objetivo de precisar los parámetros de la misma y de esta forma informar qué acción debe ejecutarse para responder a la petición. Este comando brinda la posibilidad de delegar a los métodos **Actions** la comprobación de los parámetros de la petición.

Patrones estructurales

Decorator: este patrón plantea la utilización de una o varias plantillas globales, que guarden el código que es usual para todo el sistema, para no tener que repetirlo en cada interfaz.

[9] Sigla en inglés de Uniform Resource Locator (Localizador de Recursos Uniformes).

Ejemplos gráficos del uso de patrones

El patrón **Creador** se evidencia en la acción createAction() del archivo UsuarioController.php cuando se crea una instancia de la entidad Usuario para encapsular los datos que inserta el usuario e insertarlos en la base de datos.

```php
/**
 * Creates a new Usuario entity.
 *
 */
public function createAction(Request $request)
{
    $entity = new Usuario();
    $form = $this->createCreateForm($entity);
    $form->handleRequest($request);

    if ($form->isValid()) {
        $em = $this->getDoctrine()->getManager();

        $entity->setSalt(md5(time()));
        $encoder = $this->get('security.encoder_factory')->getEncoder($entity);
        $passwordCodificado = $encoder->encodePassword(
            $entity->getPassword(), $entity->getSalt()
        );
        $entity->setPassword($passwordCodificado);

        $em->persist($entity);
        $em->flush();

        return $this->redirect($this->generateUrl('usuario_show', array('id' => $entity->getId())));
    }

    return $this->render('MainBundle:Usuario:new.html.twig', array(
        'entity' => $entity,
        'form' => $form->createView(),
    ));
}
```

Ilustración 7 Patrón creador.

El patrón **Alta Cohesión** ejemplo de ello dentro de la aplicación, es a la hora de editar un usuario, la acción editAction() utiliza la colaboración de otras clases, tanto del modelo como de la vista, y además hace llamada a otra acción createEditForm(), con el objetivo de delegar responsabilidades.

```php
public function editAction($id) {
    $em = $this->getDoctrine()->getManager();

    $entity = $em->getRepository('MainBundle:Usuario')->find($id);

    if (!$entity) {
        throw $this->createNotFoundException('Unable to find Usuario entity.');
    }

    $editForm = $this->createEditForm($entity);
    $deleteForm = $this->createDeleteForm($id);

    return $this->render('MainBundle:Usuario:edit.html.twig', array(
            'entity' => $entity,
            'edit_form' => $editForm->createView(),
            'delete_form' => $deleteForm->createView(),
    ));
}

/**
private function createEditForm(Usuario $entity){
    $form = $this->createForm(new UsuarioType(), $entity, array(
        'action' => $this->generateUrl('usuario_update', array('id' => $entity->getId())),
        'method' => 'PUT',
    ));

      $form->add('submit', 'submit', array('label' => 'Update'));

    return $form;
}
```

Ilustración 8 Patrón Alta Cohesión.

El uso del patrón **Decorator**, en este caso se representan en las plantillas login.html.twig y layout.html.twig, las cuales establecen el maquetado para todas las plantillas del sistema.

```
{% extends "MainBundle::layout.html.twig" %}

{% block title %}
    Login
{% endblock %}

{% block id 'login' %}

{% block body %}
    <div class="container">
        <form class="form-signin" action="{{ path('usuario_login_check') }}" method="post" role="form">
            {% if error %}
```

Ilustración 9 Patrón Decorator. login.html.twig

```
{% extends "::base.html.twig" %}

{% block stylesheets %}
    <link href="{{ asset('bundles/main/css/bootstrap.min.css') }}" rel="stylesheet">
    <link href="{{ asset('bundles/main/css/signin.css') }}" rel="stylesheet">
    <link href="{{ asset('bundles/main/css/custom.css') }}" rel="stylesheet">
    <link href="{{ asset('bundles/main/css/font-awesome.min.css') }}" rel="stylesheet">
{% endblock %}
```

Ilustración 10 Patrón Decorator. layout.html.twig

2.12. Conclusiones parciales del capítulo

- A partir de la descripción de la propuesta de solución, se pudieron documentar sus principales características fundamentadas en la necesidad de realizar la administración centralizada de manera asistida por una interfaz gráfica permitiendo la asignación simultánea de tareas de configuración a un conjunto de máquinas de un área específica, teniendo en cuenta además las ventajas de los principios del desarrollo web.
- El progreso en la construcción de la propuesta se evidenció a partir de los artefactos documentados en este capítulo. Algunos de ellos no están contemplados en la metodología XP, pero como se expresara anteriormente se realizaron para registrar con mayor detalle los elementos de la solución quedando descritos de esta manera los conceptos del dominio, los requisitos funcionales y no funcionales, las historias de usuario, el plan de iteraciones, las características gráficas a través de un prototipo de interfaz de usuario, las responsabilidades de las clases y el modelo de datos.
- Además se explicaron a través de ejemplos los patrones de diseño empleados facilitando una mayor comprensión de la arquitectura y el diseño de la solución per se.

Capítulo 3: Implementación y pruebas

Introducción

En el presente capítulo se ilustra la implementación a través de la descripción de la solución por capas y del estándar de codificación. Además se determinan los tipos de pruebas a realizar y los casos de pruebas que serán aplicados al sistema. Asimismo se incluye el epígrafe de validación de la investigación.

3.1. Capas del sistema

Clases pertenecientes a la capa *Controlador*.

Nombre	Descripción
DefaultController.php	Esta clase realiza varias funcionalidades en el sistema, entre las que se encuentran: Autenticar, Editar perfil y Visualizar perfil del usuario. La misma mantiene una estrecha relación con otras clases del sistema, como son: **Usuario**, **UsuarioType** y **Maquina**.
UsuarioCotroller.php	Esta clase se encarga de gestionar los usuarios en el sistema, permitiendo crear, listar, editar y eliminar los mismos. La misma mantiene una relación con las clases **Usuario** y **UsuarioType**.
EventoController.php	Esta clase se encarga de gestionar los comandos en el sistema, permitiendo crear, listar, editar, ejecutar y eliminar los mismos. La misma mantiene una relación con las clases **Evento** y **EventoType**.
AreaController.php	Esta clase se encarga de gestionar las áreas en el sistema, permitiendo crear, listar, editar y eliminar las mismas. Además permite adicionar las máquinas a las áreas. La misma mantiene una relación con las clases ***Area*** y **AreaType**.

Tabla 7 Descripción de clases en la capa Controlador.

Plantillas de la capa *Vista*.

	Nombre	Descripción
Usuario	index.html.twig	Estas plantillas se encargan de generar la parte visual

	new.html.twig	que corresponde a la Sección de Usuario.
	edit.html.twig	
	show.html.twig	
Área	index.html.twig	Estas plantillas se encargan de generar la parte visual que corresponde a la sección de Área.
	new.html.twig	
	edit.html.twig	
	show.html.twig	
Comando	index.html.twig	Estas plantillas se encargan de generar la parte visual que corresponde a la sección de Comando.
	new.html.twig	
	edit.html.twig	
	show.html.twig	
	execute.html.twig	
Computadoras	index.html.twig	Esta plantilla se encarga de generar la parte visual que corresponde a la sección de Computadoras.

Tabla 8 Descripción de clases en la capa Vista.

Entidades de la capa *Modelo*

Entidad	Descripción
Usuario	Esta entidad contiene toda la información de los usuarios y estará asociada al único rol de administrador.
Área	Esta entidad relaciona la información de las áreas físicas que contienen las computadoras conectadas en una red de área local.
Comando	Esta entidad está asociada a los comandos empleados para la configuración y administración de computadoras de forma remota y contendrá la instrucción a ejecutar.
Computadoras	Esta entidad es la equivalente a las estaciones de trabajo que contendrán una llave y un estado.

Tabla 9 Descripción de clases en la capa Modelo.

En estas tres capas que responden al patrón arquitectónico Modelo-Vista-Controlador descrito en el capítulo anterior, se reflejan los elementos indispensables para la construcción de la solución; por lo que es posible proceder a la implementación.

3.2. Implementación

En esta fase se implementan las HU definidas anteriormente, como parte del cumplimiento de los requisitos del sistema. Al principio de esta actividad se realiza una revisión del plan de iteraciones y se modifica en caso de ser necesario. Mediante este plan se descomponen las HU en tareas de desarrollo y se asignan responsables de implementación a cada una de las tareas.

3.2.1. Estándar de codificación.

Para desarrollar la solución es aconsejable emplear un estándar de codificación para lograr una mejor organización del código, evitar redundancias resultantes de la programación en pareja y garantizar la homogeneidad en el código. Ante tales beneficios, se decidió emplear un estándar que facilitara el trabajo concurrente sin afectar la integración y consecuente reutilización de segmentos de código entre *bundles*[10]. Por lo que a partir de las recomendaciones que ofrece el sitio oficial de Symfony2 en español, con respecto a la implementación en este marco de trabajo, el desarrollo de la propuesta de solución seguirá las pautas establecidas en el estándar de codificación para Symfony2 que se describen a continuación.

Estructura

- No se usan las etiquetas cortas (<?).
- No se finalizan las clases con la etiqueta usual de cierre (?>).
- La indentación se debe realizar utilizando cuatro espacios (las tabulaciones no están permitidas).
- Se utiliza el caracter de salto de línea (0x0A) para finalizar las líneas.
- Se agrega un único espacio después de cada delimitador coma.
- No se ponen espacios después de la apertura de un paréntesis y antes del cierre del mismo.
- Se agrega un único espacio alrededor de operadores (==, *&&*).
- Se agrega un único espacio antes de los paréntesis de apertura de una palabra clave de control (*if*, *else*, *for*, *while*).
- Se agrega una línea en blanco antes de la sentencia *return*.
- No se agregan espacios al final de las líneas.

[10] Bundles: componentes o módulos independientes de Symfony2.

- Se utilizan llaves para indicar el cuerpo de las estructuras de control sin importar el número de sentencias que éstas contengan.
- Se colocan las llaves en sus propias líneas para clases, métodos y declaración de funciones.
- Se separan las sentencias condicionales y las llaves de apertura con un único espacio sin dejar una línea en blanco.
- Se declaran explícitamente la visibilidad de clases, métodos y propiedades (no se debe usar *var*).
- Se utilizan constantes de tipo *PHP* nativas en minúsculas: false, true y null. Lo mismo aplica para *array ()*.
- Se utilizan letras mayúsculas para constantes, con palabras separadas por guiones bajos.
- Se define una clase por archivo.
- Se declaran las propiedades de las clases antes de los métodos.
- Se declaran los métodos públicos primero, luego los protegidos y finalmente los privados.

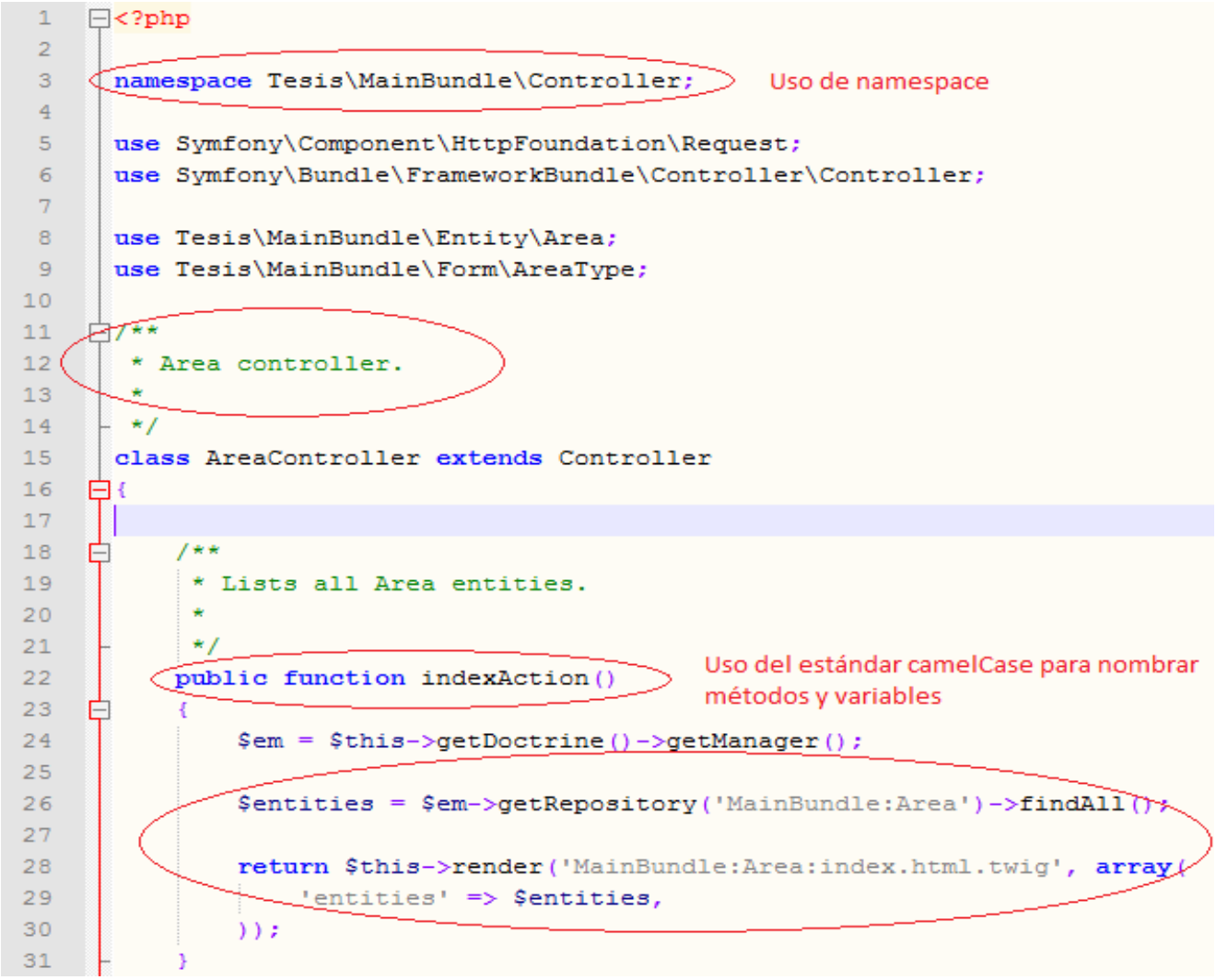
Convención de Nombres

- Se utiliza el estilo de escritura lowerCamelCase y no guiones bajos, para variables, funciones y nombres de métodos.
- Se utilizan guiones bajos para definir opciones, argumentos y nombres de parámetros.
- Se utilizan los namespace para todas las clases.
- Se utiliza *Symfony* como el namespace de primer nivel.
- Se añade como sufijo *Interface* a las interfaces.
- Se utilizan caracteres alfanuméricos y guiones bajos para nombres de archivos.

Documentación

- Se agregan los bloques *PHPDoc* para todas las clases, métodos y funciones.
- Las anotaciones *@package* y *@subpackage* no son utilizadas.

En la siguiente ilustración se muestra una evidencia de la utilización de los estándares de codificación propuestos por symfony2 para el desarrollo del sistema:




```php
<?php

namespace Tesis\MainBundle\Controller;

use Symfony\Component\HttpFoundation\Request;
use Symfony\Bundle\FrameworkBundle\Controller\Controller;

use Tesis\MainBundle\Entity\Area;
use Tesis\MainBundle\Form\AreaType;

/**
 * Area controller.
 *
 */
class AreaController extends Controller
{

    /**
     * Lists all Area entities.
     *
     */
    public function indexAction()
    {
        $em = $this->getDoctrine()->getManager();

        $entities = $em->getRepository('MainBundle:Area')->findAll();

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

Ilustración 11 Estándar de codificación.

3.3. Métricas

Las métricas son una medida cuantitativa que permiten lograr una visión profunda de la eficacia de la construcción del software. Ayudan en la planificación, seguimiento y control de un proyecto de software y evalúan la calidad del producto.

3.3.1. Métrica para requisitos

Estabilidad de los requisitos

El objetivo de esta métrica es medir la estabilidad de los requisitos para asegurar su adecuación antes de pasar al próximo flujo de trabajo. Se considera que los requisitos son estables cuando no existen adiciones o supresiones en ellos que impliquen modificaciones en las funcionalidades principales de la aplicación.

$$ETR = \left[\frac{RT - RM}{RT}\right] \times 100$$

Donde:

- ETR: valor de la estabilidad de los requisitos.
- RT: total de requisitos definidos.
- RM: número de requisitos modificados, que se obtienen como la sumatoria de los requisitos insertados, modificados y eliminados.

Esta métrica ofrece valores entre 0 y 100. El mejor valor de ETR es el más cercano a 100 porque mostrará que los requisitos son estables y por tanto es confiable trabajar el análisis y diseño sobre ellos.

Al ejecutar la fórmula teniendo en cuenta el total de requisitos definidos y los que habían sido modificados hasta la primera iteración de la validación, se tiene que RT = 32 y RM = 4, por lo que ETR = 87.5%. De ese resultado puede inferirse que al estar cercano a 100, la estabilidad de los requisitos es bastante alta demostrando que solo 1/8 de los mismos sufrió alguna variación.

Luego de ejecutar una segunda iteración de validación asumiendo como valores iniciales a los obtenidos después de la primera iteración, se tiene que RT = 32 y RM = 0, por lo que ETR = 100 %. Después de lograr ese resultado puede concluirse que luego del segundo ajuste con el cliente, los requisitos se mantuvieron estables y fueron los que se desarrollaron finalmente en la aplicación.

Especificidad de los requisitos

Por su parte, la métrica de especificidad de requisitos tiene como objetivo cuantificar la especificidad o la ausencia de ambigüedad en la definición de los requisitos. Para calcular esta métrica deben contarse los

requisitos que tuvieron igual interpretación por los revisores y compararlos con el total de requisitos definidos.

La misma se calcula como:

$$ER = {}^{n_{ui}}/_{n_r}$$

Donde:

- ER: grado de especificidad de los requisitos.
- nui: número de requisitos para los que todos los revisores tuvieron interpretaciones idénticas.
- nr: cantidad de requisitos en una especificación y comprende la suma de los requisitos funcionales y no funcionales.

El valor de esta métrica debe estar siempre entre 0 y 1. Mientras más cerca de 1 esté el valor de ER mayor será la consistencia de la interpretación de los revisores para cada requisito y menor será la ambigüedad en la especificación de los requisitos.

Para una primera iteración se obtuvieron los siguientes resultados:

ER = 0.65	nui = 21	nr = 32

Para una segunda iteración se arrojaron los siguientes resultados:

ER = 0.84	nui = 27	nr = 32

Para una tercera iteración se arrojaron los siguientes resultados:

ER = 1	nui = 32	nr = 32

A partir de los resultados de la tercera iteración donde ER = 1 se puede concluir que los requisitos tienen un alto grado de legibilidad en su especificación, eliminando cualquier incongruencia, ambigüedad o incoherencia que pudieran haberse detectado en las iteraciones anteriores.

Grado de validación de los requisitos

Por último, se deben poder validar de forma general los requisitos mediante un consenso del equipo de desarrollo al contrastar lo que desea el cliente con la posibilidad real de implementarlo. El grado de validación de los requisitos mide la corrección en la definición de los mismos.

Para ello se tiene la siguiente fórmula: $VR = {}^{n_c}/_{(n_c + n_{nv})}$

Donde:

- VR: grado de validación de los requisitos.
- nc: número de requisitos que se han validado como correctos.
- nnv: número de requisitos no validados aún.

El resultado de esta métrica está siempre entre 0 y 1. El valor óptimo de esta métrica es el más cercano a 1 e indica un alto nivel de corrección en la definición de los requisitos.

En la primera iteración se obtuvieron los siguientes resultados:

VR = 0.36	nc = 7	nnv = 12

En la segunda iteración se obtuvieron los siguientes resultados:

VR = 0.72	nc = 13	nnv = 5

En la tercera iteración se obtuvieron los siguientes resultados:

VR = 1	nc = 18	nnv = 0

Luego de haber realizado tres iteraciones de validación, se pudo obtener el valor máximo de corrección en los requisitos entendiéndose como una definición libre de ambigüedades e incoherencias, un alto grado de especificación de los requisitos y una gran estabilidad.

3.3.2. Métricas para el diseño

Las métricas para el diseño tienen varias aplicaciones, pero en este caso se empleará la relacionada con el tamaño operacional de las clases (TOC), centrada en el recuento de atributos y operaciones para cada clase individual. El tamaño general de una clase puede medirse determinando:

- El total de operaciones (heredadas y privadas de la instancia), que se encapsulan dentro de la clase.
- El número de atributos (heredados y privados de la instancia), encapsulados por la clase.

Estos dos valores son sumados de acuerdo a la clase que se analiza y los resultados son tomados como cantidad de procedimientos (CP) que luego son comparados para determinar el TOC. En la *Tabla 10 Clasificación de umbral de valores según Lorenz y Kidd*, pueden evidenciarse los umbrales de valores de TOC que fueron establecidos por Mark Lorenz y Jeff Kidd en su libro *Object Oriented Software Metrics.*

Clasificación	Valores
Pequeño	CP <= 20
Medio	CP > 20 y <= 30
Grande	CP > 30

Tabla 10 Clasificación de umbral de valores según Lorenz y Kidd[11]

En la *Tabla 11 TOC de las clases del sistema*, se ilustra el cálculo del TOC de las clases como se explicara anteriormente.

Clases	TOC
Usuario.php	CP= 11 operaciones + 4 atributos = 15
Area.php	CP= 9 operaciones + 4 atributos = 13
Evento.php	CP= 7 operaciones + 4 atributos = 11
Maquinas.php	CP= 8 operaciones + 4 atributos = 12

Tabla 11 TOC de las clases del sistema

Luego puede inferirse que todas las clases medidas poseen un TOC pequeño ya que ninguna sobrepasa las 20 operaciones. Ello significa que no poseen un alto grado de responsabilidad, aumentando por tanto la reutilización de las mismas, simplificando su implementación y la realización de pruebas. Todo ello puede corroborarse con la medición de los atributos de calidad Responsabilidad, Complejidad de Implementación y Reutilización de acuerdo a como aparecen en la *Tabla 12 Rango de valores por atributos de calidad de la métrica TOC.*

Es preciso destacar que el TOC es directamente proporcional al grado de responsabilidad de una clase, siendo este último inversamente proporcional a la reutilización y la complejidad de implementación.

Atributo	**Categoría**	**Criterio**
Responsabilidad	Baja	CP <=Promedio
	Media	Promedio<= CP <=2*Promedio

[11] Autores de la métrica de diseño TOC.

	Alta	CP >2*Promedio
Complejidad de Implementación	Baja	CP <= Promedio
	Media	Promedio<= CP <=2*Promedio
	Alta	CP >2*Promedio
Reutilización	Baja	CP >2*Promedio
	Media	Promedio<= CP <=2*Promedio
	Alta	CP <= Promedio

Tabla 12 Rango de valores por atributos de calidad de la métrica TOC

Después de medidos los anteriormente mencionados atributos de acuerdo a los umbrales de la tabla 12, se confirmó el planteamiento relacionado con el TOC y a continuación se ilustran de manera individual en la *Tabla 13 Valores generales de TOC por clases*.

Clases	**Valores de TOC**	**Responsabilidad**	**Complejidad**	**Reutilización**
Usuario.php	15	Media	Media	Media
Area.php	13	Media	Media	Media
Evento.php	11	Baja	Baja	Alta
Maquinas.php	12	Baja	Baja	Alta

Tabla 13 Valores generales de TOC por clases

3.4. Pruebas de software

Después de validado el diseño, se debe comprobar si la implementación está en igual correspondencia, por tanto se deben aplicar pruebas de software. Particularmente puede entenderse a una prueba de software como el proceso mediante el cual se analiza un producto, componente o segmento de código para detectar errores o incongruencias entre lo que está desarrollado y las restricciones establecidas por el cliente. Además se realizan para evaluar las características del software en cuestión, por lo que la actividad de pruebas debería ser realizada durante todo el proceso de desarrollo de software. (Laurie Williams, 2006)

Las pruebas de software son también unas de las prácticas de verificación y validación, en unión con las inspecciones y la programación en pareja. La verificación específicamente, es el proceso de evaluar un sistema o componente para determinar si los productos de una determinada fase de desarrollo satisfacen las condiciones impuestas al inicio de la misma y responde a siguiente la interrogante: ¿Se está construyendo el producto correctamente?

Por su parte, la validación es el proceso de evaluar un sistema o componente a lo largo del proceso de desarrollo de software o cuando este termina, para comprobar si se satisfacen los requisitos del cliente y son rastreables en sentido inverso. Dichas prácticas responden a la pregunta siguiente: ¿Se está construyendo el producto apropiado?

A continuación, se ilustran los niveles de prueba, los métodos, técnicas y artefactos según aplique en cada caso.

Nivel de prueba	Alcance general	Métodos de prueba	Rol involucrado
Unidad	Unidades de código	Caja blanca	Programador
Integración	Múltiples clases	Caja blanca Caja negra	Programador
Funcional	Todo el producto	Caja negra	Probador independiente
Sistema	Todo el producto en un entorno	Caja negra	Probador independiente
Aceptación	Todo el producto en el entorno del cliente	Caja negra	Cliente
Beta	Todo el producto en el entorno del cliente	Caja negra	Cliente
Regresión	Todos los escenarios anteriores	Caja blanca Caja negra	Programador probador

Tabla 14 Niveles de prueba de software.

En el caso específico de la presente investigación se emplean dos métodos de pruebas de software: caja blanca y caja negra. Además se practican pruebas automatizadas al software. En el caso del método de caja blanca, se emplea la técnica de camino básico con su respectivo artefacto de Caso de prueba. De modo similar, el método de caja negra emplea la técnica de partición equivalente o clases de equivalencia, acompañándola con el artefacto correspondiente: Caso de prueba.

3.4.1. Método de prueba de Caja negra

Técnica de partición de equivalencia

Esta técnica consiste en diseñar un Caso de prueba por funcionalidad creando valores válidos e inválidos para cada uno de los campos de la interfaz de usuario, de manera que se compruebe su validación y correspondencia con el comportamiento que debería tener el sistema como respuesta automática a la interacción con el usuario (Laurie Williams, 2006).

Para evidenciar esta prueba, se seleccionó como Caso de estudio al requisito funcional Autenticar usuario, para el cual se construyó el Caso de prueba que se describe a continuación. Los restantes artefactos de este tipo se encuentran reflejados en el Anexo 3: *Diseño de casos de prueba*

CP1_Autenticar usuario

Escenario	Descripción	Nombre de usuario	Contraseña	Respuesta del sistema	Flujo central
EC 1.1 Autenticar usuario de forma correcta.	El sistema autentica al usuario de forma correcta.	V prueba	V prueba1234	El sistema autentica al usuario y le permite el acceso a las funcionalidades del sistema, según su rol.	1-El usuario teclea en el navegador la url del sistema seguido de /login 2-El sistema le muestra un formulario para su autenticación. 3-El usuario introduce la información y selecciona la opción Entrar.
EC 1.2 Autenticar usuario de forma incorrecta.	El sistema no autentica al usuario.	I 4r	V prueba1234	El sistema no autentica al usuario y emite los siguientes mensajes de errores: Lo sentimos. No reconocemos el nombre de usuario o la contraseña.	
EC 1.3	El sistema no	I	I	El sistema no	

Autenticar usuario dejando campos vacíos.	autentica al usuario dejando campos obligatorios vacíos.	Vacío	Vacío	autentica al usuario y emite los siguientes mensajes de errores: "El campo Usuario es obligatorio". "El campo Contraseña es obligatorio".	

Tabla 15 Caso de prueba del RF Autenticar usuario en el sistema.

Se puede concluir que la implementación del RF Autenticar usuario en el sistema cumple con las restricciones de los campos que posee su interfaz y permite que el sistema responda como se espera.

A pesar de tener la certeza de que las funcionalidades responden como se espera desde sus interfaces gráficas, es necesario saber si la codificación se realizó con la cantidad necesaria de sentencias sin tener redundancia, líneas inutilizadas y métodos incompletos. Para ello se aplica el método de Caja blanca con su técnica de camino básico.

3.4.2. Método de prueba de Caja blanca

Técnica de camino básico

Esta técnica consiste en diseñar pruebas que fuercen el recorrido de esos caminos, para garantizar que se ejecute al menos una vez cada sentencia del programa y que cada condición se ejecute en sus variantes verdadera y falsa. Se debe tener en cuenta que de un mismo diseño procedimental se pueden derivar varios conjuntos básicos. (Ma Isabel, 2001)

Se procede entonces a construir un grafo donde cada nodo indica la secuencia en que se ejecutan las sentencias de código de un método específico incluyendo las bifurcaciones que simbolizan el tratamiento de las condiciones y sus flujos básicos y alternos.

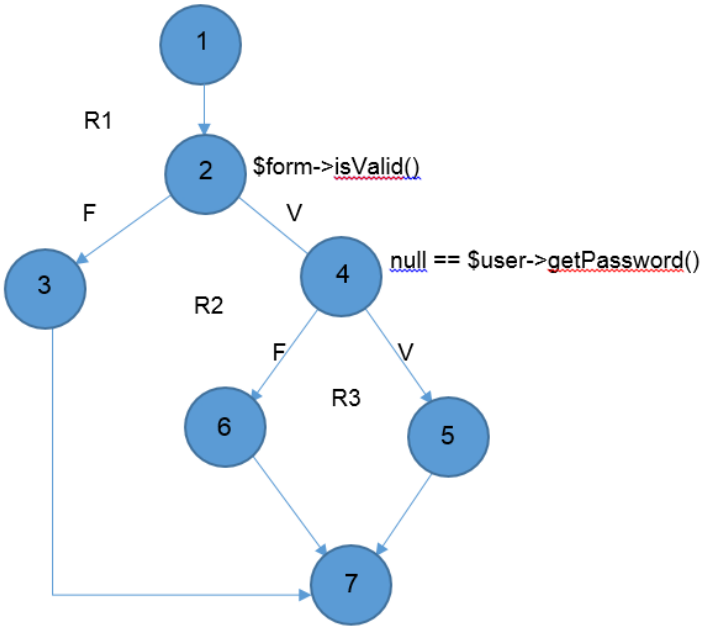


Ilustración 12 Grafo del camino básico del método perfiAction().

A partir de este grafo se calcula la complejidad ciclomática que es el número de caminos independientes del conjunto básico de un programa y nos da un límite superior para el número de pruebas que se deben realizar para asegurar que se ejecute cada sentencia al menos una vez. Se tienen entonces 3 caminos básicos:

Camino 1: 1-2-3-7
Camino 2: 1-2-4-5-7
Camino 3: 1-2-4-6-7

Luego de tener elaborado el Grafo de Flujo y los caminos a recorrer, se preparan los casos de prueba que forzarán la ejecución de cada uno de esos caminos. Se escogen los datos de forma que las condiciones de los nodos predicados estén adecuadamente establecidas, con el fin de comprobar cada camino.

```php
public function perfilAction(Request $request) {
    $em = $this->getDoctrine()->getManager();

    $user = $this->getUser();

    $form = $this->createForm(new UsuarioType(), $user);

    $originalPassword = $form->getData()->getPassword();

    $form->handleRequest($request);

    if ($form->isValid()) {
        if (null == $user->getPassword()) {
            $user->setPassword($originalPassword);
        } else {
            $encoder = $this->get('security.encoder_factory')->getEncoder($user);
            $codifiedPassword = $encoder->encodePassword(
                    $user->getPassword(), $user->getSalt()
            );
            $user->setPassword($codifiedPassword);
        }
        $em->persist($user);
        $em->flush();

        $this->get('session')->getFlashBag()->add('success', 'Perfil actualizado');
        return $this->redirect($this->generateUrl('usuario_perfil'));
    } else {
        $this->get('session')->getFlashBag()->add('warning', 'Revise los datos');
    }
    return $this->render('MainBundle:Default:perfil.html.twig', array(
                'usuario' => $user,
                'form' => $form->createView(),
    ));
}
```

Ilustración 13 Método perfiAction()

Luego de confeccionar los casos de prueba, se ejecutan cada uno de estos y se comparan los resultados con los esperados. Una vez terminados todos los casos de prueba, se estará seguro de que todas las sentencias del programa se han ejecutado por lo menos una vez. Es importante considerar que algunos caminos no se pueden probar de forma aislada. O sea, la combinación de datos requeridos para recorrer el camino no se puede obtener con el flujo normal del programa. En tales casos, estos caminos se prueban como parte de otra prueba de camino.

Caso de prueba: Camino básico # 1	
Entrada	Realizar la petición mediante el método perfilAction (Request $request), obtener el usuario autenticado en la aplicación mediante el atajo $this->getUser()
Resultados esperados	Se cargan los datos del perfil que se encuentra autenticado en ese momento y se construye un formulario con los datos del mismo.
Condiciones	El método de la petición tiene que ser GET.

Tabla 16 Caso de prueba: Camino básico # 1.

Caso de prueba: Camino básico # 2	
Entrada	Se debe dejar el campo contraseña en blanco.
Resultados esperados	Se actualizan los datos del usuario autenticado en el sistema; el campo contraseña se mantiene igual.
Condiciones	Se debe cumplir la condición de que el formulario sea válido (que sea válido significa que el formulario fue enviado mediante el método post y que los datos enviados son correctos), después se realiza la siguiente comprobación null == $user->getPassword() que verifica que cuando se editó el formulario el usuario no cambió el password y por tanto le asigna el mismo password que tenía.

Tabla 17 Caso de prueba: Camino básico # 2.

Caso de prueba: Camino básico # 3	
Entrada	Se debe editar el campo contraseña.
Resultados esperados	Se actualizan los datos del usuario autenticado en el sistema; incluyendo el campo contraseña.
Condiciones	Se debe cumplir la condición de que el formulario sea válido (que sea válido significa que el formulario fue enviado mediante el método post y que los datos enviados son correctos), después se realiza la siguiente comprobación null == $user->getPassword() que sería falsa por tanto codifica la nueva contraseña y persiste los datos es decir los guarda.

Tabla 18 Caso de prueba: Camino básico # 3.

Después de aplicados los casos de prueba de la técnica de camino básico se pudo constatar que todas las sentencias se ejecutan al menos una vez y que no existe redundancia de código, líneas en desuso o métodos incompletos.

Pruebas automatizadas al software

Las pruebas automatizadas se ejecutan con acceso al código fuente (datos y lógica). Se trabaja con entradas, salidas y el conocimiento interno. Son pruebas unitarias que se usan cuando se conoce la estructura interna y el funcionamiento del código a probar. (Pérez, 2008)

Pruebas unitarias

Este tipo de prueba constituye la piedra angular de la metodología XP, debido a que todos los módulos deben pasar las pruebas unitarias antes de ser liberados o publicados. Estas pruebas deben ser definidas antes de realizar el código, y se deben guardar junto a este en caso de futuras correcciones y actualizaciones. Cuando se encuentren errores, estos deben ser corregidos inmediatamente y se deben definir nuevas pruebas para verificar que el error haya sido resuelto (Joskowics, 2008). Las mismas se aplicaron a las entidades **Usuario y Comando.**

Para realizar las pruebas unitarias se utilizó el marco de trabajo orientado a pruebas PHPUnit. Este ayuda a probar el código y su objetivo es crear pequeñas unidades que revisen funcionalidades puntuales del código y probar que funcionen como debe, además de la posibilidad de automatizar estas pruebas para ejecutarlas frecuentemente, tanto como el código cambie. (Bergmann, 2014)

Los resultados obtenidos a partir de las pruebas unitarias generadas se muestran en el Anexo 4: *Pruebas unitarias.*

Pruebas de aceptación

Estas pruebas son creadas en base a las HU definidas, y son un equivalente a las pruebas de caja negra definidas bajo la metodología RUP. El cliente debe definir diversos escenarios para comprobar que una HU ha sido correctamente implementada. Son los clientes los responsables de verificar que los resultados de estas pruebas sean correctos. Se hace necesario publicar los resultados de las pruebas de aceptación, para que todo el equipo de desarrollo esté al tanto de las mismas (Joskowics, 2008).

Resultados obtenidos

Las pruebas de aceptación se realizaron en cuatro iteraciones, con el objetivo de verificar que las funcionalidades creadas en cada entrega del software, arrojaran el resultado esperado. En las iteraciones se obtuvieron los siguientes resultados:

Primera iteración: 15 No Conformidades (NC), de ellas 8 significativas, 5 no significativas y 2 recomendaciones.

Segunda iteración: 7 NC, de ellas 4 significativas, 2 no significativas y 1 recomendación.

Tercera iteración: 5 NC, de ellas 3 significativas, 1 no significativa y 1 recomendaciones.

Cuarta iteración: 1 NC que no representó relevancia para el funcionamiento del sistema.

En sentido general se resolvieron todas las NC por parte del equipo de desarrollo, donde los resultados de dichas pruebas se representan en la siguiente gráfica:

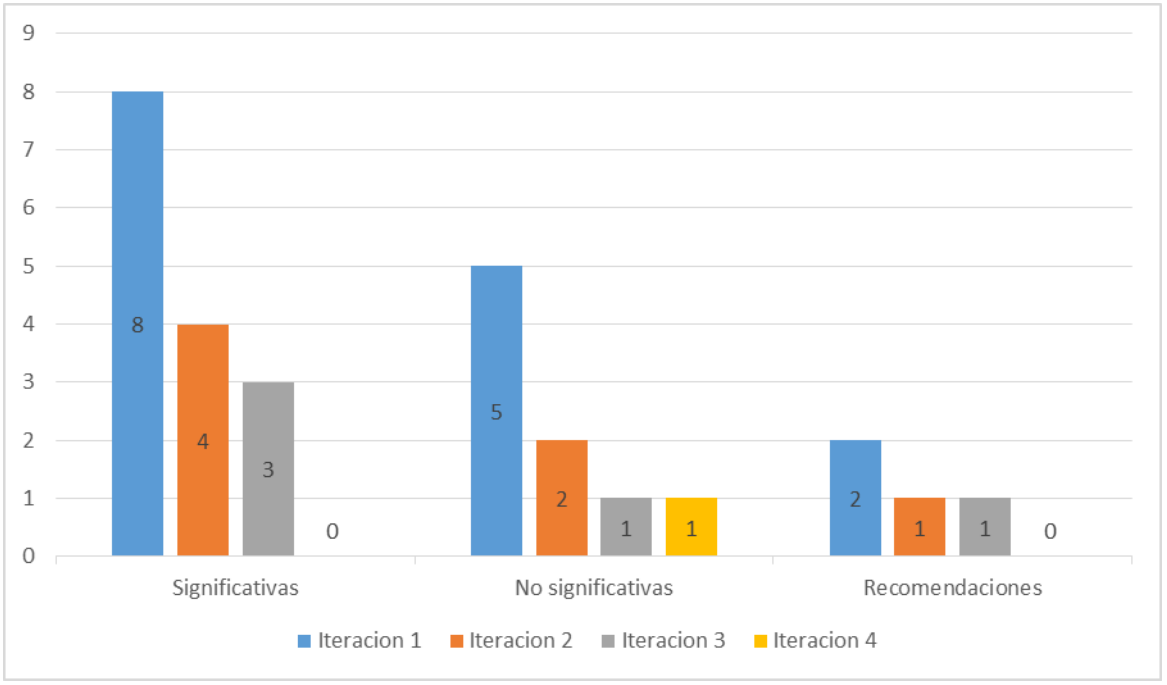


Ilustración 14 Resultados de las pruebas de aceptación.

El último paso de las pruebas de aceptación es firmar la Carta de aceptación por parte del cliente que valida que todos los resultados obtenidos en las iteraciones de prueba están en correspondencia con los objetivos iniciales y que satisfacen las expectativas del mismo.

3.5. Validación de la investigación

Para la validación de la investigación debe partirse de la variable dependiente, declarada como el tiempo de ejecución del proceso de administración centralizada de las estaciones de trabajo. Para poder

determinar si la propuesta de solución resuelve el problema planteado, se expresa el análisis en una ecuación matemática que permitirá calcular un aproximado del tiempo necesario para realizarlo de forma manual y a través de la herramienta SweetSalt.

Para ello se plantean las siguientes variables:

TE – tiempo de escritura del usuario sobre la consola. Se mide en segundos.

TC – tiempo de ejecución del comando. Se mide en segundos.

TS – tiempo de selección en la interfaz de las computadoras a configurar. Se mide en segundos

nTA – total de tareas de administración en relación a las estaciones de trabajo. Se mide en unidades enteras.

Tiempo de configuración manual: TCM	Tiempo de configuración en SweetSalt: TCS
$TCM = (TE + TC)^{*} nTA$	$TCS = TC + TS$

Donde TCM > TCS debido a la cantidad de veces (nTA) que deben escribirse las tareas de administración para diferentes computadoras conectadas en la red de área local.

En función de lograr una mejor representación se les darán respuesta a las preguntas de la investigación:

1. ¿Cuáles elementos de la problemática resultan críticos para los umbrales actuales de tiempo en la administración centralizada de computadoras a través de la consola de SaltStack?

En la forma manual desde la terminal de Linux se empleaba mucho tiempo por concepto de los desplazamientos de los especialistas para la configuración de cada computadora, consumiendo tiempo del horario de trabajo y así afectando la terminación de todas las tareas de configuración en el tiempo establecido. Además del tiempo que consumía la escritura de los bloques de comandos y tareas de configuración en la consola de SaltStack para cada estación de trabajo.

Con la introducción de SweetSalt ya no es necesario moverse de la oficina donde se va a instalar el Master de la solución, los comandos se ejecutan de manera predefinida por lo que no hay que teclearlos cada vez que se vaya a configurar o administrar una estación de trabajo. Es posible realizar la selección múltiple de computadoras sin necesidad de realizar las tareas de administración máquina por máquina.

Por lo tanto, este resultado contribuye a la disminución del tiempo de ejecución del proceso de administración centralizada de computadoras.

2. ¿Qué características funcionales son determinantes para disminuir el tiempo de ejecución del proceso de administración centralizada de computadoras?

Resulta determinante incluir algún medio de interpretación e interacción entre la aplicación y el cliente, entiéndase como una interfaz gráfica que permita simultaneidad en la configuración, asignación de áreas y computadoras, la creación de ítems para sustituir la escritura de los comandos en cada tarea de configuración por la ejecución de instrucciones a través de botones, donde el tiempo empleado sería solo el de seleccionar las computadoras en la interfaz más el tiempo de ejecución de los comandos en la consola.

SweetSalt integra todas esas características funcionales que resultan determinantes en la disminución del tiempo de ejecución del proceso de administración centralizada de computadoras, por lo que se puede concluir que la creación de la solución informática en cuestión, constituye una respuesta válida para el problema planteado en la investigación.

3.6. Conclusiones parciales

- El desarrollo de la propuesta de solución se realizó bajo el estándar de codificación propuesto para Symfony2, permitiendo un mayor aprovechamiento de sus beneficios, organización de las clases y el empleo del estilo de escritura lowerCamelCase.
- Con la aplicación de las métricas para requisitos y al diseño, se pudo constatar la calidad del producto, la ausencia de ambigüedades y la creación de las clases y operaciones necesarias para el funcionamiento eficaz de la solución.
- A través de las pruebas de caja blanca y las automatizadas con PHPUnit, se pudo revisar y corregir la presencia de segmentos de códigos redundantes, así como se evidenció el uso adecuado de las estructuras y clases.
- Finalmente, a partir de un análisis matemático se pudo comprobar que la propuesta de solución da respuesta al problema planteado, validando a SweetSalt como respuesta a las preguntas de la investigación.

Conclusiones generales

- La propuesta de solución se definió a partir de un estudio de las soluciones homólogas, incorporando las mejores prácticas de SaltStack.
- Después de la selección de las herramientas y la metodología de desarrollo de software se pudo desarrollar la solución con un enfoque ágil, pero potenciando su documentación.
- La aplicación de los patrones de diseño permitió construir una solución con una alta reutilización del código, bajo acoplamiento y alta cohesión entre clases.
- El desarrollo siguió el estándar de codificación propuesto para Symfony2, permitiendo un mayor aprovechamiento de sus beneficios, organización de las clases y el empleo del estilo de escritura lowerCamelCase.
- Con la aplicación de las métricas y pruebas al software se pudo constatar la calidad del producto, la ausencia de ambigüedades y segmentos de códigos redundantes, así como el uso adecuado de las estructuras y clases.
- Finalmente, a partir de un análisis matemático se pudo comprobar que la propuesta de solución da respuesta al problema planteado, validando a SweetSalt como respuesta a las preguntas de la investigación.

Referencias bibliográficas

1. LARMAN, Craig, 2004, UML y patrones: introducción al análisis y diseño orientado a objetos [online]. Félix Varela. [Accessed 9 April 2015]. Available from: http://books.google.com.cu/books?id=eNJjtwAACAAJ&dq=uml+y+patrones&hl=es&sa=X&ei=NNJVU47eDsTNsQTJ24D4BQ&redir_esc=y.
2. BERNARDO, Juan, ANAYA DE PÁEZ, Raquel, MARÍN, Juan Carlos and BILBAO LÓPEZ, Alex, 2005, Un estudio comparativo de herramientas para el modelado con UML. Revista Universidad EAFIT [online]. March 2005. Vol. 41, no. 137, p. 60 – 76. [Accessed 10 February 2015]. Available from: http://publicaciones.eafit.edu.co/index.php/revista-universidadafit/article/view/838/737/.
3. DARIAS, PEREZ. 2008. Darling, Análisis y Diseño de Componentes para Pruebas de Caja Blanca. UCI. Ciudad de La Habana, Cap. 1, pp.14, 15.
4. Williams, Laurie, 2007 A Survey of Agile Development Methodologies [online]**.**
5. JOSKOWICS, José, 2008, Reglas y Prácticas en eXtreme Programming. España: Universidad de Vigo. JQUERY COMMUNITY, 2013, jQuery 2.0 Released [online]. 18 May 2015. Available from: http://www.jquery.com.
6. COPYRIGHT © 2008-2015 - DEFINICION.DE. Definición de eficiencia, administración. *Definición.DE* [online]. [Accessed 7. febrero 2015]. Available from: http://definicion.de/ eficiencia/#ixzz3Z0057DPR, : http://definicion.de/administracion/#ixzz3QrZtFDIe
7. *SaltStack Walk-through* [online]. [Accessed 10. febrero 2015]. Available from: http://docs.saltstack.com/en/latest/topics/tutorials/walkthrough.html.
8. MAGAÑA ORÚE, Sara, 2009, Estudio comparativo de Lenguajes de Modelado de procesos de negocio para su integración en procesos de desarrollo de software dirigido por modelos. [online]. Proyecto de Fin de Carrera. Madrid, España: Universidad Carlos III de Madrid. [Accessed 10 February 2015]. Available from: http://earchivo.uc3m.es/bitstream/handle/10016/9077/PFC_SARA_MAGANA_ORUE.pdf?sequence=1.
9. HIDALGO URBINO, Rafael Jacobo, 2010, Análisis, diseño e implementación de una herramienta que permita la centralización de la información gestionada por el módulo Resultados de la Colección Multisaber. [online]. Trabajo de diploma para optar por el título de Ingeniero en Ciencias Informáticas.

La Habana, Cuba: Universidad de las Ciencias Informáticas. [Accessed 11 March 2015]. Available from: http://repositorio_institucional.uci.cu/jspui/bitstream/ident/TD_02857_10/1/TD_02857_10.pdf.

10. PRESSMAN, R.S. 2010. *Software Engineering: A Practitioner's Approach*. S.l.: McGraw-Hill. ISBN 9780073375977.
11. POTENCIER, Fabien, 2011, What is Symfony2? [online]. 25 November 2011. [Accessed 16 February 2015]. Available from: http://www.fabien.potencier.org/article/49/what-is symfony2.
12. CRUZ CASTRO, Julio, 2011, Sistema de Gestión de Calidad del Grupo de Calidad de FORTES. Módulo de Capacitación. [online]. Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas. La Habana, Cuba: Universidad de las Ciencias Informáticas. [Accessed 11 March 2015]. Available from: http://repositorio_institucional.uci.cu/jspui/bitstream/ident/TD_04629_11/1/TD_04629_11.pdf/.
13. SANTIESTEBAN PÉREZ, I., PÉREZ, I.I.S., RAMÍREZ, M.M., GONZÁLEZ, J.L.P. y VARGAS, Y.G. 2011. Desarrollo de funcionalidades que faciliten al docente su preparación y el control del aprendizaje de los estudiantes en la plataforma educativa Zera. *Serie Científica* [online], vol. 4, no. 11. Available from: http://publicaciones.uci.cu/index.php/SC/article/view/765.
14. Koontz, Harold; Weihrich, Heinz, 2011Principles of Management, [online].
15. MIR HUGUET, Josep, 2012, Estudio de los futuros estándares HTML5 y CSS3. Propuesta de actualización del sitio www.mpiua.net. Lleida: Universidad de Lleida, Escuela Politécnica Superior, Ingeniería Técnica en Informática de Gestión.
16. CALDERÍN, Dunet and PÉREZ, Leonardo, 2012, Integración de los instrumentos de evaluación para la gestión de las evidencias en la plataforma educativa Zera. La Habana, Cuba: Universidad de las Ciencias Informáticas.
17. CABRERA GONZÁLEZ, Lianet, Lianet Cabrera GONZÁLEZ a Enrique Roberto Pompa TORRES. 2012. Extensión de Visual Paradigm for UML para el desarrollo dirigido por modelos de aplicaciones de gestión de información. Serie Científica [online]. 2012, Vol. 5, No. 10. Available from: https://publicaciones.uci.cu/index.php/SC/article/view/1032.
18. Desktop Central Official website, 2012, [online]. Available from: https://www.manageengine.com
19. Mora, S.L., 2012, *Historia, principios básicos y clientes web.*
20. BAUKES, Mike, 2013, ScriptRock. MySQL vs Postgres [online]. 9 August 2013. [Accessed 14 February 2015]. Available from: https://www.scriptrock.com/articles/postgres -vsmysql/.

21. PINTA MUSO, Fausto Ramiro and SALAZAR LLUMITASIG, Luis Efrain, 2013, Sistema de control de asistencia de personal del instituto de suelos de Granma, [online]. 2013. [Accessed 5 June 2015]. Available from: http://repositorio.utc.edu.ec/handle/27000/1653.
22. SAAVEDRA LÓPEZ, Disney, Yuniesky ARMENTERO MORENO a Zoila Esther MORALES TABARES. 2013. Aplicación web para la realización de estudios farmacocinéticos, versión 2.0. Revista Cubana de Informática Médica. 2013, s. 118–131. ISSN 1684-1859.
23. TORRES HERNÁNDEZ, Oscar Daniel. 2013. Propuesta arquitectónica para el núcleo del Gestor de Documentos Administrativos eXcriba 3.0. La Habana, 2013. Universidad de las Ciencias Informáticas.
24. Robbins, Stephen P; Coulter, Mary, 2013 Management 10th Edition, [online].
25. VERMILION, 2013, Responsive CSS Framework Comparison. Bootstrap vs. Foundation vs. Skeleton [online]. 21 November 2013. [Accessed 10 February 2015]. Available from: http://responsive.vermilion.com/compare.php.
26. Philips, Mark, 2013 Review: Puppet vs. Chef vs. Ansible vs. Salt on *InfoWorld* [online], [Accessed 24 April 2015]. Available from: http://www.infoworld.com/article/2609482/data-center/data-center-review-puppet-vs-chef-vs-ansible-vs-salt.html.
27. KOTOV, 2013, GitLab, gestor de repositorios para Git. | RooTeando. [online]. 18 February 2013. [Accessed 15 February 2015]. Available from: http://rooteando.com/gitlab-administrador-web-derepositorios -git.
28. PostgreSQL: Dossier de Prensa de PostgreSQL 9.3, 2014, [online], [Accessed 14 Febrero 2015]. Available from: http://www.postgresql.org/about/press/presskit93/es/.
29. ScriptRock, 2014. Ansible vs Chef [online], [Accessed 23 April 2015]. Available from: http://www.scriptrock.com/articles/ansible-vs-chef
30. DameWare Remote Support Official web site, 2015, [online]. Available from: https://www.dameware.com
31. TeamViewer Official web site, 2015, [online]. Available from: https://www.teamviewer.com
32. NETBEANS.ORG. NetBeans IDE 7.2.1 Release Notes, [online]. [Cited: 21 de febrero de 2015]. Available from: https://netbeans.org/community/releases/72/relnotes.html
33. Historias de Usuario. [cited 28/3/2014;Available from: http://www.genbetadev.com/metodologias-de-programacion/historias-de-usuario-una-formanatural-de-analisis-funcional

34. *Desarrollo web ágil con Symfony2* [online] [no date]. S.l.: s.n. [Accessed: 13 marzo 2015]. Available from: http://symfony.es/libro.
35. MySQL, [no date], [online], [Accessed 27 May 2015]. Available from: http://www.mastermagazine.info/termino/6051.php.
36. SANDEEP BEDI, [no date], About Bootstrap Framework. *QueryHome* [online]. [Accessed 16 June 2015]. Available from: http://www.queryhome.com/51419/about-bootstrap-framework.
37. Stoner, James. Administración. Parte I y II, 5ta. Edición.
38. Bergmann, Sebastián. PHPUnit Manual. [online] [Accessed: 10 de 04 de 2015.] Publication date Edition for PHPUnit 4.0. Updated on 2014-03-29. http://phpunit.de/manual/current/en/phpunit-book.pdf.
39. SaltStack Enterprise product overview, 2015. SaltStack [online], [Accessed 24 April 2015]. Available from: http://saltstack.com/enterprise/.
40. www.postgresql.org.es [online]. [vid. 10. febrero 2015]. Available from: http://postgresql.org.es/
41. Welcome to the Apache Software Foundation! [online]. [vid. 10. febrero 2015]. Available from: http://apache.org/
42. VALDIVIA, Guzmán, et al. Concepto y elementos de administración.
43. BUYENS, Jim, [no date], *Aprenda Desarrollo de bases de datos web*. McGRAW-HILL. ISBN 84-481-2903-2...
44. Conceptos de Centralización y Descentralización y sus diferencias. *Webscolar* [online]. [Accessed 7. febrero 2015]. Available from:http://www.webscolar.com/conceptos-de-centralizacion-y-descentralizacion-y-sus-diferencias.
45. What is programming language? Definition and meaning, [no date]. *BusinessDictionary.com* [online], [Accessed 27 May 2015]. Available from: http://www.businessdictionary.com/definition/programming-language.html.
46. Martínez, Vicen. Desarrollo web y Marketing online. Introducción a Symfony2, [no date], [online], [Accessed 16 Febrero 2015]. http://vicenmartinez.wordpress.com/2013/04/24/introduccion-a-symfony2/.
47. Oracle Normalization Normal Form 1st 2nd 3rd 4th 5th, [no date], [online], [Accessed 16 June 2015]. Available from:http://psoug.org/reference/normalization.html.

Anexos

Anexo 1: Especificación de Historias de Usuario

HU2 ARF2. Administrar llaves

RF relacionados:

- **RF 2.1** Listar llaves.
- **RF 2.2** Aceptar llave.
- **RF 2.3** Rechazar llave.
- **RF 2.4** Crear computadora.

Historia de usuario	
No.:5	**Nombre:** Listar llaves
Usuario: Administrador	
Prioridad en el negocio: Alta	**Nivel de complejidad:** Baja
Estimación: 4d	**Iteración asignada:** 1
Descripción: se listan las llaves de los Minion (computadoras), tanto las aceptadas, pendientes y denegadas.	
Información adicional (observaciones): No aplica.	
No.:6	**Nombre:** Aceptar llave
Usuario: Administrador	
Prioridad en el negocio: Alta	**Nivel de complejidad:** Baja
Estimación: 4d	**Iteración asignada:** 1
Descripción: se aceptan las llaves de los Minion.	
Información adicional (observaciones): No aplica.	
No.:7	**Nombre:** Rechazar llave
Usuario: Administrador	
Prioridad en el negocio: Alta	**Nivel de complejidad:** Baja
Estimación: 2d	**Iteración asignada:** 1
Descripción: se rechazan las llaves de los Minion (computadoras).	
Información adicional (observaciones): No aplica.	
Historia de usuario	
No.:8	**Nombre:** Crear computadora
Usuario: Administrador	
Prioridad en el negocio: Alta	**Nivel de complejidad:** Medio
Estimación: 3d	**Iteración asignada:** 1

Descripción: se crea una computadora en el sistema.
Información adicional (observaciones): el nombre que aparece para las computadoras es el asociado al dominio.

HU4 ARF4. Administrar Áreas

RF relacionados:

- **RF 4.1** Listar áreas.
- **RF 4.2** Crear área.
- **RF 4.3** Asignar computadora.
- **RF 4.4** Editar área.
- **RF 4.5** Eliminar área.

Historia de usuario	
No.:9	**Nombre:** Listar áreas
Usuario: Administrador	
Prioridad en el negocio: Alta	**Nivel de complejidad:** Alta
Estimación: 4d	**Iteración asignada:** 2
Descripción: se listan las áreas existentes en el sistema.	
Información adicional (observaciones): No aplica.	
No.:10	**Nombre:** Crear área
Usuario: Administrador	
Prioridad en el negocio: Alta	**Nivel de complejidad:** Alta
Estimación: 4d	**Iteración asignada:** 2
Descripción: se crea un área en el sistema.	
Información adicional (observaciones): No aplica.	
No.:11	**Nombre:** Asignar computadora
Usuario: Administrador	
Prioridad en el negocio: Alta	**Nivel de complejidad:** Bajo
Estimación: 2d	**Iteración asignada:** 1
Descripción: se asigna una o un grupo de computadoras a un área en el sistema.	
Información adicional (observaciones): No aplica.	
No.:12	**Nombre:** Editar área
Usuario: Administrador	
Prioridad en el negocio: Media	**Nivel de complejidad:** Media
Estimación: 4d	**Iteración asignada:** 2
Descripción: se editan las áreas por concepto de cambio de nombre.	
Información adicional (observaciones): se ejecuta cuando es necesario modificar la información de un área.	

No.:13	**Nombre:** Eliminar área
Usuario: Administrador	
Prioridad en el negocio: Baja	**Nivel de complejidad:** Baja
Estimación: 3d	**Iteración asignada:** 2
Descripción: se elimina un área específica del sistema borrando de igual manera todos sus datos asociados.	
Información adicional (observaciones): No aplica.	

HU5 ARF5. Administrar comandos

RF relacionados:

- **RF 5.1** Crear evento.
- **RF 5.2** Listar evento.
- **RF 5.3** Ejecutar evento.
- **RF 5.4** Editar evento.
- **RF 5.5** Eliminar evento.

Historia de usuario	
No.:14	**Nombre:** Crear evento
Usuario: Administrador	
Prioridad en el negocio: Muy alta	**Nivel de complejidad:** Muy alta
Estimación: 5d	**Iteración asignada:** 2
Descripción: se crea un evento en el sistema.	
Información adicional (observaciones): estos eventos son generados a partir de un formulario.	
No.:15	**Nombre:** Listar evento
Usuario: Administrador	
Prioridad en el negocio: Muy alta	**Nivel de complejidad:** Muy alta
Estimación: 4d	**Iteración asignada:** 2
Descripción: se listan los eventos creados en el sistema.	
Información adicional (observaciones): No aplica.	
No.:16	**Nombre:** Ejecutar evento
Usuario: Administrador	
Prioridad en el negocio: Muy alta	**Nivel de complejidad:** Muy alta
Estimación: 5d	**Iteración asignada:** 1
Descripción: se asigna un evento que se ejecutará sobre una o varias máquinas.	
Información adicional (observaciones): No aplica.	
No.:17	**Nombre:** Editar evento
Usuario: Administrador	
Prioridad en el negocio: Alta	**Nivel de complejidad:** Alta

Estimación: 4d	**Iteración asignada:** 2
Descripción: se editan los eventos por concepto de cambio de nombre, descripción y comando a ejecutar.	
Información adicional (observaciones): se ejecuta cuando es necesario modificar la información de un evento.	
No.:18	**Nombre:** Eliminar evento
Usuario: Administrador	
Prioridad en el negocio: Media	**Nivel de complejidad:** Media
Estimación: 2d	**Iteración asignada:** 2
Descripción: se elimina un evento específico del sistema borrando de igual manera todos sus datos asociados.	
Información adicional (observaciones): No aplica.	

Anexo 2: Especificación de las tarjetas CRC

Módulo Usuario

Tarjeta CRC	
Número: 1	**Nombre de la clase:** DefaultController
Responsabilidades: ▪ Autenticar usuario ▪ Editar perfil de usuario ▪ Visualizar perfil de usuario ▪ Salir del sistema	
Colaboraciones: Usuario, UsuarioType	
Tarjeta CRC	
Número: 2	**Nombre de la clase:** Usuario
Responsabilidades: Representa la entidad Usuario que define las características de los objetos de este tipo y será mapeada a tabla relacional en la base de datos de la aplicación.	
Colaboraciones:	
Tarjeta CRC	
Número: 3	**Nombre de la clase:** UsuarioType
Responsabilidades: Esta clase permite construir, en la vista, el formulario del perfil de usuario en las acciones de visualizar y editar perfil de usuario.	
Colaboraciones:	
Tarjeta CRC	
Número: 4	**Nombre de la clase:** UsuarioController
Responsabilidades: ▪ Crear usuario	

▪ Listar usuario ▪ Editar usuario ▪ Eliminar usuario
Colaboraciones: Usuario, UsuarioType

Módulo Área

Tarjeta CRC	
Número: 5	**Nombre de la clase:** DefaultController
Responsabilidades: ▪ Listar áreas	
Colaboraciones: Area, AreaType, AreaController	
Tarjeta CRC	
Número: 6	**Nombre de la clase:** Area
Responsabilidades: Representa la entidad *Area* que define las características de los objetos de este tipo y será mapeada a tabla relacional en la base de datos de la aplicación.	
Colaboraciones:	
Tarjeta CRC	
Número: 7	**Nombre de la clase:** AreaType
Responsabilidades: Esta clase permite construir, en la vista, el formulario de área en las acciones de crear, visualizar y editar área.	
Colaboraciones:	
Tarjeta CRC	
Número: 8	**Nombre de la clase:** AreaController
Responsabilidades: ▪ Crear área ▪ Listar áreas ▪ Editar área ▪ Eliminar área ▪ Adicionar maquinas al área ▪ Refrescar las áreas	
Colaboraciones: Area, AreaType	

Módulo Comando

Tarjeta CRC

Número: 9	**Nombre de la clase:** EventoController
Responsabilidades: ▪ Listar comandos ▪ Crear comando ▪ Editar comando ▪ Ejecutar comando ▪ Eliminar comando	
Colaboraciones: Evento, EventoType	
Tarjeta CRC	
Número: 10	**Nombre de la clase:** Evento
Responsabilidades: Representa la entidad Evento que define las características de los objetos de este tipo y será mapeada a tabla relacional en la base de datos de la aplicación.	
Colaboraciones:	
Tarjeta CRC	
Número: 11	**Nombre de la clase:** EventoType
Responsabilidades: Esta clase permite construir, en la vista, el formulario de comando en las acciones de crear, visualizar, editar área y ejecutar.	
Colaboraciones:	

Módulo Máquinas

Tarjeta CRC	
Número: 12	**Nombre de la clase:** DefaultController
Responsabilidades: ▪ Listar máquinas ▪ Aceptar llaves ▪ Rechazar llaves	
Colaboraciones: Máquina	

Anexo 3: Diseño de casos de prueba

CP2_Crear Usuario

Escenario	Descripción	Usuario	Contraseña	Confirmar Contraseña	Respuesta del sistema	Flujo central
EC 2.1	El sistema	V	V	V	El sistema	1-El usuario accede

Crear usuario de forma correcta.	crea un usuario de forma correcta.	prueba	Prueba1234	Se introduce la misma contraseña que se introdujo en el campo Contraseña	registra la información del usuario y muestra un mensaje de confirmación de la acción: "Se creó una nueva cuenta de usuario para prueba".	al menú de Usuario. 2-El usuario selecciona la opción Añadir usuario. 3-El sistema muestra un formulario. 4-El usuario introduce la información y presiona el botón: Aceptar.
EC 2.2 Crear usuario de forma incorrecta.	El sistema no crea un usuario.	I	V	V	El sistema no crea el usuario y emite los siguientes mensajes de errores: "El campo usuario no tiene un formato válido, debe tener al menos 3 caracteres." "Las contraseñas que ha indicado no coinciden."	
		12nh	fortes1234	Se introduce la misma contraseña que se introdujo en el campo Contraseña		
		I	I	I		
		asd	as	No se introduce la misma contraseña que se introdujo en el campo Contraseña		
EC 2.3 Crear usuario dejando campos vacíos.	El sistema no crea un usuario dejando campos obligatorios vacíos.	I	I	I	El sistema no crea el usuario y emite los siguientes mensajes de errores: "El campo Nombre de usuario es obligatorio". "El campo Contraseña es obligatorio".	
		Vacío	Vacío	Vacío		

CP3_Editar Usuario

Escenario	Descripción	Usuario	Contraseña	Confirmar Contraseña	Respuesta del sistema	Flujo central
EC 3.1 Editar usuario de forma correcta.	El sistema modifica la información del usuario de forma correcta.	V probador	V probador1234	V Se introduce la misma contraseña que se introdujo en el campo Contraseña	El sistema verifica y actualiza la información del usuario y muestra un mensaje de confirmación de la acción: "Se han guardado los cambios".	1-El usuario accede al menú de Usuario. 2-El sistema muestra el listado de los usuarios existentes. 3-El usuario selecciona el usuario que desea modificar y presiona la opción: editar. 4- El sistema habilita un formulario para editar la información deseada. 5-El usuario introduce la información y presiona el botón: Guardar.
EC 3.2 Editar usuario de forma incorrecta.	El sistema no modifica la información del usuario.	I 12nh	V fortes1234	V Se introduce la misma contraseña que se introdujo en el campo Contraseña	El sistema no modifica al usuario y emite los siguientes mensajes de errores: "El campo usuario no tiene un formato válido, debe tener al menos 3 caracteres." "Las contraseñas que ha indicado no coinciden."	
		I	I	I		
		asd	as	No se introduce la misma contraseña que se		

				introdujo en el campo Contraseña		
EC 3.3 Editar usuario dejando campos vacíos.	El sistema no modifica un usuario dejando campos obligatorios vacíos.	I Vacío	I Vacío	I Vacío	El sistema no modifica al usuario y emite los siguientes mensajes de errores: "El campo Nombre de usuario es obligatorio". "El campo Contraseña es obligatorio".	

CP4_Eliminar Usuario

Escenario	Descripción	Respuesta del sistema	Flujo central
EC 4.1 Eliminar usuario.	El sistema elimina al usuario.	El sistema le muestra un mensaje al usuario de confirmación de la acción: "La cuenta prueba se ha eliminado".	1-El usuario accede al menú de Usuario. 2-El sistema muestra el listado de los usuarios existentes. 3-El usuario selecciona el usuario que desea eliminar y presiona la opción: eliminar. 4-El sistema muestra un formulario con las opciones Eliminar y Cancelar. 5-El usuario selecciona la opción Eliminar. 6-El sistema elimina la cuenta y muestra un mensaje de confirmación al usuario.1-El usuario accede al menú de Usuario. 2-El sistema muestra el listado de los usuarios existentes. 3-El usuario selecciona el usuario que desea eliminar y presiona la opción: eliminar. 4-El usuario selecciona la opción Cancelar. 5-El sistema cancela la acción y muestra el listado con los usuarios existentes.
EC 4.2 Cancelar la operación.	El sistema cancela la operación.	El sistema cancela la operación.	

CP5_Mostrar Usuario

Escenario	Descripción	Respuesta del sistema	Flujo central
EC 5.1 Listar usuarios.	El sistema lista a los usuarios existentes en el sistema.	El sistema muestra los usuarios existentes en el sistema con el rol y el estado introducido por el usuario.	1-El usuario accede al menú de Usuarios. El sistema muestra el listado de los usuarios existentes.

CP6_Crear Comando

Escenario	Descripción	Nombre de comando	Descripción (comando)	Comando	Respuesta del sistema	Flujo central
EC 6.1 Crear comando de forma correcta.	El sistema crea un comando de forma correcta.	V Prueba	N/A	V Se introduce un comando en el campo Comando.	El sistema registra la información del comando y muestra un mensaje de confirmación de la acción: "Se creó un nuevo comando".	1-El usuario accede al menú de Comandos. 2-El usuario selecciona la opción Añadir Nuevo comando. 3-El sistema muestra un formulario. 4-El usuario introduce la información y presiona el botón: Aceptar.
EC 6.2 Crear comando de forma incorrecta.	El sistema no crea un comando.	I 12nh	N/A	V Se introduce la misma contraseña que se introdujo en el campo Contraseña	El sistema no crea el comando y emite los siguientes mensajes de errores: "El campo nombre no tiene un formato válido, debe tener al menos 3 caracteres."	
		I asd	N/A	V No se introduce la misma contraseña que se introdujo en el campo Contraseña		
EC 6.3	El sistema no	I	N/A	I	El sistema no	

Crear comando dejando campos vacíos.	crea un comando dejando campos obligatorios vacíos.	Vacío		Vacío	crea el usuario y emite los siguientes mensajes de errores: "El campo Nombre de comando es obligatorio". "El campo Comando es obligatorio".	

CP7_Editar Comando

Escenario	Descripción	Nombre de comando	Descripción (comando)	Comando	Respuesta del sistema	Flujo central
EC 7.1 Editar comando de forma correcta.	El sistema modifica un comando de forma correcta.	**V**	N/A	**V**	El sistema registra la información del comando y muestra un mensaje de confirmación de la acción: "Comando modificado".	1-El usuario accede al menú de Comandos. 2-El usuario selecciona la opción Añadir Nuevo comando. 3-El sistema muestra un formulario. 4-El usuario introduce la información y presiona el botón: Aceptar.
		prueba		Se introduce un comando en el campo Comando.		
EC 7.2 Editar comando de forma incorrecta.	El sistema no modifica un comando.	I	N/A	V	El sistema no modifica el comando y emite los siguientes mensajes de errores: "El campo nombre no tiene un formato válido, debe tener al menos 3 caracteres."	
		12nh		Se introduce un comando en el campo Comando.		
		I	N/A	V		
		asd		Se introduce un comando en el campo Comando.		

EC 7.3 Editar comando dejando campos vacíos.	El sistema no modifica un comando dejando campos obligatorios vacíos.	I	N/A	I	El sistema no crea el usuario y emite los siguientes mensajes de errores: "El campo Nombre de comando es obligatorio". "El campo Comando es obligatorio".	

CP8_Eliminar Comando

Escenario	Descripción	Respuesta del sistema	Flujo central
EC 8.1 Eliminar comando.	El sistema elimina al comando.	El sistema le muestra un mensaje al usuario de confirmación de la acción**: "El comando prueba se ha eliminado"**.	1-El usuario accede al menú de Comandos. 2-El sistema muestra el listado de los comandos existentes. 3-El usuario selecciona el comando que desea eliminar y presiona la opción: eliminar. 4-El sistema muestra un formulario con las opciones Eliminar y Cancelar. 5-El usuario selecciona la opción Eliminar. 6-El sistema elimina el comando y muestra un mensaje de confirmación al usuario.

EC 8.2 Cancelar la operación.	El sistema cancela la operación.	El sistema cancela la operación.	1-El usuario accede al menú de Comandos. 2-El sistema muestra el listado de los comandos existentes. 3-El usuario selecciona el comando que desea eliminar.4-El sistema muestra un formulario con las opciones Eliminar y Cancelar. 5-El usuario selecciona la opción Cancelar. 6-El sistema cancela la acción y muestra el listado con los comandos existentes.

CP9_Listar Comando

Escenario	Descripción	Respuesta del sistema	Flujo central
EC 9.1 Listar comandos.	El sistema lista los comandos existentes en el sistema.	El sistema muestra las áreas existentes en el sistema con el nombre, descripción y comando.	1-El usuario accede al menú de Comandos. 2-El sistema muestra el listado de los comandos existentes.

CP10_Crear Áreas

Escenario	Descripción	Nombre Areas	Computadoras	Respuesta del sistema	Flujo central
EC 10.1 Crear área de forma correcta.	El sistema crea el área de forma correcta.	V	N/A	El sistema almacena la información y muestra un mensaje de confirmación de la acción: El área ha sido creada.	1-El usuario autenticado en el sistema selecciona la opción Áreas. 2-El usuario selecciona la opción Nueva Área. 3-El sistema muestra un formulario. 4-El usuario introduce la
		prueba			
EC 10.2 Crear área de forma incorrecta.	El sistema no crea el área dejando campos obligatorios vacíos.	I	N/A	El sistema no autentica al usuario y emite los siguientes mensajes de errores: "El campo Usuario es obligatorio". "El	
		Vacío			

				campo Contraseña es obligatorio".	información y presiona el botón Guardar.

CP11_Editar Área

Escenario	Descripción	Nombre Areas	Computadoras	Respuesta del sistema	Flujo central
EC 11.1 Editar área de forma correcta.	El sistema modifica la información del área de forma correcta.	V	N/A	El sistema verifica y actualiza la información del área y muestra un mensaje de confirmación de la acción: "Se han guardado los cambios".	1--El usuario autenticado en el sistema selecciona la opción Áreas. 2-El usuario selecciona la opción Editar Área. 3-El sistema muestra un formulario. 4-El usuario introduce la información y presiona el botón Actualizar.
		prueba			
EC 11.2 Editar área de forma incorrecta.	El sistema no guarda el área editada dejando campos obligatorios vacíos.	I	N/A	El sistema no modifica el área y emite los siguientes mensajes de errores: "El campo nombre es obligatorio".	
		Vacío			

CP12_Eliminar Área

Escenario	Descripción	Respuesta del sistema	Flujo central
EC 12.1 Eliminar un área.	El sistema elimina el área.	El sistema verifica y actualiza la información del área y muestra un mensaje de confirmación de la acción: "Se han guardado los cambios".	1-El usuario accede al menú de Áreas. 2-El sistema muestra el listado de las áreas existentes. 3-El usuario selecciona el área que desea eliminar y presiona la opción: eliminar. 4-El sistema muestra un formulario con las opciones Eliminar y Cancelar. 5-El usuario selecciona la opción Eliminar. 6-El sistema elimina el área y muestra un mensaje de confirmación al usuario. 1-El usuario accede al menú de Áreas. 2-El sistema muestra el listado de las áreas existentes. 3-El usuario selecciona el área que desea eliminar y presiona la opción: cancelar. 4-El sistema cancela la acción y muestra el listado con las áreas existentes.
EC 12.2 Cancelar la operación.	El sistema no guarda el área editada dejando campos obligatorios vacíos.	El sistema no modifica el área y emite los siguientes mensajes de errores: "El campo nombre es obligatorio".	

CP13_Listar Áreas

Escenario	Descripción	Respuesta del sistema	Flujo central
EC 13.1 Listar áreas	El sistema lista las áreas existentes en el sistema.	El sistema muestra las áreas existentes en el sistema con el nombre y la cantidad de máquinas.	1-El usuario accede al menú de Áreas. 2-El sistema muestra el listado de las áreas existentes.

CP14_Listar Computadoras

Escenario	Descripción	Respuesta del sistema	Flujo central
EC 14.1 Listar	El sistema lista las	El sistema muestra las	1-El usuario accede al

computadoras.	computadoras existentes en el sistema.	computadoras existentes en el sistema divididas en tres grupos aceptadas, rechazadas y en espera.	menú de Computadoras. El sistema muestra el listado de las computadoras existentes.

CP15_Aceptar Llaves

Escenario	Descripción	Respuesta del sistema	Flujo central
EC 15.1 Listar áreas	El sistema acepta las computadoras existentes en el sistema.	El sistema muestra el listado de las computadoras actualizado.	1-El usuario accede al menú de Computadoras. 2-El usuario selecciona la opción aceptar todas.

CP16_Rechazar Llaves

Escenario	Descripción	Respuesta del sistema	Flujo central
EC 16.1 Listar áreas.	El sistema rechazar las computadoras existentes en el sistema.	El sistema muestra el listado de las computadoras actualizado.	1-El usuario accede al menú de Computadoras. 2-El usuario selecciona la opción rechazar todas.

Anexo 4: Pruebas unitarias

```
public function testNombre() {
    $evento = new Evento();
    //nombre en blanco
     $listaErrores = $this->validator->validate($evento);
     $error = $listaErrores[0];
     $this->assertEquals('This value should not be blank.', $error->getMessage());
     $evento->setNombre('nombreEvento');
     $this->assertEquals('nombreEvento', $evento->getNombre());
}

public function testDescripcion() {
    //descripcion
     $evento = new Evento();
     $listaErrores = $this->validator->validate($evento);
     $error = $listaErrores[0];
     $this->assertEquals('This value should not be blank.', $error->getMessage());
     $evento->setDescripcion('descripcionCompleta');
     $this->assertEquals('descripcionCompleta', $evento->getDescripcion());
}

public function testComando() {
    //comando
     $evento = new Evento();
     $listaErrores = $this->validator->validate($evento);
     $error = $listaErrores[0];
     $this->assertEquals('This value should not be blank.', $error->getMessage());
     $evento->setComando('nombreEvento');
     $this->assertEquals('nombreEvento', $evento->getComando());
}
```

Ilustración 15 Prueba unitaria. Entidad Comando.

```php
public function testUsuario() {
    $usuario = new Usuario();
      usuario en blanco
    $listaErrores = $this->validator->validate($usuario);
    $error = $listaErrores[0];
    $this->assertEquals('This value should not be blank.', $error->getMessage());
    $usuario->setUsuario('usuario');
    $this->assertEquals('usuario', $usuario->getUsuario());
    usuario corto
    $usuario->setUsuario('qw');
    $this->assertEquals('qw', $usuario->getUsuario());
    $listaErrores = $this->validator->validate($usuario);
    $error = $listaErrores[0];
    $this->assertEquals('El campo usuario debe tener al menos 3 carateres.', $error->getMessage());
    usuario largo
    $username = 'inputlasodaisdniasndfasdbfasdbfasbdfbashdfbahsdfhasdfqiluwehwbvqbvbqervbbqbdvsanvbmasbdv';
    $usuario->setUsuario($username);
    $listaErrores = $this->validator->validate($usuario);
    $error = $listaErrores[0];
    $this->assertEquals('El campo usuario no debe tener más de 30 caracteres.', $error->getMessage());
}

public function testPassword() {
    $usuario = new Usuario();
    $usuario->setUsuario('usuarioValido');
    $usuario->setSalt('asdfyasfhqwehoqwe7823');
    $listaErrores = $this->validator->validate($usuario);
    $error = $listaErrores[0];
    $this->assertEquals('El campo contraseña no puede estar en blanco.', $error->getMessage());
    //password corto
    $usuario->setPassword('qwertyui');
    $this->assertEquals('qwertyui', $usuario->getPassword());
}
```

Ilustración 16 Prueba unitaria. Entidad Usuario.

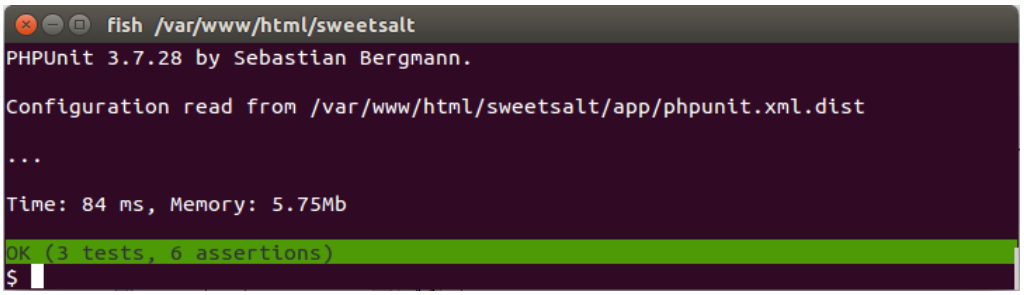


Ilustración 17 Prueba unitaria realizada en la entidad Comandos.

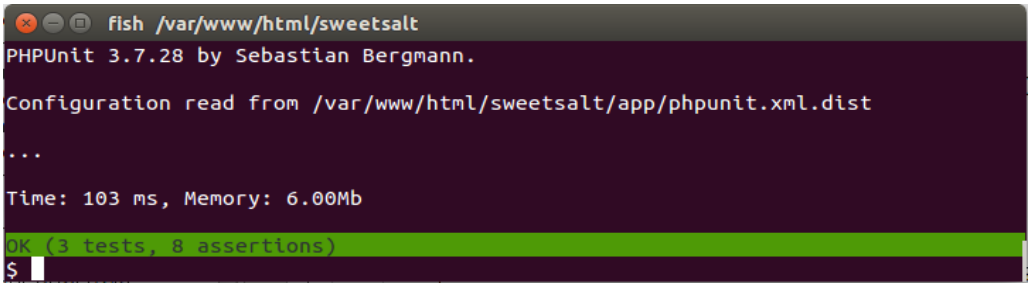


Ilustración 18 Prueba unitaria realizada en la entidad Usuario.