UNIVERSIDAD DEL VALLE DE GUATEMALA

Facultad de Ingeniería

Análisis e implementación de infraestructura para migrar ambiente de desarrollo, basado en prácticas de ingeniería de software

Trabajo de graduación presentado por Héctor Antonio Hurtarte Estrada para optar al grado académico de Licenciado en Ingeniería en Ciencias de la Computación

Guatemala

2014

Análisis e implementación de infraestructura para migrar ambiente de desarrollo, basado en prácticas de ingeniería de software

UNIVERSIDAD DEL VALLE DE GUATEMALA

Facultad de Ingeniería

Trabajo de Graduación

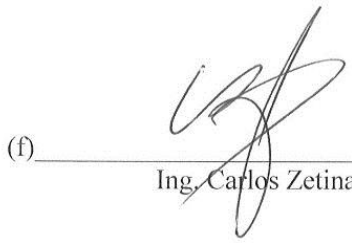
Análisis e implementación de infraestructura para migrar ambiente de desarrollo, basado en prácticas de ingeniería de software

Trabajo de graduación presentado por Héctor Antonio Hurtarte Estrada para optar al grado académico de Licenciado en Ingeniería en Ciencias de la Computación

Guatemala

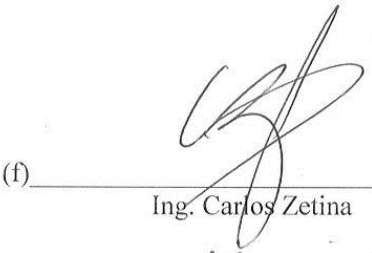
2014

Vo.Bo.

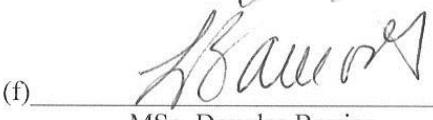


(f)_____
Ing. Carlos Zetina

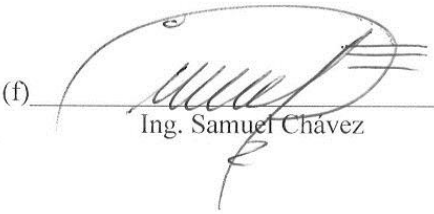
Tribunal



(f)_____
Ing. Carlos Zetina



(f)_____
MSc. Douglas Barrios



(f)_____
Ing. Samuel Chávez

Fecha de Aprobación: Guatemala 30 de Octubre del 2014

PREFACIO

El presente informe de trabajo de graduación surge de la necesidad de adecuar las tecnologías y prácticas de programación de la empresa Consulte, S.A. hacia las nuevas posibilidades y necesidades del mercado. Ofrece la oportunidad de llevar a la práctica los conocimientos adquiridos durante la trayectoria universitaria en un ambiente profesional y permite ganar una experiencia invaluable en el área de la carrera que me resulta de mayor interés: la ingeniería de software.

ÍNDICE

Página

LISTA DE GRÁFICOS

Página

LISTA DE TABLAS

Página

RESUMEN

El objetivo de la práctica profesional concluida es el diseño, implementación y establecimiento de lineamientos de una base de código sobre la cual puedan ser desarrollados sistemas con especificaciones similares a los trabajados actualmente. Esta necesidad surge del análisis del desarrollo y la forma de trabajo de una empresa de soluciones de software. Dentro de la empresa gran parte del desarrollo se lleva a cabo sobre una plataforma que ya no cuenta con soporte activo, y no se ha considerado la implementación de prácticas de ingeniería de software que puedan apoyar el trabajo que se hace.

Dado el anterior escenario se plantea el desarrollo de una base de código que implemente la funcionalidad común que la empresa ofrece a sus clientes en la mayoría de sus productos. Este desarrollo incluye una arquitectura modular pensada en la continuidad que la empresa pueda darle al desarrollo de nuevos componentes según sus necesidades. Para llevar a cabo este trabajo se hizo un análisis de tecnologías web de código abierto y la documentación del código desarrollado.

Por otro lado se efectúa la instalación y configuración de una serie de herramientas que buscan apoyar la metodología de trabajo dentro de la empresa: sistema de control de versiones, entorno de integración continua, sistema de seguimiento a asuntos. Por otro lado, el desarrollo se realiza siguiendo un subconjunto de las prácticas propuestas por las metodologías de desarrollo ágil Scrum y XP.

Luego del desarrollo de la práctica se brinda a la empresa esta base de código funcional, las herramientas instaladas y configuradas y la experiencia de haber seguido un desarrollo utilizando prácticas de amplia adopción dentro de la industria.

I. Introducción

La empresa está dedicada al desarrollo de soluciones de *software* a la medida para compañías de algunos sectores específicos del mercado como medios y publicidad, bufetes de abogados y empresas de servicios varios. Fundada en 1998, su metodología de trabajo y las tecnologías sobre las que implementa sus sistemas han ido variando con el tiempo, iniciando en ambientes para el sistema operativo de disco MSDOS, moviéndose posteriormente a entornos visuales. Dado su enfoque, la empresa ha trabajado sus distintas soluciones en base a un conjunto de características comunes que ha detectado en sus múltiples clientes y que aplica en la mayoría de sus sistemas.

Actualmente, su planificación estratégica involucra ofrecer a los clientes las bondades del *software* en un ambiente web, que ofrece accesibilidad desde medios *no convencionales*, además de la posibilidad del *Software as a service* (SaaS). Por otro lado, la experiencia adquirida les ha llevado a decidir aprovechar esta migración para reestructurar algunos de sus sistemas, haciendo enmiendas a diseños desarrollados algún tiempo atrás y buscando aprovechar ventajas de patrones de diseño de *software* tales como el Modelo Vista Controlador (MVC).

Además de la cuestión tecnológica, en la empresa se detecta la posibilidad de mejorar la forma y metodología de trabajo en términos de desarrollo, es decir, las prácticas de ingeniería de software. En la actualidad, la forma de trabajo de la empresa, desde el análisis de requerimientos hasta el soporte y mantenimiento de aplicaciones, es guiada por la experiencia que su gerente ha adquirido con el tiempo. De aquí que resulte viable implementar metodologías y herramientas estudiadas durante la carrera.

En este marco, surge la posibilidad de llevar a cabo las prácticas profesionales finales de la carrera de Ingeniería en Ciencias de la Computación. Estas consistirán en el análisis de la forma de trabajo seguida por la empresa, la detección de sus fortalezas y la sugerencia e implementación piloto de prácticas de ingeniería de software que permitan abordar los problemas que enfrenta. Para tratar de forma práctica estas sugerencias, se propone además sentar la arquitectura y las bases de código para uno de los proyectos

de la empresa, de forma que al final de la práctica puedan evidenciarse los beneficios que resultan de una aplicación más profunda de la ingeniería de software.

II. Objetivos

A. Objetivo general

Diseñar, implementar y establecer los lineamientos de una base de código sobre la cual puedan ser desarrollados sistemas con especificaciones similares a los que actualmente trabaja la empresa.

B. Objetivos específicos

1. Presentar una propuesta de arquitectura de software que incluya un análisis de tecnologías y diseño de base de datos, según los requerimientos detectados.

2. Implementar un módulo inicial de administración, de acuerdo a algunas características comunes que la empresa ofrece en todas sus soluciones de *software*, que establezca los lineamientos a seguir por los módulos más específicos.

3. Implementar herramientas que faciliten prácticas adecuadas de ingeniería de software tales como control de versiones, sistema de tickets, automatización de pruebas unitarias y funcionales, documentación y evaluación de pruebas no funcionales y de aceptación, ambiente de integración continua.

4. Implementar un subconjunto de las prácticas de desarrollo propuestas por la metodología de desarrollo ágil **Scrum** en términos de análisis de requerimientos y diseño, planificación de trabajo y estimación de tiempos.

III. Marco teórico

A. Ingeniería de software

De acuerdo con el SEVOCAB[1], Vocabulario de Sistemas e Ingeniería de Software de ISO/IEC/IEEE, la ingeniería de software puede definirse como la aplicación de un acercamiento sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento del *software*. La guía para el cuerpo de conocimiento de ingeniería de software, SWEBOK [1], de donde fue obtenida la definición anterior, trata en su edición de 2014 con relativa profundidad los distintos aspectos que abarca la práctica de la ingeniería de software. Esto va desde la toma de requerimientos, pasando por el diseño, construcción y mantenimiento del *software* hasta el aseguramiento de calidad del mismo y el proceso general de ingeniería.

A continuación son resumidas las áreas de conocimiento que se juzgan más relevantes para el desarrollo del presente proyecto.

El área de conocimiento de los **Requerimientos de *software*** distingue los requerimientos funcionales como las características o capacidades solicitados, mientras que los no-funcionales como restricciones o requerimientos de calidad (desempeño, seguridad, mantenibilidad). Dentro de este concepto se distinguen las *propiedades emergentes*, que son aquellos requerimientos que no pueden ser abordados por un solo componente sino que dependen de la interacción entre todos los componentes de un sistema. Dentro de esta área se introduce también el concepto de *Historias de usuario* como descripciones cortas y de alto nivel de funcionalidad requerida, expresadas en los términos del usuario o cliente; además se sugiere que un procedimiento de aceptación apropiado sea definido por el usuario para determinar el momento en el que el objetivo de una historia es completado [1].

Dentro del área de **Diseño de software** se define al mismo como el proceso de definición de la arquitectura, componentes e interfaces y otras características de un sistema o componente, y también como el resultado de este proceso. Posteriormente se

[1] http://www.computer.org/sevocab

establece que los principios que el diseño de software debe perseguir incluyen: abstracción, emparejamiento (*coupling*) y cohesión, descomposición y modularización, encapsulación y ocultamiento de información, separación de interface e implementación, suficiencia y completud y separación de incumbencias (*separation of concerns*). Dentro de esta área se menciona también el control de acceso como un concepto fundamental de seguridad. Se mencionan también una variedad de atributos que contribuyen a la calidad del diseño de software: mantenibilidad, portabilidad, capacidad de ser probado (*testability*), usabilidad, corrección y robustez [1].

En el área **Construcción de *software*** se hace referencia a la misma como la creación detallada de *software* funcional mediante una combinación de código, verificación, pruebas unitarias, pruebas de integración y depuración (*debugging*). Cabe destacar que en esta área de conocimiento, la reducción de complejidad es alcanzada mediante el énfasis en la creación de código que simple y legible más que astuto (*clever*). Esto también se logra mediante el uso de estándares, diseño modular y numerosas otras técnicas específicas. Algunas de estas técnicas que soportan la construcción pensada en verificación incluyen la adopción de estándares de codificación, revisiones de código, pruebas unitarias, organización de código para facilitar la automatización de pruebas y la restricción en el uso de de estructuras de lenguaje complejas o de difícil comprensión.

El SWEBOK define el **manejo de configuración de *software*** como la disciplina de identificar la configuración de un sistema en distintos puntos en el tiempo, con el propósito de controlar sistemáticamente los cambios en la configuración, para mantener la integridad y trazabilidad de la configuración durante el ciclo de vida del sistema. En este punto se menciona también la importancia de definir y comunicar estrategias de ramificación y consolidación (*branching* y *merging*), ya que esto afectará profundamente al desarrollo del proyecto. Cabe destacar que en este punto el SWEBOK hace referencia a la integración continua como una práctica común en muchos acercamientos a desarrollo de *software*, caracterizada por ciclos frecuentes de construcción-pruebas-entrega (*build-test-deploy*).

B. Arquitectura de software

Cada vez es más frecuente que las aplicaciones requeridas por clientes

empresariales deban responder a necesidades que hace unos años eran muy raras o no existían: deben ser independientes de los sistemas operativos y gestores de bases de datos; ser accedidas desde lugares geográficamente distantes; interactuar con otros sistemas ya en funcionamiento; atender grandes volúmenes de interacciones por unidad de tiempo, originadas por usuarios que utilizan la aplicación simultáneamente, entre otras necesidades.

Las aplicaciones desarrolladas han pasado de un enfoque centralizado y monolítico hacia entornos distribuidos y heterogéneos, lo que conlleva un aumento de la complejidad desde el punto de vista estructural. Por ello ha tenido un auge significativo la arquitectura de software como rama de la ingeniería de software, al tiempo que ha cobrado una gran importancia.

De acuerdo con Clements, P. [13], en el contexto de *software* podemos llamar **arquitectura** a los componentes que comprenden un sistema, la especificación del comportamiento para esos componentes y los mecanismos e interacciones entre ellos. Usualmente un sistema está compuesto de más de un tipo de componentes: módulos, tareas, funciones. La tarea de la arquitectura comprende elegir qué tipo de componentes son más apropiados para cada necesidad.

Martin, B. [11] afirma que la arquitectura de un sistema está determinada por la experiencia de usuario (*user experience*) que se desee, y no por el dominio del problema. De esta forma, un sistema puede utilizar una arquitectura **petición/respuesta** (*request/response*) como lo hacen la mayoría de sitios y aplicaciones web, o un una arquitectura **determinada por eventos** (*event driven system*) como la mayoría de juegos de computadora. Es de notar que la elección de la arquitectura tendrá un profundo efecto en la forma y el diseño del software.

Cabe citar también a Romero, P., quién define la arquitectura de software como el conjunto de decisiones de diseño que definen la estructura fundamental de un sistema de cómputo. Contemplando las partes componentes del sistema: sus interfaces, comportamiento, las relaciones entre ellas, las relaciones de ellas con el medio y la evolución de cada componente junto al sistema [17].

Dentro de la gama de posibles arquitecturas es necesario mantener la perspectiva de cuáles son los aspectos más importantes que se desean abordar, es decir, que dan mayor valor al producto. La mayoría de las aplicaciones contiene funcionalidades comunes; se pueden citar, entre otras: almacenamiento en memoria estática, comunicaciones, tratamiento de la interfaz de usuario, errores, transacciones, mantenimiento de la seguridad, etc.

Martin, B. hace notar que la tendencia de la mayoría de arquitecturas de sistemas se ha orientado históricamente a la separación de intereses (*separation of concerns*), conseguida mediante la división del *software* en capas: reglas de negocio, interfaz gráfica, interfaz con otros sistemas, acceso a datos persistentes (base de datos), etc. [10].

La idea anterior ha promovido la solución de reutilizar el código, tener unidades de trabajo diferentes que implementan las funcionalidades que pueden ser comunes a diferentes aplicaciones o a diferentes partes de una misma aplicación. Esto hace necesario que usemos el término **esquema**, que para este trabajo ha de entenderse como un conjunto de clases relacionadas que brindan, de manera unificada, las funcionalidades necesarias para implementar un servicio específico. Se busca en general desarrollar aplicaciones complejas con alto valor agregado, contando con pocos recursos: especialmente tiempo de desarrollo.

Es dentro de este marco de referencia que introducimos los conceptos descritos a continuación.

1. Aplicaciones web. Según el Dictionary of Multimedia & Internet Applications [23] se denomina **aplicación web** a aquellas herramientas que los usuarios pueden utilizar accediendo a través de un navegador web (*web browser*) a un servidor web (*web server*).

La arquitectura de un servidor web mantiene la conexión entre el procesamiento del lado del servidor y de datos del lado del cliente, y cuando conviene a través de herramientas específicas (controles ActiveX) puede realizar procesamiento también del lado de cliente. La arquitectura tradicional bajo la que se implementan este tipo de aplicaciones es la de **petición/respuesta**.

La popularidad de las aplicaciones web se debe en gran parte a la omnipresencia (*ubiquity*) de los navegadores web, que han introducido una capa de abstracción sobre los dispositivos, de forma que las aplicaciones dejan de estar sujetas al sistema operativo. Esto presenta ventajas y desventajas con respecto a arquitecturas de software de escritorio (o específicos de plataforma):

- Por su naturaleza, clientes usando distintos sistemas operativos pueden acceder de forma similar a la aplicación.
- La velocidad de respuesta de la aplicación se ve reducida, dado el nivel de abstracción adicional, con respecto a una aplicación de escritorio.
- El uso de tecnologías variadas para manejar los distintos aspectos de una aplicación (lógica de negocio, persistencia de información, interfaz de usuario, etc.) agrega complejidad al desarrollo, aunque abre al mismo tiempo una gama más amplia de posibilidades.

2. La nube y el *cloud computing*. La **computación en la nube** (*cloud computing*) ha sido definida por el Instituto Nacional de Estándares y Pruebas (NIST[2], por sus siglas en inglés) y la Alianza de Seguridad Cloud (CSA[3], por sus siglas en inglés) como: "un modelo que permite acceso conveniente y sobre demanda a un conjunto compartido de recursos informáticos configurables (por ejemplo, redes, servidores, almacenamiento, aplicaciones y servicios) que pueden ser rápidamente aprovisionados y liberados con mínimo esfuerzo de administración o servicio de interacción con el proveedor" [18].

Por otro lado, Carstensen, *et. al.*, citando al NIST[4], indica las siguientes características para el *cloud computing* [3].

1. Características esenciales que un entorno de *cloud computing* completo ha de integrar:
    a. Autoservicio a demanda: la capacidad de prestación de servicios de computación a los consumidores, como tiempo de almacenamiento y

[2] http://www.nist.gov/
[3] https://cloudsecurityalliance.org/
[4] http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf

servidores de red, según sea necesario de forma automática sin necesidad humana de interacción con cada proveedor de servicio.

b. Amplio acceso a la red: las capacidades están disponibles en la red y se accede a través de mecanismos estándar que promueven el uso de plataformas de clientes heterogéneos (por ejemplo, teléfonos móviles, tablets, portátiles y estaciones de trabajo).
c. Puesta en común de recursos: los recursos informáticos del proveedor se agrupan para servir a múltiples consumidores utilizando un modelo multi-hilo, con diferentes recursos físicos y virtuales de forma dinámica asignada y reasignada de acuerdo con la demanda del consumidor.
d. Rápida elasticidad: las capacidades pueden provisionarse elásticamente y/o liberarse, en algunos casos automáticamente, para escalar rápidamente hacia afuera y hacia adentro con la demanda. Para los consumidores, las capacidades disponibles para la provisión a menudo parecen ser ilimitadas y puede ser asignadas en cualquier cantidad en cualquier momento.
e. El servicio medido: el sistema controla automáticamente y optimiza el uso de recursos mediante el aprovechamiento de su capacidad de medición en algún nivel de abstracción apropiado para el tipo de servicio (por ejemplo, almacenamiento, procesamiento, ancho de banda, y las cuentas de usuario activas). El uso de recursos puede ser supervisado, controlado, y reportado, proporcionando transparencia tanto para el proveedor y consumidor del servicio utilizado.

2. Modelos de servicio en que se desarrolla el *cloud computing*:
    a. *Software-as-a-Service* (SaaS), se encuentra en la capa más alta y denota una aplicación completa ofrecida como un servicio.
    b. *Platform-as-a-Service* (PaaS), representa la capa intermedia. Es la encapsulación de un ambiente de desarrollo y el empaquetamiento de una serie de módulos o complementos que proporcionan una funcionalidad horizontal (persistencia de datos, autenticación, mensajería, etc.).
    c. *Infrastructure-as-a-Service* (IaaS), se encuentra en la capa inferior y es un medio de entregar almacenamiento básico y capacidades de cómputo como servicios en la red.
3. Modelos de implementación de *cloud computing*:

a. Nube privada: la infraestructura de nube está preparada para el uso exclusivo de una sola organización que comprende varios consumidores (por ejemplo, unidades de negocio). Puede ser de propiedad de, gestionada y operada por la organización, un tercero, o alguna combinación de ellos, y que puede existir dentro o fuera de las instalaciones.
b. Nube comunitaria: la infraestructura de nube está preparada para el uso exclusivo de una determinado comunidad de consumidores de las organizaciones que han compartido requerimientos (por ejemplo de seguridad, políticas o de rendimiento). Puede ser de propiedad de, administrada y operada por una o más de las organizaciones en la comunidad, un tercero, o alguna combinación de ellos, y que pueden existir dentro o fuera de las instalaciones.
c. Nube pública: la infraestructura de nube está preparada para el uso abierto al público en general. Puede ser propiedad de, administrada y operado por una empresa, una institución académica u organización gubernamental, o alguna combinación de ellos. Existe en las instalaciones del proveedor de la nube.
d. Nube híbrida: la infraestructura de la nube es una composición de dos o más nubes distintas de infraestructura (privada, comunitaria o pública) que siguen siendo entidades únicas, pero que están unidas por la tecnología estandarizada o propietaria que permite portabilidad de datos y aplicación (por ejemplo, la disolución de una nube en varias, para distribuir y balancear la carga, es decir *load balancing*).

3. ***Software-as-a-service*** **(SaaS).** El *software* como servicio (*SaaS*, por sus siglas en inglés), es la primera capa de los tres modelos del *cloud computing*. Es la capacidad ofrecida al consumidor de utilizar las aplicaciones del proveedor que se ejecutan en una infraestructura *cloud*. Las aplicaciones son accesibles desde diferentes dispositivos cliente a través de una interfaz de cliente liviano como un navegador web (por ejemplo, email basado en web). El consumidor no gestiona ni controla la infraestructura de nube en donde se ejecuta la aplicación (la red, servidores, sistemas operativos, almacenamiento o incluso capacidades de aplicaciones individuales). La posible excepción son los ajustes de configuración de aplicaciones específicas para el usuario.

El SaaS ha sido definido también por Saugatuck Tecnología como la entrega y el uso de la aplicación o funcionalidad de *software* de infraestructura a través de una red. SaaS se utiliza en función de la demanda, y puede ser vendido, entregado y pagado a través de un modelo de suscripción, con carácter de 'pago por uso', o, más frecuentemente, en una combinación de modelos que permitan el uso de tipo servicio [18].

La Asociación de Auditoría y Control de Sistemas de Información (ISACA[5], por sus siglas en inglés), ha identificado algunos de los beneficios clave de negocio que ofrece SaaS, incluyendo la reducción de costos ya que la nube ofrece a las empresas la opción de escalabilidad sin el compromiso financiero necesario para la compra y mantenimiento de infraestructura [18]:

- Inmediatez: muchos usuarios iniciales de la computación en la nube han citado la capacidad de suministro y de la ventaj de poder utilizar un servicio en un solo día.
- Disponibilidad: los proveedores de *cloud* tienen la infraestructura y ancho de banda para dar cabida a requisitos de negocio para el acceso de alta velocidad, almacenamiento y aplicaciones.
- Escalabilidad: con capacidad ilimitada, los servicios en la nube ofrecen una mayor flexibilidad y escalabilidad para evolucionar la tecnología de la información (TI) a diferentes necesidades.
- Eficiencia: La reasignación de las actividades operacionales de gestión de la información a la nube ofrece a las empresas una oportunidad única para centrar

[5] https://www.isaca.org/Pages/default.aspx

los esfuerzos en la innovación y la investigación y desarrollo. Esto permite el crecimiento del negocio y del producto y puede ser aún más beneficioso que las ventajas financieras que ofrece la nube.

- Capacidad de recuperación: proveedores de la nube han reflejado soluciones que pueden ser utilizados en un escenario de desastre, así como para el tráfico de equilibrio de carga.

También se han identificado algunos riesgos potenciales de SaaS que tendrán que ser gestionados:

- Las empresas tienen que ser cuidadosas en la elección del proveedor, cosas como la reputación del mismo, la historia y la sostenibilidad de todo deben ser factores importantes a considerar.
- El proveedor de SaaS a menudo toma la responsabilidad por el manejo de la información, que es una parte crítica del negocio. El incumplimiento de los niveles de servicio acordados puede afectar no sólo la confidencialidad, sino también la disponibilidad.
- La naturaleza dinámica de SaaS puede dar lugar a confusión en cuanto a la ubicación en donde información reside en realidad. Cuando se requiere la recuperación de información, esto puede crear retrasos.
- El acceso de terceros a la información sensible crea un riesgo de compromiso a la información confidencial. En la computación en nube, esto puede representar una amenaza significativa para asegurar la protección de la propiedad intelectual y los secretos comerciales.
- Debido a la naturaleza dinámica de la nube, la información puede no ser inmediatamente localizada en el caso de un desastre. La continuidad del negocio y los planes de recuperación de desastres deben estar bien documentados y probados.
- El cumplimiento de las regulaciones y leyes en diferentes regiones geográficas puede ser un reto para las empresas. En este momento, hay poco precedente legal relativo a la responsabilidad en la nube.

4. Arquitectura Modelo-Vista-Controlador (MVC). El diseño de arquitecturas de software marca un cambio de punto de vista en el proceso de desarrollo. En comparación con los procesos de recolección de requisitos, y el diseño de datos e hipertexto, que se centran en la especificación progresiva de la aplicación, el diseño de una arquitectura se centra en la elección de los componentes de hardware, red y *software* que componen el sistema, para encontrar la combinación de estos componentes que mejor cumple con los requisitos de la aplicación, y que al mismo tiempo respete las condiciones técnicas y económicas del proyecto [4].

La definición de la arquitectura de la aplicación debe garantizar la consecución del adecuado nivel de servicio con respecto a las siguientes dimensiones:

- Rendimiento: la aplicación debe sostener la carga de trabajo prevista, expresada por parámetros como el número máximo de usuarios simultáneos, el número de solicitudes de página sirve por unidad de tiempo, o el tiempo máximo para la entrega de una página al cliente.
- Escalabilidad: la arquitectura debe ser extensible, de modo que, cuando la carga de trabajo aumenta, es posible añadir más potencia de cálculo y mantener estable el rendimiento.
- Disponibilidad: la aplicación debe trabajar de forma continua, y los fallos no debe afectar de manera significativa el servicio prestado a los usuarios. Idealmente, el fallo de cualquiera de los componentes de la arquitectura debe ser tolerado y no interrumpir el servicio.
- Mantenimiento Estado: el estado de la interacción con el usuario (representado, por ejemplo, por los datos de la sesión se mantienen del lado del servidor) deben ser preservados, incluso cuando la aplicación se distribuye en varias máquinas o fallos ocurren.
- Seguridad: la información alojada en el nivel de datos y de transmisión entre la aplicación y sus usuarios debe ser protegido, y los usuarios deben identificarse y concede el acceso sólo al contenido y las funciones a las que tienen derecho.

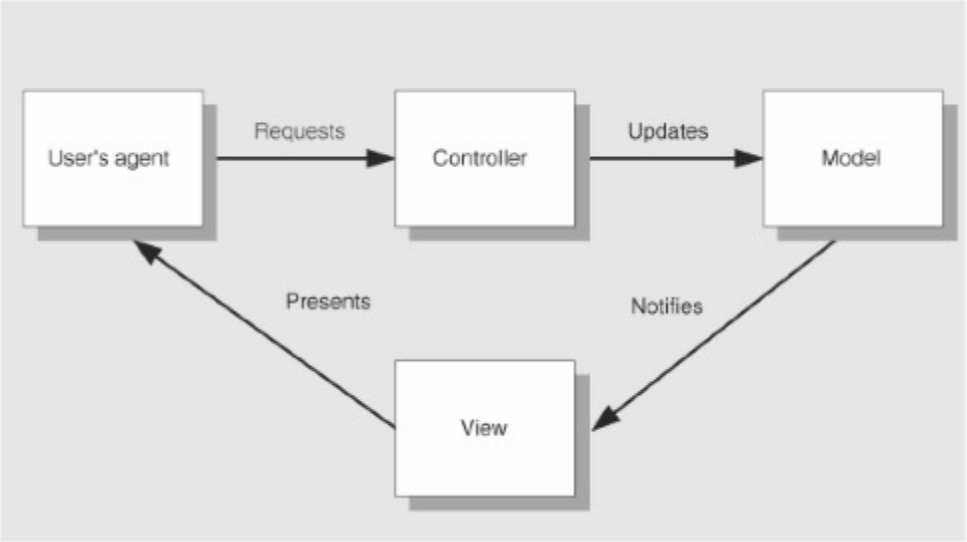
Una de las más poderosas arquitecturas de software propuestos por los ingenieros de software para hacer frente a estos problemas es el llamado patrón de diseño Modelo Vista Controlador (MVC, para abreviar).

El MVC se concibe para separar mejor y aislar las tres funciones esenciales de una aplicación interactiva:

- La lógica empresarial de la aplicación (el modelo)
- La interfaz de presentación al usuario (la vista)
- El control de la interacción provocada por las acciones del usuario (el controlador)

La **Figura No.1** resume brevemente los elementos y la interacción en una arquitectura MVC.

Figura No.1. Elementos e interacción MVC



La operación El flujo de la operación se activa por la petición de para de un usuario para un cierto contenido o servicio:

- La solicitud es interceptada por el controlador, que es el responsable de decidir qué medidas tomar para servirla.
- El controlador envía la solicitud, en la forma de una *solicitud de acción*, al componente adecuado del modelo.
- El modelo incorpora la lógica de negocio para llevar a cabo la acción, y ejecuta esta lógica, que actualiza el estado de la aplicación y produce un resultado que debe comunicarse al usuario.
- El cambio en el modelo define la vista más apropiada, la cual a su vez define la presentación de la respuesta. Dicha presentación normalmente implica objetos en interacción, por lo que el usuario puede plantear una nueva solicitud y reactivar

nuevamente el proceso.

La arquitectura MVC define una distinción clara de las responsabilidades entre los componentes de la aplicación:

- El modelo encapsula lógica de negocio requerida para la respuesta a la solicitud de un usuario y mantiene el estado de la aplicación. El modelo debe ignorar el formato en el que se plantean las peticiones y la manera en que se construye la respuesta y la presenta al usuario.
- La vista incorpora la lógica de presentación para el montaje de la interfaz de usuario. Una aplicación puede tener una vista única o varias vistas, y cada vista puede a su vez estar compuesto de subvistas, correspondientes a los diferentes tipos de resultados. La vista debe obviar la naturaleza de los resultados para presentar, así como los detalles de la solicitud se origina este tipo de resultados.
- El controlador se encarga de unir los otros componentes de la arquitectura correctamente. Es el responsable de interpretar la petición del usuario, de ejecutar la solicitud apropiada para la acción, de examinar el resultado de cada acción, y de decidir qué hacer a continuación. El controlador es totalmente inconsciente de la lógica de negocio de la actuación que invoca, y de la lógica de presentación de la Vista.

Las acciones son los componentes reales que implementan la lógica de negocio. Toda la estructura está diseñada para ser reutilizable en diferentes aplicaciones, posiblemente utilizando diferentes interfaces.

Cabe destacar que esta arquitectura es mencionada por el SWEBOK como una forma efectiva de mantener la presentación de la información separada de la información misma [1].

C. Metodologías y desarrollo

1. Métodos ágiles. Al tratar de entender el significado de la palabra **ágil** en el contexto de los métodos ágiles, autores como Larman (2003) y Hunt (2005) afirman que el ser ágil es ser flexible y rápido para responder a los cambios. Highsmith (2002) afirma que la agilidad significa ser capaz tanto crear como de responder al cambio con el fin de obtener beneficios en un entorno empresarial turbulento. Por otro lado, Shore y Warden (2008) entienden que ser ágil consiste en seguir una filosofía ágil que tiene la mejora de la productividad como consecuencia de una forma diferente de trabajo, y no sólo trabajar más rápido. Shore y Warden destacan que los métodos ágiles son procesos que apoyan la filosofía ágil. De acuerdo con Shore y Warden, con el fin de ser ágil, es necesario poner realmente los valores y prácticas ágiles en el trabajo. Shore y Warden (2008) recuerdan que, con el fin de seguir una filosofía ágil, es importante tener en cuenta el equilibrio entre las necesidades de cada grupo de interés de un proyecto de *software*. Utilizan el término *éxito* para referirse a la satisfacción de esas necesidades. Según ellos, el desarrollo ágil propone alcanzar el éxito organizacional, personal y técnico [22].

Los principios del desarrollo ágil de *software* que han de seguirse y adoptarse emergieron de los principios tradicionales de desarrollo de *software* y de diversas experiencias sobre la base de los éxitos y fracasos en los proyectos de *software*. Es un hecho que los clientes tuvieron dificultades para definir sus necesidades debido a la tecnología, que cambia rápidamente. Los nuevos métodos, ahora llamados métodos ágiles, se han diseñado para definir los requisitos cambiantes en entornos de *software* [14].

En febrero de 2001, los académicos y expertos de la industria del *software* se reunieron en Utah, Estados Unidos, con el fin de discutir los valores y principios que faciliten un desarrollo de *software* más rápido y responde a los cambios que puedan surgir durante el proyecto. La idea era ofrecer una alternativa a los procesos de desarrollo tradicional. Como resultado de esta reunión fue concebida la Alianza Ágil[6]. Esta es una organización sin fines de lucro dedicada a promover los conceptos relacionados con el desarrollo ágil de *software* y ayudar a las organizaciones a adoptar dicho conceptos. El

[6] http://www.agilealliance.org

resultado de esta reunión es conocido en inglés como *Agile Manifiesto*[7].

Las metodologías para el desarrollo ágil de *software* se basan fundamentalmente en la colaboración con los usuarios de *software* durante todo el proceso de desarrollo, la sencillez de adaptar el producto a los cambios en los requisitos y en la entrega del producto incrementalmente. Estas metodologías se basan en el **Manifiesto Ágil**, una serie de postulados que han sido aceptados y se utilizan con éxito en proyectos en los que se desconocen en un primer momento los requisitos detallados, que se han identificado durante el proceso de desarrollo de las interacciones con los usuarios y las votaciones obtenidas de este modo [2].

El ciclo de desarrollo aplicada por Metodologías Ágiles es iterativo e incremental. Este modelo permite que el *software* que se entrega en piezas pequeñas y utilizables, conocidos como incrementos. Cada iteración se puede considerar como un pequeño proyecto en el que se llevan a cabo actividades como requisito, análisis, diseño, implementación y pruebas con el objetivo de producir un subconjunto del sistema final. El proceso se repite varias veces produciendo un nuevo incremento en el ciclo cada vez hasta que se terminó el producto completo.

Aunque todas las metodologías ágiles adoptan este ciclo, cada una de ellas presenta sus propias características. Las metodologías ágiles más comúnmente utilizados son Scrum, Crystal Methodologies, Dynamic Systems Development Method (DSDM), Adaptive Software Development (ASD), Feature-Driven Development (FDD), Extreme Programming (XP).

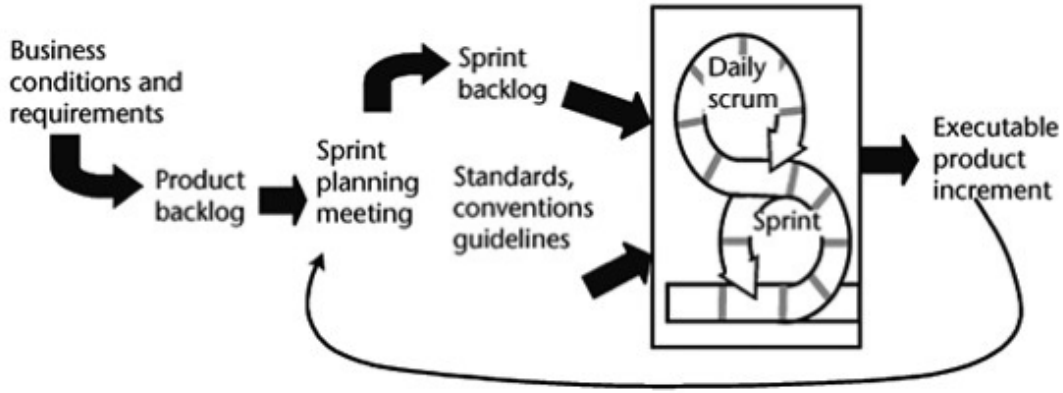
Las metodologías ágiles tienen sus propias características y cada una hace hincapié en aspectos específicos. La primera y última mencionadas anteriormente, Scrum y XP, son metodologías que promueven cuestiones tales como el trabajo en equipo, lo que favorece las relaciones interpersonales entre sus miembros, aumentando la relación fluida con el cliente y la generación de la documentación mínima que aporten valor al proyecto.

[7] http://agilemanifesto.org

2. **Scrum, modelo de desarrollo ágil**. Scrum no trata principalmente sobre el desarrollo de *software*. Es un método para gestionar el desarrollo de productos que se puede adaptar a cualquier tecnología específica, incluyendo el *software*. Scrum, tal como existe hoy en día, creció desde sus inicios en Japón a mediados de 1980. El nombre **Scrum** proviene de la práctica del rugby y se refiere a una estrategia utilizada para conseguir una bola en juego [9].

El proceso de Scrum, mostrado gráficamente en la **Figura No.2**, es gradual e iterativo, al igual que otros métodos ágiles.

Figura No.2. Proceso Scrum



De acuerdo a lo publicado por *Scrum Alliance*[8], una organización que promueve y apoya la adopción generalizada y la práctica efectiva de Scrum, este se aplica a los tipos de trabajo que las personas han encontrado difíciles de manejar con procesos definidos, ya que incluyen requisitos inciertos, combinados con riesgos impredecibles en la tecnología de aplicación. Al decidir si se debe aplicar Scrum, a diferencia de los enfoques impulsados por la planificación tales como los descritos en la Guía del PMBOK[9], se recomienda tener en cuenta si el trabajo depende de la creación de conocimiento y colaboración. Por ejemplo, Scrum no fue pensado para los tipos de trabajo repetitivos de producción o de servicios. También es aconsejable considerar si existe un compromiso suficiente para fomentar un equipo auto-organizado.

[8] https://www.scrumalliance.org/
[9] http://gio.uniovi.es/documentos/software/GUIA_PMBok.pdf

Las actividades o reuniones de **Scrum** que se llevan a cabo son [9]:

- Planificación de iteración (*sprint planning*): durante esta planificación el cliente presenta al equipo la lista de requisitos priorizada del producto o proyecto (*product backlog*). El equipo resuelve dudas con el cliente y selecciona los requisitos que se compromete a completar en la iteración. Luego el equipo elabora la lista de tareas de la iteración (*sprint backlog*) necesarias para desarrollar los requisitos a que se ha comprometido. La estimación de esfuerzo se hace de manera conjunta y los miembros del equipo se autoasignan las tareas.
- Ejecución de la Iteración (*daily scrum*): cada día el equipo realiza una reunión de sincronización. Cada miembro del equipo inspecciona el trabajo que el resto está realizando para poder hacer las adaptaciones necesarias que permitan lograr el objetivo. En cada reunión el equipo responde a tres preguntas:
    - Qué he hecho desde la última reunión de sincronización
    - Qué voy a hacer a partir de este momento
    - Qué impedimentos tengo o voy a tener
- Durante la iteración el facilitador (*scrum master*) se encarga de que el equipo pueda cumplir con su compromiso, y de que su productividad se mantenga.
- Inspección y adaptación (*sprint review*): el último día de la iteración se realiza la reunión de revisión de la iteración (*sprint retrospective*). Tiene dos partes:
- Demostración (*product grooming*): el equipo presenta al cliente los requisitos completados de la iteración, en forma de incremento de producto preparado para ser entregado con el mínimo esfuerzo.
- Retrospectiva (*retrospective*): el equipo analiza cómo ha sido su manera de trabajar y cuáles son los problemas que podrían impedirle progresar adecuadamente, mejorando de manera continua su productividad. El *scrum master* se encargará de ir eliminando los obstáculos identificados.

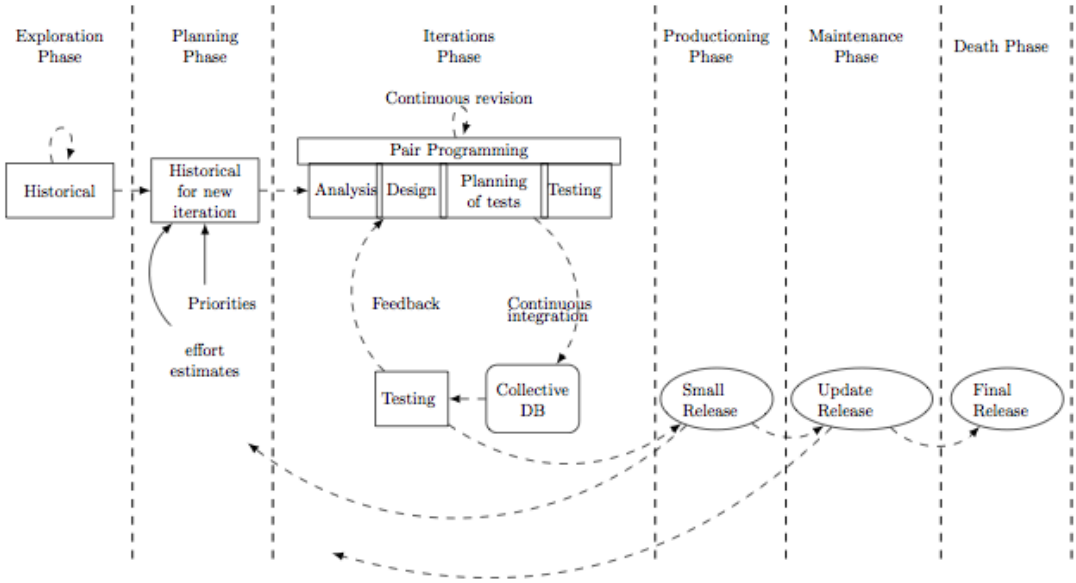
Scrum prescribe también tres reportes de utilidad para el manejo de proyectos: *sprint burndown*, *release burndown* y *velocity* [15].

- El reporte *sprint burndown* es utilizado para indicar diariamente el progreso alcanzado en la consecución de una iteración.
- El reporte de velocidad (*velocity report*) indica la cantidad de esfuerzo que puede llevar a cabo en un *sprint*.
- El reporte *release burndown* provee información acerca de cuánto trabajo se

encuentra pendiente para la entrega (*release*) y cuando trabajo se está realizando durante cada sprint. Su presentación consiste en reportar el estado del *product backlog* al inicio de cada iteración.

3. ***eXtreme Programming (XP)***. Es un enfoque iterativo para el desarrollo de *software*, descrito gráficamente en la **Figura No.3**. Esta metodología está diseñada para ofrecer el *software* que el cliente necesita, cuando se necesita. Esta metodología hace hincapié en el trabajo en equipo. Los gerentes, clientes y desarrolladores, son parte de un equipo dedicado a ofrecer *software* de calidad. XP implementa un modo simple, pero eficaz para permitir el desarrollo del estilo de trabajo en grupo y mejora un proyecto de *software* en cuatro aspectos esenciales; la comunicación, la sencillez, la retroalimentación y el coraje [6].

Figura No.3. Enfoque iterativo de desarrollo XP



XP se enfoca en manejar los cambios al código de forma eficiente, llevar la programación a un ritmo sistemático, trabajar en base a solicitudes "cortas" del cliente y no en base a un plan maestro y en enfocar el trabajo del equipo a las necesidades actuales del cliente.

Los valores que promueve son: comunicación, que permite ajustar rápidamente el cambio; realimentación (*feedback*), es decir realizar consultas al cliente para realizar modificaciones, y mantener pruebas unitarias que informen sobre el estado del código debidos a los cambios; simplicidad, que implica buscar que se implemente solo que tiene que ser implementado de la forma más simple posible mediante la aplicación constante de refactorización del código; coraje o valentía, que pretende hacer ver los errores y ser persistente para corregirlos; respeto que se manifiesta en que los miembros del equipo no pueden hacer cambios que hagan fallar las pruebas o que demoren el trabajo de sus compañeros.

4. **Desarrollo orientado a pruebas (TDD)**. Dogša T, *et. al.* define el desarrollo orientado a pruebas (TDD) como una práctica de desarrollo de software donde los casos de prueba son escritos incrementalmente previo a la implementación del código de producción [7]. En su informe *The effectiveness of test-driven development: an industrial case study*, indica que sus resultados ponen de relieve que TDD promueve una calidad de código mayor y que es más fácil de mantener, aunque hace mención a una disminución en su nivel de productividad.

Martin, B. indica en su su artículo *The Land that Scrum Forgot* que durante la adopción de **XP** es recomendable iniciar aplicando TDD debido a que una base de código sin pruebas es ya de por sí poco organizada, sin importar cuán limpia pueda estar en otros sentidos [12]. En este artículo hace una breve comparación entre las prácticas desarrollo y de manejo de contabilidad: así como en la contabilidad todo es ingresado dos veces para efectos de corroboración (*dual entry bokkeeping*), en la práctica del desarrollo esta práctica es TDD.

D. Tecnologías

1. **Entornos de integración continua (CI)**. La integración continua es una práctica de desarrollo de software en donde los miembros de un equipo integran su trabajo con frecuencia. Por lo general cada persona integra al menos una vez al día, lo que conduce a múltiples integraciones por día. Cada integración es verificada por un sistema automatizado (incluyendo prueba) para detectar errores de integración lo más rápidamente posible. Muchos equipos encuentran que este enfoque conduce a la reducción de manera significativa los problemas de integración y permite que un equipo desarrolle *software* cohesivo más rápidamente [8].

En particular, el aumento de la Programación Extrema (XP) y Test Driven Development (TDD) ha popularizado el código de con pruebas incluidas, y como resultado muchas personas han visto el valor de la técnica. Las pruebas incluidas dentro del código tienen que ser capaces de hacerse con un simple comando con el objetivo de comprobar el estado del código. El resultado de ejecutar el banco de pruebas debería indicar si las pruebas fracasaron.

2. ***Frameworks* y librerías de componentes**. De acuerdo con Coronado, D. un *framework* es una colección de librerías, clases y recursos que forman una base de soporte para el desarrollo de software. Estos *frameworks* ayudan a que el inicio de una aplicación sea fácil y rápido, ya que proveen la fundación base necesaria para convertir rápidamente las ideas escritas en la pizarra a una aplicación funcional con código listo para producción. Un *framework* moderno, flexible y extensible es casi tan esencial como el propio lenguaje de programación para el desarrollador web de hoy en día, especialmente cuando los dos se complementan de manera que se convierten en herramientas poderosas [5].

De acuerdo con Tojin, K., a pesar de que el uso de un *framework* o enfoque estructurado para el desarrollo de software puede parecer restrictivo en un principio, a mediano plazo permite que el esfuerzo del desarrollo se centre en las tareas más complicadas, permitiendo así realizar el trabajo de una forma más rápida y eficiente. Además, el uso de modelos estandarizados y de buenas prácticas garantiza la estabilidad y facilidad del mantenimiento y actualización, así como la escalabilidad de las

aplicaciones que se desarrollan [21].

a. ***Symfony2***. Symfony2 es un framework de desarrollo web escrito en PHP, y una comunidad de desarrollo. Symfony2 fue desarrollado por Fabien Potencier como un marco de trabajo consistente en una caja de herramientas o toolbox, es decir un conjunto de componentes de software prefabricados y rápidamente integrables para montaje de aplicaciones.

Symfony2 es un proyecto de PHP de software libre que permite crear aplicaciones y sitios web rápidos y seguros de forma profesional.

Entre sus principales características encontramos que:

- Su código y el de todos sus componentes y librerías que incluye, se publican bajo la licencia MIT de software libre.
- La documentación del proyecto también es libre e incluye varios libros y decenas de tutoriales específicos.
- En producción, las aplicaciones **Symfony2** solamente necesitan permiso de escritura en dos directorio internos de la propia aplicación. Además incluye varias herramientas gráficas y de consola para depurar fácilmente errores que se produzcan en las aplicaciones.
- Para evitar el uso de contraseñas en archivos de configuración, Symfony permite establecer parámetros de configuración de las aplicaciones a través de variables de entorno del propio servidor.
- Dispone de un programa de certificación para validar los conocimientos de los programadores de **Symfony2**.

Una de las ventajas introducidas por **Symfony2** sobre varios otros *frameworks* de trabajo **PHP** es la introducción de un inyector de dependencias (*dependency injection*[10]), un patrón de diseño orientado a objetos en el que se suministran objetos a una clase en lugar de ser la propia clase quien crea los objetos. Este patrón, implementado dentro de **Symfony2**, tiene la ventaja de permitir extender e incluso sobreescribir mucha de la funcionalidad introducida por el framework de una manera relativamente flexible.

[10] http://www.martinfowler.com/articles/injection.html

3. **Object relational mapping (ORM)**. De acuerdo con Tojín, K. el mapeo objeto-relación es una técnica de programación que convierte datos entre sistemas incompatibles a objetos de lenguajes orientados a objetos. Con esta técnica se crea un objeto virtual de base de datos que puede ser utilizado dentro del lenguaje de programación.

ORM permite acceder y manipular objetos sin tener que considerar o preocuparse por cómo estos objetos se relacionan a sus fuentes. ORM permite a los desarrolladores mantener una vista consistente de objetos en el tiempo, incluso cuando se obtienen de fuentes de datos distintas.

Basado en la abstracción, **ORM** maneja el mapeo de detalles entre un conjunto de objetos y una base de datos relacional, XML, repositorios u otras fuentes de datos, mientras que se encarga de ocultar detalles relacionados a la interface de conexión a los desarrolladores [21].

La herramienta **Symfony2** trabaja principalmente con dos herramientas **ORM** fundamentalmente distintos: **Propel** y **Doctrine**. La primera implementa la abstracción **ORM** mediante el patrón de diseño *active record*[11], en donde un objeto que envuelve un registro en una tabla de la base de datos o vista encapsula el acceso a la base de datos y agrega lógica de dominio sobre los datos. La segunda herramienta implementa esta abstracción por medio del patrón de diseño *unit of work*[12], en donde se mantiene una lista de objetos afectados por una transacción y se coordina la escritura de dichos cambios y la resolución de problemas de concurrencia.

[11] http://www.martinfowler.com/eaaCatalog/activeRecord.html
[12] http://www.martinfowler.com/eaaCatalog/unitOfWork.html

IV. Análisis

A. Análisis del negocio

La estrategia que la empresa ha seguido durante los años en los que ha desarrollado *software* activamente en el mercado, está marcada por el desarrollo de un motor central en torno al cual giran las distintas aplicaciones o módulos que mercadea a sus clientes.

El planteamiento presentado durante las prácticas consiste en detectar y dar solución a problemas tecnológicos y de ingeniería, que se presentan tanto en la empresa como organización como dentro de sus productos. A continuación se describen algunos de los mismos.

B. Problemas con tecnologías

Aunque inicialmente el entorno de trabajo giró alrededor del sistema operativo más popular del momento, MSDOS, muchos de los productos y el trabajo realizado por el equipo de desarrollo actualmente se encuentra sobre la plataforma **Visual FoxPro**, de Microsoft. Si bien esta plataforma cuenta con mantenimiento hasta el 2015, Microsoft ha anunciado que no se publicarán nuevas versiones [20]. Esto presenta para la empresa una desventaja competitiva en el mediano plazo, dado que sus productos tienen el riesgo de no ser soportados o no tomar provecho de las nuevas plataformas o avances que implementen en el mercado (tales como la arquitectura de 64 bits o los procesadores multinúcleo).

Por otro lado, se detectó y se hizo notar que la tecnología utilizada actualmente para el almacenamiento de información (MySQL con motor de almacenamiento MyISAM) no provee las garantías de integridad referencial requeridas para sistemas con abundantes referencias y amarre entre tablas[13]. Si bien este problema puede mitigarse teniendo aplicando validaciones y medidas especiales con la manipulación de datos, esto necesariamente tiene impactos en el rendimiento y puede dar lugar a serios problemas de

[13] http://dev.mysql.com/tech-resources/articles/mysql-enforcing-foreign-keys.html

integridad.

Finalmente, el análisis de la estructura de base de datos actual denota problemas de normalización que han impactado la cantidad de campos usados en cada tabla, por lo que cada tabla termina heredando los campos de la llave de las tablas con las que se relaciona. Esto presenta al menos dos problemas iniciales: agrega complejidad a la documentación del sistema de cara a los desarrolladores y demanda más procesamiento, puesto que requiere consultas de validación con múltiples campos.

C. Problemas de ingeniería

Dentro de la empresa se ha notado la ocurrencia relativamente frecuente de problemas en el desarrollo del código según los cuales la adición de una característica o funcionalidad, o la resolución de un problema, resulta en la aparición de un problema nuevo o en la reaparición de un problema resuelto anteriormente, tras lo cual se obtienen inconformidades de los clientes con eventuales actualizaciones sobre el *software*.

Otra característica negativa detectada es la falta de documentación interna en los sistemas de cara a los desarrolladores, tanto en cuanto al código (*inline documentation*) como a la arquitectura de los sistemas y su núcleo. Esto presenta el problema de que el involucrar a una persona en el desarrollo del sistema y sus módulos requiere una especial atención de parte del personal del equipo actual. Por otro lado, esto genera muchas veces duplicación de código o evita la reutilización (y consecuente abstracción) que podría ser beneficiosa y hacer más eficiente el trabajo desarrollo.

Se ha detectado también que aunque el equipo de desarrollo actual cuenta con 3 miembros, la ausencia de convenciones de programación y lineamientos de codificación hacen que el desarrollo siga líneas divergentes y agrega complejidad a la revisión del código.

Por otro lado, se ha determinado la posibilidad de involucrar herramientas de *Software configuration management*, ya la integración del trabajo que el equipo actual lleva a cabo se hace de forma completamente manual. Esto consume tiempo y es considerablemente propenso a errores en los cuales alguno de los cambios no es incluído

en la versión final del *software*.

Finalmente, se ha detectado dentro de la empresa una relativa falta visibilidad en cuanto a las tareas que cada miembro del equipo está realizando, así como al control de los requerimientos de cambios o nuevas funcionalidades que son reportadas por terceros o por miembros de la misma empresa. Esto resulta en que algunas tareas o pendientes no son completados no por falta de tiempo o de conocimiento sino porque no fueron registrados y no se les dió seguimiento.

D. Análisis de requerimientos

Dado el análisis anterior, se acordó con la empresa implementar una base de código sobre la cual la empresa pueda continuar con el desarrollo de las soluciones de *software* que ofrece a sus clientes.

Esta base de código tiene como requerimiento fundamental definir los lineamientos para futuros módulos específicos, y debe regular principalmente el área común, que incluye el manejo de: módulos, empresas, usuarios, perfiles de acceso y un directorio de entidades con contactos.

Además, se propuso la implementación de una serie de prácticas y herramientas que ayuden a abordar los problemas de ingeniería detectados.

De acuerdo con la metodología de trabajo **Scrum**, los requerimientos funcionales comunes de *software* de cara a los clientes finales de la empresa fueron descritos como historias de usuario [19]. Por otro lado, dada la orientación del proyecto a la ingeniería de software de cara a la empresa, se describieron como historias de usuario los aspectos de ingeniería de software a ser implementados dentro de la organización.

Esta decisión tiene como fundamento al mismo Sutherland, quien describe en su libro *The power of Scrum**,*** que la metodología puede ser utilizada, no solo para proyectos de software sino para cualquier tipo de proyecto [19]. En este caso, se ha optado por trabajar **Scrum** para la implementación de estos aspectos de ingeniería de software, que se encuentran en el **Apéndice A**. Dado que el equipo de trabajo en este caso consistió

únicamente en un miembro, la estimación de complejidad en puntos de estas historias fue unilateral.

Como requerimientos no funcionales se acordó realizar el desarrollo como una aplicación web, con tecnologías open source. Por otro lado, dada la naturaleza del proyecto, se solicitó también evaluar y estructurar el desarrollo de módulos a futuro, y la arquitectura de la interacción entre ellos de cara a áreas específicas. Además se solicitó tomar en cuenta los aspectos de seguridad de seguridad pertinentes.

V. Implementación y desarrollo

A. Metodologías Scrum y XP

Para el presente proyecto se trabajó un subconjunto de las prácticas sugeridas por las metodologías de desarrollo ágil **Scrum** y **XP**, basados en las observaciones y sugerencias publicadas por Jeff Sutherland [19] y Robert Martin [10].

Esta elección fue basada, en primer lugar, en los lineamientos de las metodologías ágiles se apegan muy bien a las necesidades y objetivos de la empresa (ver **Manifiesto ágil** dentro de **Marco teórico**). Por otro lado, se determinó que los problemas abordados por cada metodología se complementan muy bien para proveer una solución integral a la empresa, tanto desde el aspecto administrativo como desde el técnico.

En concreto se eligió y abordó la implementación de las siguientes aspectos de las metodologías:

1. Elaboración y mantenimiento del *product backlog* con requerimientos funcionales y no funcionales. Estos requerimientos partieron de las **Historias de usuario**, que fueron desglosadas en tareas para conformar el *product backlog*. Una vez desglosadas, se procedió a asignarles una valoración numérica de puntos de complejidad en una escala de 1 a 10, siendo 1 lo menos complejo y 10 lo más complejo.

2. Definición de *sprints* con una duración de dos semanas. La definición de los *sprints* se fundamentó tanto en la prioridad de las **Historias de usuario** como en la coherencia en el orden de su implementación.

3. Presentación de reportes release burndown, trabajando el proyecto completo como un solo una sola entrega (*release*) dividida en múltiples iteraciones (*sprints*). Para los diagramas, el concepto de *esfuerzo restante* que se describe en [15] fue tomando como los puntos de complejidad pendientes de resolver en cada task del *product backlog*.

4. Desarrollo basado en pruebas. El desarrollo se realizó considerando las

indicaciones sugeridas por **TDD**. Si bien no se pretende un 100% de cobertura de pruebas automatizadas (unitarias y funcionales) dentro del código, sí se busca implementarlas de los aspectos de lógica de negocio más importantes, a manera de establecer un lineamiento en cuanto a esta forma de trabajo.

5. Entorno de integración continua. Como parte del trabajo se implementó un entorno de integración continua, en estrecha relación con las pruebas automatizadas y el control de versiones.

B. Diseño

1. Ingeniería de software. Se incluye bajo este apartado el acercamiento y prácticas que se propone implementar dentro de la empresa, específicamente dentro del presente proyecto, de acuerdo con las áreas más relevantes de la ingeniería de software según los problemas de ingeniería detectados (ver la sección **Ingeniería de software** en **Marco teórico** y **Problemas de ingeniería** dentro del **Análisis del negocio**).

Se plantea abordar el problema de la reaparición de problemas resueltos previamente o la introducción de nuevos problemas con actualizaciones con la implementación de desarrollo basado en pruebas (TDD). Si bien la plataforma que será desarrollada no contará inicialmente con clientes reales, se considera conveniente proponer este estilo de desarrollo desde el inicio del proyecto.

Con respecto a la falta de documentación, se propone como primer acercamiento la elaboración de diagramas Entidad Relación para la documentación del trabajo que se realizará. Por otro lado, se plantea la adopción del estándar de codificación **PSR-2**[14], propuesto por el *PHP Framework Interop group*, para reducir la fricción cognitiva en el análisis y lectura de código de distintos autores. De acuerdo con el SWEBOK, esta práctica soporta la construcción de código más fácilmente verificable.

Por otro lado, se plantea la adopción de **Subversion** como herramienta de manejo de configuración de *software*, y se define que se seguirá un desarrollo según el cual cada

[14] https://github.com/php-fig/fig-standards/blob/master/accepted/PSR-2-coding-style-guide.md

nueva funcionalidad será trabajada en una rama (*branch*) independiente para ser integrada dentro de la línea de desarrollo principal (*trunk*) al ser completada. Por otro lado, de acuerdo con SWEBOK, se plantea la implementación de un sistema de integración continua que posibilite la supervisión y entrega constante de *software* funcional.

Con respecto a la visibilidad de estado del trabajo y continuidad de requerimientos, se propone la implementación de un sistema de asuntos (*tickets*) en donde puedan registrarse y consultarse los avances obtenidos sobre distintas tareas.

2. Arquitectura del sistema. Para dar respuesta a los problemas de tecnologías detectados, se propone realizar el desarrollo utilizando un lenguaje de programación de relativa amplia popularidad, así como con soporte estable a mediano plazo. Por otro lado, este lenguaje deberá tener la capacidad de aprovechar las tecnologías nuevas de *hardware* conforme se vayan desarrollando.

Con respecto al almacenamiento de la información, se plantea determinar y utilizar un motor de base de datos que provea integridad referencial, de forma que se pueda incluir una capa más de abstracción sobre el trabajo hecho por los desarrolladores.

De acuerdo con el análisis, se define la implementación del sistema como una aplicación web que siga la arquitectura Modelo-Vista-Controlador. Este sistema será dividido en una serie de módulos independientes haciendo uso de un *framework* de desarrollo. Cada uno de estos módulos tendrá la capacidad de ampliar y la funcionalidad existente (tablas en base de datos,vistas y elementos de despliegue en interfaz con usuario), así como especificar de qué otros módulos y componentes depende mediante una herramienta de manejo de dependencias.

La aplicación podrá ser instalada sobre las plataformas más populares y ser ofrecida como servicio en la nube, de acuerdo a los lineamientos de SaaS.

La base de código desarrollada implementará la funcionalidad de autenticación y autorización sobre los componentes de la aplicación, tanto de los desarrollados como opcionalmente de futuros módulos.

C. Herramientas

A continuación se describen las herramientas elegidas para la implementación de la base de código solicitada. Uno de los requerimientos que cumplen es ser expresamente multiplataforma (abarcando los sistemas operativos de uso más común), de manera que cualquier inclusión de un nuevo miembro al equipo de trabajo no encuentre una barrera con respecto a la tecnología utilizada. Por otro lado, se tuvo como criterio de elección el uso de estas herramientas con amplia difusión en entornos de producción.

- Control de versiones: **Subversion**[15]

Para el control de versiones del desarrollo a realizar se eligió la herramienta **Subversion** dado su nivel de estabilidad y documentación. Por otro lado, se consideró que, dado que el desarrollo se hace por completo físicamente dentro de la empresa, un sistema de control de versiones centralizado ofrece la suficiente funcionalidad comparado con la complejidad y mantenimiento de un sistema de control de versiones distribuido como **Mercurial** o **GIT**.

- Control de tickets y navegación de repositorio: **Trac**[16]

Para efectuar el control de posibles nuevos requerimientos y problemas se eligió esta herramienta ya que permite integración con **Subversion** para visualización de código y que además ofrece control de *milestones* y *wiki* (útil para documentación).

- Lenguaje principal de desarrollo: **PHP**

La elección de este lenguaje de desarrollo obedece a la naturaleza del proyecto como aplicación web, además de la popularidad y universalidad del lenguaje. Por otro lado, el lenguaje implementa patrones de diversos estilos de programación (procedural, orientada a objetos, etc.), por lo que ofrece algunas ventajas al momento de querer migrar a un equipo de desarrollo de un lenguaje de programación diferente.

[15] http://subversion.tigris.org
[16] http://trac.edgewal.org

Para el manejo de librerías y dependencias PHP del *software*, se elige el estándar *de facto* más reciente producido por la comunidad: **Composer**[17].

- Framework y arquitectura: **Symfony2** y **MVC**

La elección del entorno de trabajo (*framework*) **Symfony2** se debe a la experiencia que el estudiante tiene de la herramienta, así como su versatilidad. Por otro lado, el patrón MVC que este *framework* implementa se ajusta bien a la naturaleza del proyecto como aplicación orientada a datos.

- Base de datos y ORM: **MySQL** y **Propel**

Con respecto sistema de manejo de base de datos, la elección de **MySQL** se fundamenta en la experiencia que la empresa tiene en su uso, así como la amplio uso que tiene dentro de sus socios**.** Por otro lado, dados que maneja distintos motores de bases de datos, es posible trabajar tanto con tablas que ofrecen ventajas de integridad referencial (**InnoDB**) como con tablas similares a las que actualmente se trabajan (**MyISAM**).

Acerca del **ORM** a utilizar dentro de la aplicación, se elige trabajar con **Propel** dado que es la librería con mayor soporte dentro de la comunidad **Symfony2** que implementa el patrón *ActiveRecord*, un patrón más fácilmente asociable al trabajo directo con bases de datos relacionales que el utilizado por otras librerías, *Unit of work*, lo que implica menor curva de aprendizaje que librerías como **Doctrine2**.

- Servidor de aplicaciones: **Apache2** o **IIS**

Con respecto a la tecnología a utilizar como servidor de aplicaciones, se determinó que **Apache2**, a pesar de ser multiplataforma, no suele utilizarse en

[17] https://getcomposer.org

entornos de producción que utilicen herramientas sistemas operativos Windows[18]. Por esta razón, se llevaron a cabo las pruebas necesarias para mostrar que el resto del *stack* se acopla sin problemas a la plataforma *Internet Information Services* (**IIS**) de Microsoft.

- Pruebas automatizadas: **PHPUnit[19]**

Dentro de la comunidad de desarrollo de **PHP**, **PHPUnit** puede considerarse como el estándar *de facto* en cuanto a pruebas unitarias y funcionales. Aunque no es la herramienta para pruebas que mejor se acopla al estilo de programación TDD[20], cuenta con soporte integrado con **Symfony2** y tiene la inicial ventaja de ser similar a otros entornos de pruebas de amplia popularidad como **JUnit** o **xUnit**.

- Entorno de integración continua: **Jenkins[21]**

Para trabajar el aspecto de integración y entrega continua se eligió el paquete **Jenkins** debido a su amplio uso, su naturaleza de código abierto, y a la posibilidad de descargar la aplicación e instalarla en un servidor local. Esto es fundamental para el funcionamiento de la empresa ya que su código es privativo y no se desea en este caso que herramientas SaaS como **Travis-CI[22]** tengan acceso al mismo. Por otro lado, se determinó que, aunque está enfocado y escrito en Java, se determinó que existen proyectos que permiten su correcta integración con lenguajes como PHP.

- Entorno de desarrollo integrado (IDE): **Sublime Text**

Acerca de la herramienta para edición general de código, se decide utilizar un editor de texto en lugar de un IDE completo principalmente por dos razones: la primera es que se acopla mejor al hardware disponible por su menor consumo de

[18] La version primaria para trabajar Apache 2.0 es Windows NT, de acuerdo con: http://httpd.apache.org/docs/2.0/platform/windows.html
[19] https://phpunit.de/
[20] Ver por ejemplo Behat, http://docs.behat.org/en/latest/
[21] http://jenkins-ci.org
[22] http://travis-ci.org

recursos; la segunda es que forza a una mejor comprensión del *framework* y las librerías a utilizar, ya que al contar con la principal característica de los IDEs completos, *autocompletación*, el desarrollador se ve forzado a consultar la documentación de la librería y, dado que se está trabajando con librerías *open source*, esto no es ningún problema.

Esto tiene la ventaja también de contar con una fuente de ejemplos y de documentación interna de la cual se puede obtener mucho conocimiento y experiencia acerca de qué patrones seguir durante el desarrollo.

Finalmente, al estudiar las posibilidades del editor de texto elegido, **Sublime Text**, se determina que cuenta con suficientes *plugins* para hacer manejable el desarrollo de un proyecto de tamaño considerable: control de versiones, entorno de *debugging*, herramienta diferencial, navegación rápida entre la estructura de carpetas y archivos de un proyecto, persistencia de configuración de ventanas.

- Ofuscamiento de código: **Ioncube**[23]

Para llevar a cabo la tarea de ofuscamiento de código se eligió **Ioncube** por su soporte integrado con línea de comando, lo que permite integrar este proceso dentro del resto de tareas dentro de un entorno de integración basado en **Jenkins**.

D. Desarrollo

El objetivo del trabajo entonces se resume en la implementación de una base de código y la inclusión de prácticas de ingeniería de software para actualizar no solo la tecnología sino la forma de trabajo dentro de la empresa.

De acuerdo con **Scrum**, el trabajo a realizar se agrupó en *sprints* con una duración de dos semanas, previo a las cuales se definían las historias que deben ser ser trabajadas, así como los indicadores de logro y las tareas.

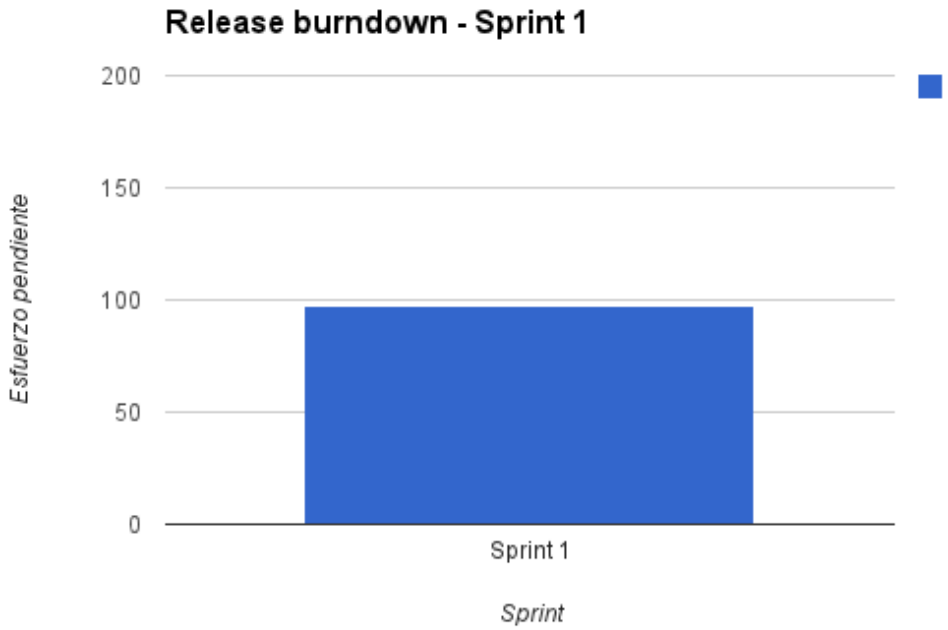
[23] http://www.ioncube.com/

De acuerdo con la orientación del proyecto, se resolvió abarcar en cada *sprint* tanto aspectos del *core* a desarrollar como de ingeniería de software dentro de la empresa. A continuación se describen las tareas realizadas en cada uno de los *sprints*.

1. **Sprint 1 – Inicialización**. El objetivo de esta iteración es definir instalar y configurar las herramientas indispensables para iniciar con el desarrollo de forma ordenada. Por otro lado, se plantea como objetivo crear el esqueleto de la aplicación para demostrar la utilización y la utilidad de dichas herramientas.

Al inicio de esta iteración se presenta el *release burndown* mostrado en la **Figura No.4**.

Figura No.4. *Release burndown* de iteración 1



Además, se decide abarcar en esta fase del proyecto los requerimientos descritos por las **Historias de usuario 8** y **10**. Esto debido a que dichas historias conciernen al desarrollo mismo del proyecto, es conveniente abarcarlas desde su inicio.

La iteración, por tanto, cubre las siguientes tareas:

- Instalar y configurar ambiente de trabajo dentro de la empresa
- Crear una aplicación utilizando **Symfony2** y registrarla en el repositorio de **Subversion**
- Crear y configurar la comunicación hacia la base de datos MySQL
- Configurar el entorno de *debugging*

a. Montar ambiente de trabajo. Previo a iniciar con el desarrollo de la aplicación es necesario configurar el ambiente de trabajo. El sistema operativo bajo el que se trabaja es Linux, en su distribución Ubuntu. Sin embargo, tal como se describió anteriormente, la elección del *stack* de tecnología tiene en consideración que todas las herramientas fueran multiplataforma.

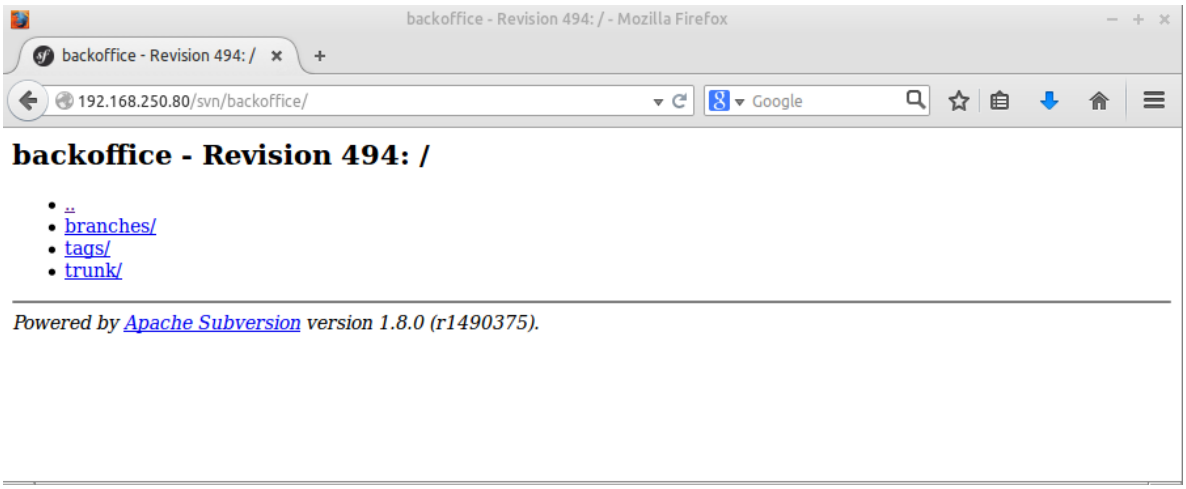
Por otro lado, dado que se determinó previamente que la empresa no contaba con un sistema de control de versiones, se juzga conveniente proceder a instalar y configurar un repositorio de **Subversion** dentro de uno de los servidores Linux internos de la empresa. Este servidor fue puesto a disposición del estudiante para instalar las herramientas que juzgara apropiadas.

Esta instalación es configurada para soportar un acceso local a través de la intranet, que a su vez se encuentra detrás del *firewall* de la empresa. Como método de acceso se decide instalar el módulo `web_dav_svn`[24] junto con el servidor de aplicaciones **Apache**, ya que permite crear y configurar múltiples repositorios, así como utilizar el método `HTTP Basic` de autenticación, que se considera un buen primer acercamiento.

El resultado de esta instalación es un servidor de control de versiones que puede ser fácilmente navegado a través de un *browser* dentro de la intranet, tal y como se muestra en la **Figura No.5**.

[24] http://svnbook.red-bean.com/en/1.8/svn.serverconfig.httpd.html#svn.serverconfig.httpd.ref.mod_dav_svn

Figura No.5. Navegación de control de versiones



Para dar respuesta al problema detectado de la falta de continuidad a requerimientos se considera también conveniente trabajar, como parte de esta iteración, la instalación y configuración de un sistema de control de temas (*issue tracking*). La instalación de este sistema pretende apoyar los aspectos de *Requerimientos de Software* y *Manejo de configuración de software*, de acuerdo con las prácticas descritas en el SWEBOK.

Para esto, se instala y configura la aplicación *open source* **Trac** dentro del mismo servidor interno. Esta aplicación ofrece funcionalidades tales como la integración con **Subversion** para la visualización de código y cambios, registro de *tickets* , control de *milestones* y *wiki*.

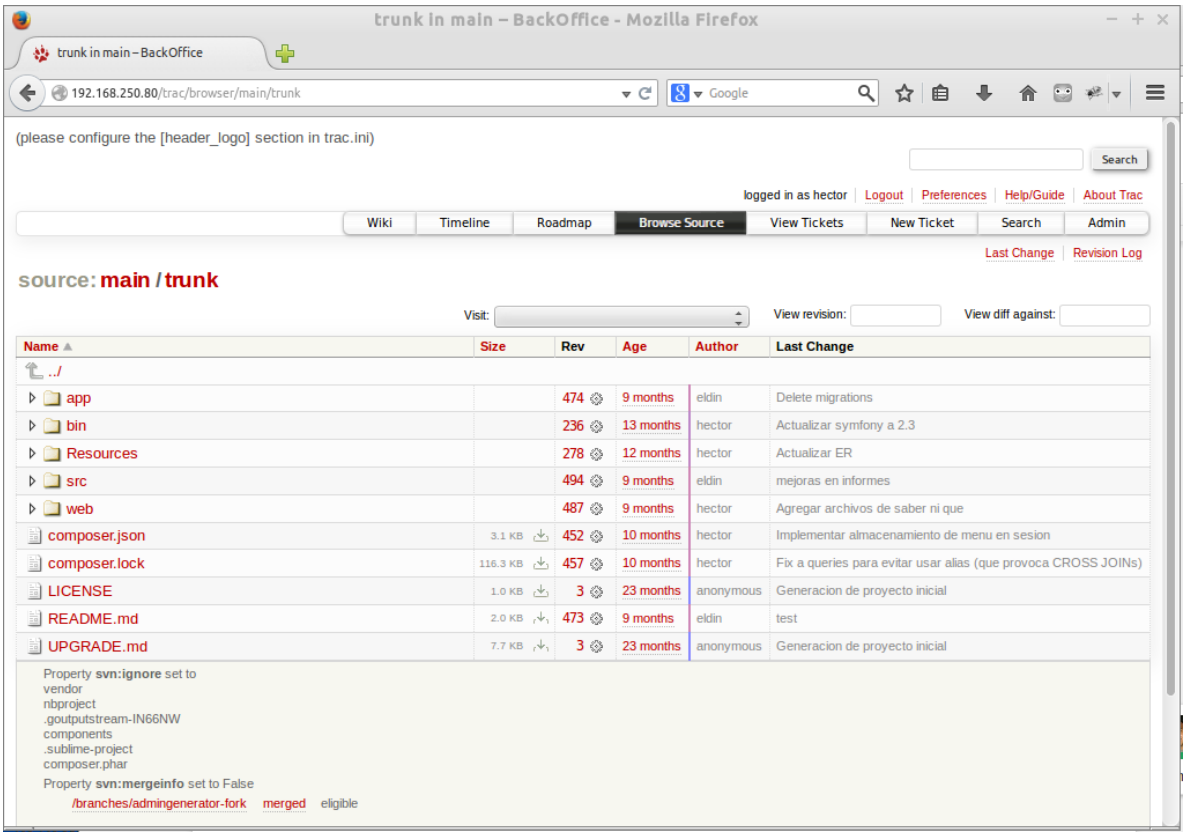
Con el objetivo de optimizar recursos, se elige configurar la aplicación para ser ejecutada a través de **Apache**, en contraposición con utilizar su propio servidor, a manera de centralizar las distintas herramientas de trabajo. Para esto, resulta necesario configurar **Apache** con el módulo `mod_wsgi`[25], que permite ejecutar aplicaciones *Python* compatibles con **WSGI**[26].

De esta forma, el sistema de *tickets* instalado puede ser accedido tal y como se

[25] https://code.google.com/p/modwsgi
[26] http://wsgi.readthedocs.org/en/latest

muestra en la **Figura No.6**.

Figura No.6. Control de tickets configurado



b. Crear aplicación. Una vez montado el entorno de trabajo, se procede a crear la aplicación de **Symfony2**. Los proyectos en **Symfony2** pueden inicializarse fácilmente a través del manejador de paquetes **Composer**, que se encarga también de manejar las dependencias a otras librerías de PHP que serán utilizadas durante el desarrollo.

El principal aspecto a considerar durante esta fase es la correcta aplicación de las funcionalidades de **Subversion** para lograr que el repositorio almacene exclusivamente las clases y archivos relevantes para el desarrollo. Con este objetivo se implementa el uso de la propiedad `svn:ignore`, que se configura para ignorar el contenido de las carpetas de `cache` y `logs`, así como la carpeta de carga de archivos, cuyo contenido no se desea mantener en el repositorio.

En el sitio oficial[27] puede encontrase una justificación más amplia de la correcta aplicación de las propiedades de **Subversion** para un proyecto **Symfony2**.

c. Crear y configurar comunicación hacia base de datos. Una vez completada esta configuración inicial, es necesario definir el archivo de configuración de conexión a la base de datos.

Es en este punto donde empieza a tomar realce el uso correcto del manejador de paquetes **Composer**, ya que la instalación de la librería **Propel** y su respectivo *bundle* (módulo de **Symfony2**) consiste únicamente en la inclusión de una línea de configuración al archivo `composer.json`. Una vez hecho esto, basta con ejecutar

```
php composer.phar update
```

Para que descargar el *bundle* de **Propel** para **Symfony2**, así como todas sus dependencias.

Luego de la actualización, se procede a configurar el archivo app/config/parameters.yml en donde se coloca la información de la conexión a la base de datos, tras lo cual puede utilizarse la misma consola de **Symfony2** y **Propel** para crear la base de datos y actualizar continuamente su estructura a través de archivos de migración[28].

d. Configurar entorno de debugging. Finalmente, es conveniente configurar el entorno de trabajo de tal forma que cuente con la herramienta **XDebug**. Esta es una herramienta de *debugging* que resulta de extrema utilidad cuando se desea efectuar una sesión compleja de revisión de problemas.

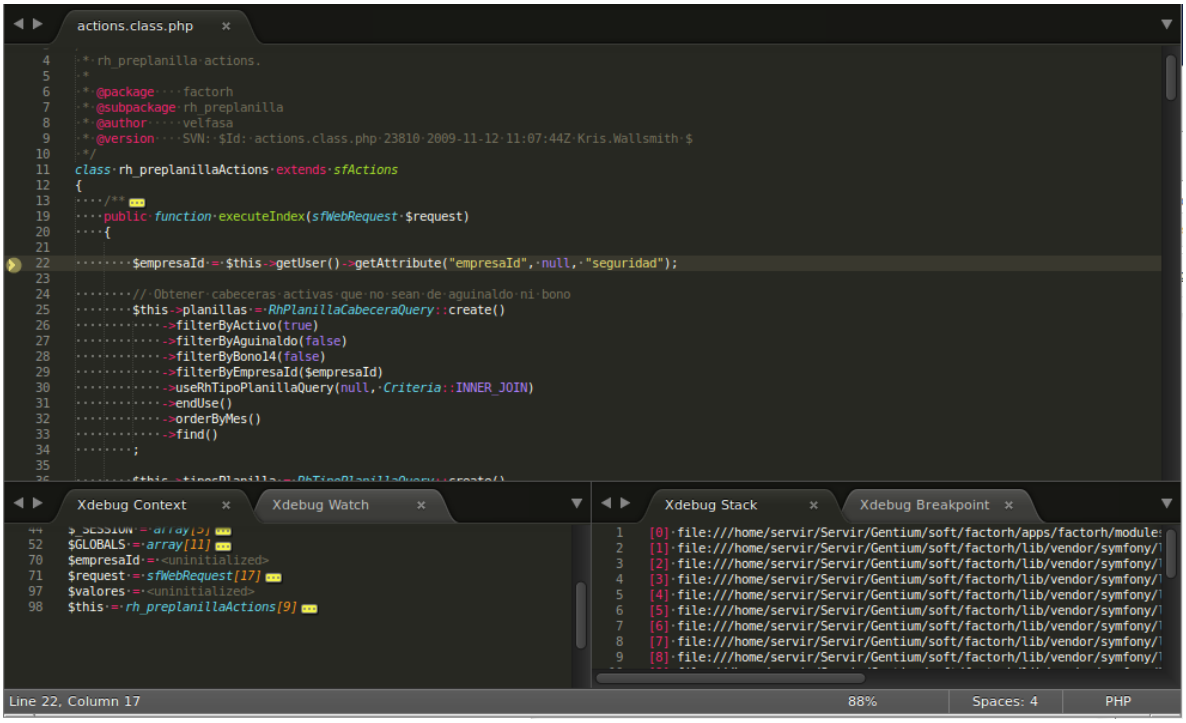
Este tipo de herramientas suele incluirse por defecto en IDEs como **Netbeans** o **IntelliJ IDEA**, sin embargo para **Sublime Text** es necesario hacer la instalación manual de **XDebug** y del plugin **Xdebug client**[29].

---

[27] http://symfony.com/doc/current/cookbook/workflow/new_project_svn.html
[28] http://propelorm.org/Propel/documentation/10-migrations.html
[29] https://github.com/martomo/SublimeTextXdebug

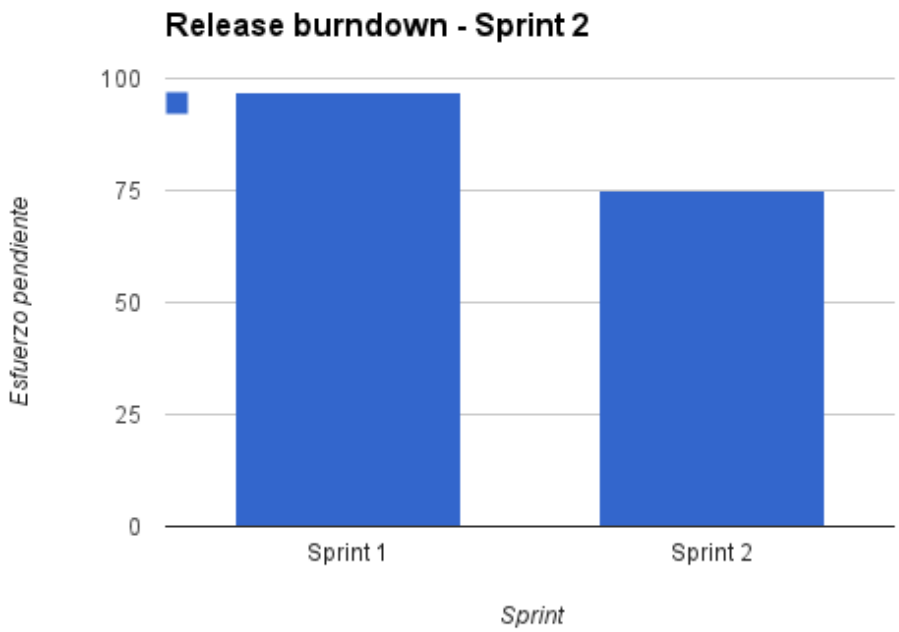
Una vez instalado y configurado este plugin, es posible seguir el flujo de ejecución del código del proyecto por medio de las herramientas habituales de *breakpoints* y *watches* tal y como se muestra en la **Figura No.7**.

Figura No.7. Ejecución de XDebug en entorno Sublime Text



2. Sprint 2 – Empresas, entidades y pruebas automáticas. Al inicio de esta iteración se presenta el *release burndown* mostrado en la **Figura No.8**.

Figura No.8. *Release burndown* de iteración 2



El objetivo de esta iteración es implementar las primeras funcionalidades de la base de código, con el objetivo de abarcar las **Historias de usuario 1**, **2**, **5** y **8**. Esto implica las siguientes tareas:

- Definir la estructura funcional de los módulos
- Llevar a cabo la modelación de la base de datos
- Implementar las operaciones CRUD para empresas, usuarios, módulos y perfiles

Por otro lado, dado que durante esta iteración se implementa el primer módulo funcional del sistema, es un buen momento para iniciar con el desarrollo basado en pruebas que pueda luego ser automatizado.

a. Estructura y funcionalidad modular. Dada la naturaleza modular que se pretende alcanzar con el presente proyecto, y considerando el *stack* de tecnología a utilizar, se propone explotar y expandir el concepto de *bundles* utilizado por **Symfony2**. Esto permite desarrollar la funcionalidad solicitada dentro de un *bundle* principal, y trabajar los demás componentes con su respectiva funcionalidad dentro de paquetes autocontenidos que pueden desarrollarse independientemente.

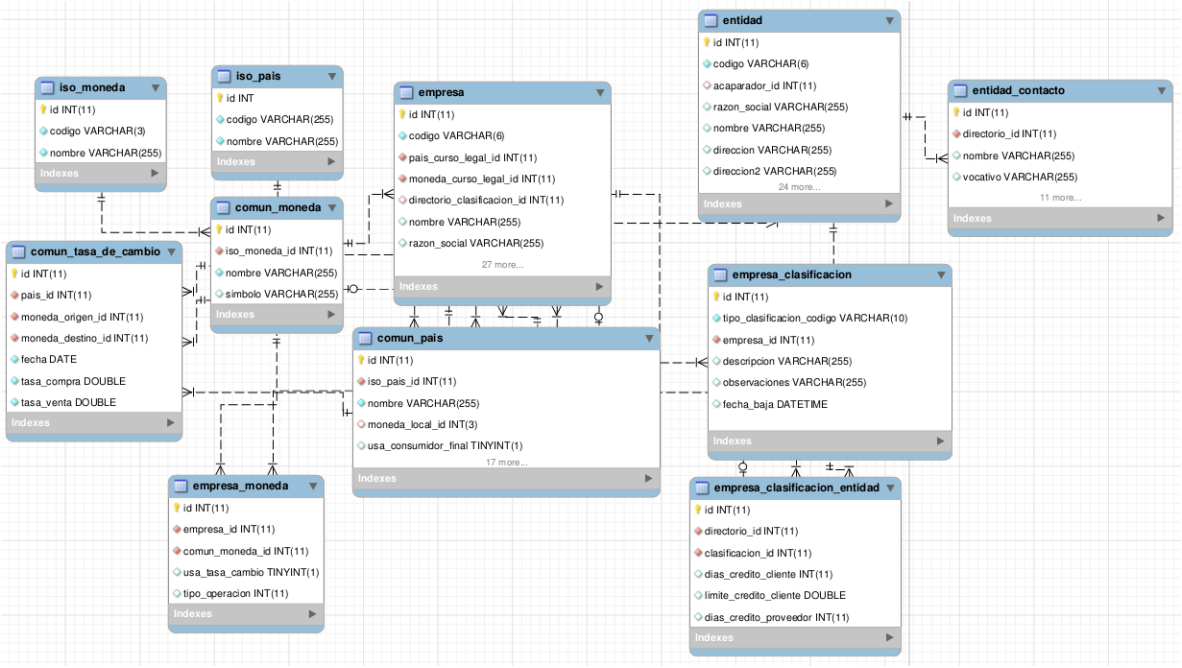
En la documentación oficial[30] de **Symfony2** se encuentra una ámplia explicacóin sobre el proposito, la extensibilidad y la funcionalidad que puede ir contenida en un *bundle*.

b. Modelado de base de datos. De acuerdo con las **Historias de usuario 2** y **5**, se procede a desarrollar durante esta iteración el modelo de base de datos para los objetos de dominio *empresas* y *entidades*. Se crea además la asociación hacia tablas auxiliares que pueden llegar a ser utilizadas por diversos módulos: países, monedas.

La fase de la base de datos desarrollada puede visualizarse en el *Diagrama Entidad Relación* (ER), creado utilizando la *suite* **MySQL Workbench[31]**, que se muestra en la **Figura No.9**.

[30] http://symfony.com/doc/current/cookbook/bundles/best_practices.html
[31] http://www.mysql.com/products/workbench

Figura No.9. Diagrama entidad-relación de iteración 2



La consideración relacionada con las tablas `iso_moneda` e `iso_pais` consiste en que el módulo bajo desarrollo de la aplicación brinde información general de monedas y países según los esquemas de ISO 3166-1 e ISO 4217. De acuerdo con esto, el sistema permitirá definir qué monedas serán utilizadas, tomando como base las monedas y países establecidos por los estándares.

Cabe destacar que se propone, a manera de convención, utilizar las tablas con los nombres completos y en singular de los objetos de dominio y/o relaciones que almacenan. Esto con el objetivo de facilitar el proceso de capacitación de futuros colaboradores dentro de la aplicación, así como de proveer una primera documentación *inline* sobre lo que realiza cada componente del sistema.

c. Operaciones CRUD para empresas, entidades y asociados. El concepto **CRUD** (*Create, read, update, delete*, por sus siglas en inglés) se refiere a las funciones y operaciones básicas de un *software* sobre su base de datos. Durante esta iteración se procede a evaluar, elegir e instalar el *bundle* `AdmingeneratorGeneratorBundle`[32] que provee herramientas para la generación de módulos básicos de CRUD.

Cabe destacar que aún luego de la evaluación, se detectó que el *bundle* elegido aún carece de algunas funcionalidades de extensibilidad que iban a resultar requeridas para el desarrollo, tales como la posibilidad de definir *fallbacks* para las rutas, la posibilidad de disparar eventos asociados a los distintos objetos, la posibilidad de detectar cambios en *bundles* adicionales al principal, para permitir extender formas y acciones y la capacidad de sobreescribir con mayor nivel de detalle las vistas en formato `twig`.

Lo anterior lleva a crear un *fork* del bundle, aprovechando su disponibilidad como código *open source*. La versión utilizada del *bundle*[33] incluye muchas de las posibilidades requeridas.

Para proceder con la implementación, se crea y habilita el *bundle* `AdministracionGeneralBundle`, utilizando la interfaz de línea de comando (CLI) de **Symfony2**. Dentro del mismo, se define la estructura de la base de datos utilizando el formato XML definido por **Propel**[34].

Propel permite entonces generar y ejecutar las declaraciones DDL (*Data definition language*, por sus siglas en inglés) para la sincronización de la base de datos con la estructura definida utilizando los siguientes dos comandos:

```
php app/console propel:migration:generate-diff

php app/console propel:migration:migrate
```

[32] http://symfony2admingenerator.org
[33] Puede consultarse en https://github.com/hectorh30/AdmingeneratorGeneratorBundle
[34] Puede consultarse en http://propelorm.org/Propel/references/schema.html

d. Desarrollo basado en pruebas (TDD). Durante esta iteración se realiza la revisión de las alternativas existentes en el entorno de PHP y **Symfony2** para trabajar siguiendo los lineamientos del desarrollo basado en pruebas. Este desarrollo está orientado tanto hacia pruebas unitarias (ejecutadas contra clases específicas) como pruebas funcionales (ejecutadas contra la aplicación completa).

Symfony2 trabaja por defecto con la herramienta **PHPUnit** para desarrollo de pruebas automatizadas. En el caso de las pruebas unitarias, basta con extender la clase nativa de **PHPUnit** `PHPUnit_Framework_TestCase`.

Por el otro lado, las pruebas funcionales utilizan un flujo de trabajo que involucra la simulación de una petición, el levantamiento de la aplicación (*bootstraping*), el procesamiento de posibles redirecciones y la evaluación de la respuesta; todo esto incluso múltiples veces. Es por esto que **Symfony2** provee la clase base `FrameworkBundle\Test\WebTestCase`, que provee la funcionalidad inicial para llevar a cabo estos pasos.

Dada la naturaleza orientada a datos de la aplicación bajo desarrollo, se considera conveniente implementar una clase `WebTestCase` que ejecute, para cada prueba a realizar, el levantamiento de una base de datos con un conjunto de datos predefinido, a forma de verificar el entorno en el que se realizan las pruebas.

En la **Figura No.10** puede visualizarse un ejemplo de una prueba funcional asociada al CRUD recién creado del objeto de dominio *Empresa*.

Figura No.10. Prueba funcional asociada a CRUD de *Empresa*

```php
<?php

namespace Gentium\AdministracionGeneralBundle\Tests\Functional\Controller\ComunNomenclaturaBase;

use Symfony\Component\HttpFoundation\RedirectResponse;

use Gentium\AdministracionGeneralBundle\Tests\Functional\WebTestCase;
use Gentium\AdministracionGeneralBundle\Model\ComunNomenclaturaBase;

class EmpresaTest extends WebTestCase
{
    public function testIndex()
    {
        $client = static::createClient();
        $client->insulate();

        $this->login($client);

        $client->request('GET', '/adm/empresa/');
        $response = $client->getResponse();

        $this->assertTrue($client->getResponse()->isSuccessful());
    }

    public function testNew()
    {
        $client = static::createClient();
        $client->insulate();

        $this->login($client);

        $client->request('GET', '/adm/empresa/new');
        $response = $client->getResponse();

        $this->assertTrue($client->getResponse()->isSuccessful());
    }
}

```

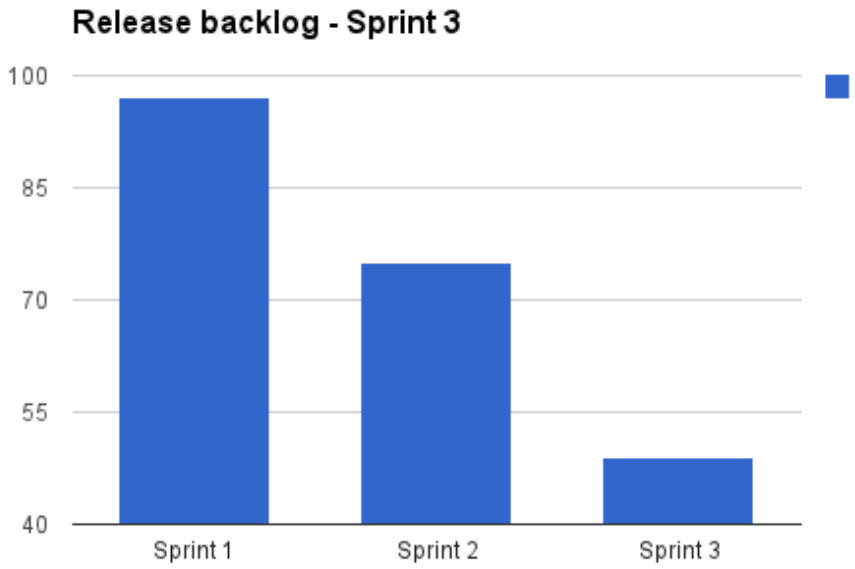
Para ejecutar estas pruebas en el entorno local, basta con ejecutar el comando:

```
phpunit -c app/
```

Previendo el eventual escenario en que múltiples desarrolladores empiecen a realizar aportaciones de código a este y otros módulos, se establece la dependencia `phpunit/phpunit` dentro del archivo `composer.json` bajo la categoría `require-dev` de forma que, en modo desarrollo, **PHPUnit** sea instalado automáticamente y así se puedan ejecutar pruebas como parte del flujo habitual de trabajo.

3. **Sprint 3 – Perfiles de usuario y accesos**. Al inicio de esta iteración se presenta el *release burndown* mostrado en la **Figura No.11**.

Figura No.11. *Release burndown* de iteración 3



Con el objetivo de abarcar durante esta iteración las **Historias de usuario 3** y **4**, se plantea continuar con la implementación de la base de código solicitada. El procedimiento a realizar puede desglosarse en las siguientes tareas:

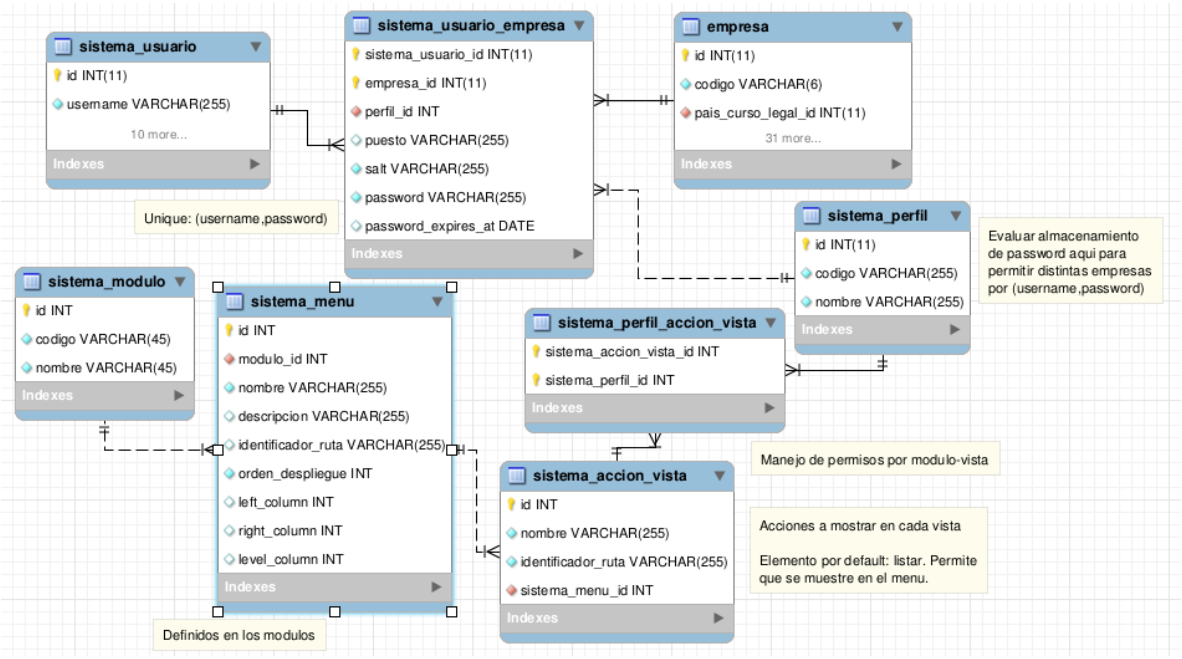
- Modelar la base de datos para perfiles, usuarios, módulos y accesos
- Implementar el CRUD para los objetos de dominio principales y algunas características generales
- Implementar las reglas de validación y construcción de permisos según los perfiles y empresas asignadas al usuario

a. Modelado de base de datos para perfiles, usuarios, módulos y accesos. Las **Historias de usuario 3** y **4** detallan el requerimiento de implementar la autenticación y autorización de los usuarios al sistema basados en la empresa elegida por el usuario, el perfil y las vistas configuradas para el mismo. Partiendo de esta descripción se propone el diseño e implementación de los siguientes objetos de dominio: *usuario*, *módulo*, *perfil*. Por otro lado, se prevé la necesidad de almacenar información sobre *menús* y las *acciones* que cada elemento de menú permitirá.

Cabe destacar que los objetos de dominio *módulo* y *menús* están diseñados para ser incluidos en los módulos, y, al menos inicialmente, no creados dentro de la aplicación.

Esto conduce a diseñar la siguiente parte de la base de datos, y a diagramarla de la misma forma que en el *sprint 2*, como puede verse en la **Figura No.12**.

Figura No.12. Diagrama entidad-relación de iteración 3



Del requerimiento establecido por las historias de usuario abordadas en este *sprint* se deduce también la necesidad de hacer asignar el almacenamiento de las contraseñas de los usuarios a la relación entre un usuario y una empresa, de forma que resulte posible eventualmente que un usuario tenga acceso a empresas distintas utilizando contraseñas distintas.

Cabe destacar que, en consideración al futuro desarrollo que girará alrededor de esta base de datos, se propone implementar una convención tal que, con excepción de las tablas de uso general (tal como la tabla `empresa`), las tablas sean creadas con un prefijo que identifique a qué módulo corresponden, ya que esto contribuirá a facilitar más adelante el desarrollo. En este caso, el único prefijo utilizado es `sistema_`, ya que la mayoría de tablas trabajadas pertenecen al núcleo del sistema.

b. Implementación de CRUD y características generales. De los objetos de dominio implementados durante esta iteración, basta de momento implementar CRUD de forma similar a lo que se realizó previamente.

Cabe destacar en este punto que fue necesario hacer uso de la funcionalidad de inyección de dependencias provista por **Symfony2** para hacer un cambio fundamental en la manera en que las formas de `HTML` eran procesadas y asociadas a los objetos del **ORM**. Esto ya que, por defecto, el componente `PropertyPath` de **Symfony2** no consideraba los *magic methods* de PHP para asignar valores, funcionalidad que es usada extensamente por **Propel2**. Gracias a la forma en la se trabaja la parametrización dentro de **Symfony2**, bastó con especificar que las instancias del objeto `PropertyPath` utilizarían *magic methods* desde el archivo `config.yml` de la aplicación con el siguiente código `YAML`:

```
services:
   property_accessor:
       class: %property_accessor.class%
       arguments: [true]
```

En este punto se detecta también la necesidad de trabajar algunas características generales que se desea considerar en todas las tabla, tales como la posibilidad de almacenar información sobre el usuario y la fecha de creación/modificación o de almacenar independientemente la manipulación hecha sobre algunas tablas, por motivos de trazabilidad.

Para cumplir con este requerimiento se aprovecha la funcionalidad de *behaviours*

provista por **Propel**, que permite definir comportamientos que se aplicarán a todas las tablas marcadas para ese efecto con la etiqueta `<behavior />` dentro de su archivo `XML` de definición.

La funcionalidad de almacenar la fecha y hora de creación/modificación de un registro está incluída dentro del componente de Propel bajo el *behavior* `timestampable`. Sin embargo, la funcionalidad de almacenar el usuario se implementó bajo el nombre de `usuario_asociado`.

Este *behavior* fue diseñado para que todas las clases generadas por propel que lo utilizaran extendieran la interfaz `UsuarioAsociadoInterface`, de forma que fuera posible utilizar dicha información a través de la estructura de implementación de interfaces provista por **PHP** desde la versión 5.

c. Implementación de reglas de validación y permisos según empresas y perfiles. La aplicación para la cual se desarrolla el módulo central tiene como requerimiento controlar el acceso a las distintas funcionalidades en base al usuario que realiza la petición. Esto quiere decir que se la aplicación debe poder identificar quién está realizando la petición (autenticación) y qué permisos tiene dicho usuario para acceder a los recursos solicitados (autorización).

Durante esta iteración se agregó la funcionalidad de poder dar respuesta a estos dos elementos de la seguridad.

A pesar de la funcionalidad base provista por el *framework* para este trabajo, el requerimiento no convencional de manejar la validación de acuerdo a la empresa en la que se encuentra el usuario del sistema requiere la implementación de un validador especial, que extienda la funcionalidad provista por **Symfony2** para el caso habitual de autenticación a través de formas.

Esta validación especial se implementa dentro del módulo principal bajo desarrollo, extendiendo las clases del `SecurityBundle` inlcuido dentro de **Symfony2**. Iniclamente se extendió la clase `FormLoginFactory` para especificar que el proveedor de autenticación sería uno definido dentro de nuestro módulo, en lugar del proveedor por

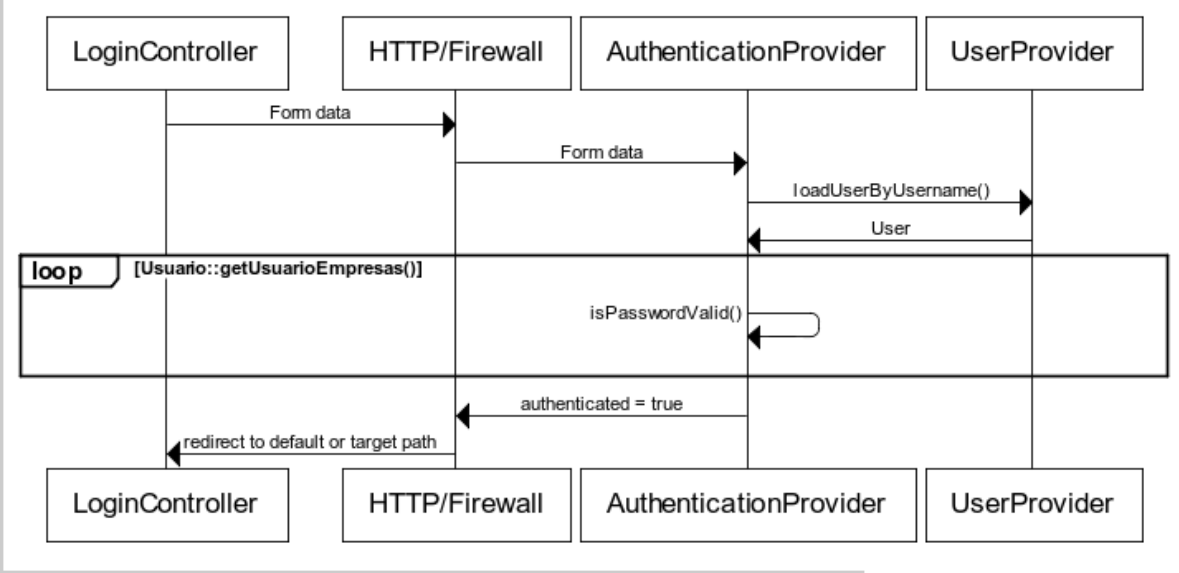
defecto. Luego se extendió el proveedor de autenticación `DaoAuthenticationProvider` para que el método `checkAuthentication` obtuviera el listado de empresas asociadas a un usuario, y validara la contraseña contra la empresa elegida.

Posteriormente fue necesario implementar un UserProvider que realizara la consulta a la base de datos y retornara los objetos `Usuario` de nuestro módulo.

Este nivel de personalización en la seguridad se debe a que la autenticación basada en formas de **Symfony2** está profundamente amarrada a la noción de utilizar un usuario y una contraseña, mientras que en nuestro caso debíamos también considerar la empresa para la autenticación y autorización.

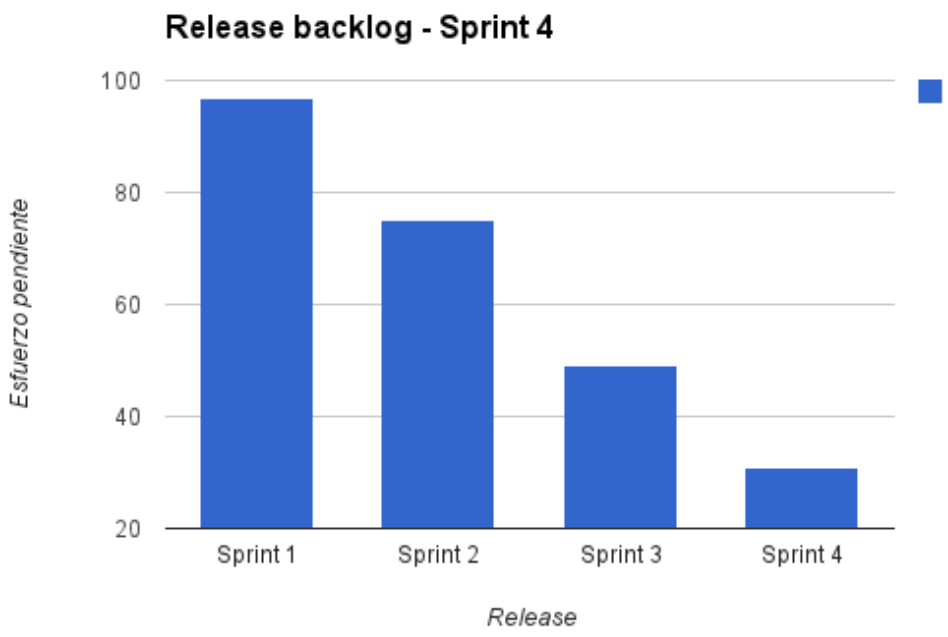
De esta forma, el proceso de autenticación de usuarios, incluyendo los componentes provistos por **Symfony2**, puede describirse a través del diagrama de secuencia mostrado en la **Figura No.13**.

Figura No.13. Diagrama de secuencia de autenticación



4. **Sprint 4 – Menús, integración y entrega continua**. Al inicio de esta iteración se presenta el *release burndown* mostrado en la **Figura No.14**.

Figura No.14. *Release burndown* de iteración 4



El objetivo de esta iteración consiste en implementar la construcción de la interfaz de usuario en base a permisos y perfiles del mismo, de acuerdo a lo descrito en la **Historia de usuario 6**. Además, según lo descrito en las **Historias de usuario 7** y **9**, se pretende automatizar la actualización de una instalación que se encuentre siempre al día, de forma que la administración de la empresa pueda evaluar el desempeño constantemente. Por lo tanto, se proponen las siguientes tareas a desarrollar:

- Implementación de construcción de menú con base a permisos y perfiles del usuario
- Instalación y configuración de herramientas para automatizar entrega continua

a. Implementación de construcción de menú. Esta iteración aportó información de los objetos de dominio *menu* y *acceso* para completar el esquema de base de datos descrito en la iteración anterior.

Cabe destacar que, para llevar a cabo la implementación de menú, se eligió trabajar el manejo de la jerarquía utilizando el modelo **nested set**, en contraposición con el habitual puntero recursivo. Esto se debe a la mayor eficiencia que el modelo **nested set**

presenta durante los *queries* (consultas) de lectura, que serán mayores en este caso que los de escritura. Esta elección se debe también a la carencia de *queries* recursivos dentro de **MySQL**, ya que, aunque otros motores de base de datos si los soportan, **MySQL** fue elegido como el motor de trabajo.

Para trabajar el modelo **nested set**, se determina que **Propel** cuenta con el *behavior* `nested_set` que agrega las columnas `left_column`, `right_column`, `level_column` y `scope` a las tablas sobre las que se especifica. Por lo tanto, la tabla sistema_menu recibe este *behavior*. Se

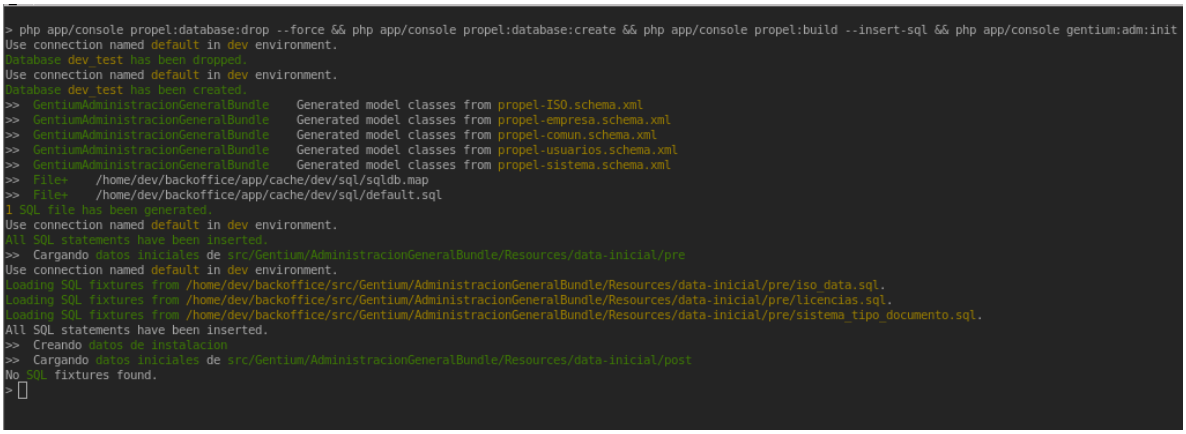
Como puede observarse en la **Figura N.12**, se plantea construir una jerarquía de elementos de menú sin límite de niveles, en donde cada rama está asociada a un módulo, de forma que los futuros módulos del sistema puedan dinámicamente crear estos registros al ser instalados. Además, cada nodo de este árbol tendrá una o más acciones, y son los perfiles quienes determinarán la configuración final del árbol al que un usuario tendrá acceso.

Introducimos aquí la noción de elementos de menú asociados a cada módulo. Este concepto nos brinda la posibilidad de permitir que los módulos a ser desarrollados para el sistema tengan la capacidad de introducir sus propios elementos de menú y acciones, lo que amplía las características de extensibilidad del desarrollo.

Este elemento de extensibilidad se ve ampliado durante esta iteración gracias a la implementación de comandos dentro del proyecto. Se plantea que cada módulo del sistema incluya opcionalmente una clase que extienda a la funcionalidad de comandos de **Symfony2**, en donde se introduzcan al sistema el módulo, los elementos de menú y los permisos requeridos para operar dicho módulo.

De esta forma, al momento de realizar una instalación, además de la instalación del proyecto deberán ejecutarse los comandos de cada módulo a instalar. En la **Figura No.15** observamos un ejemplo de este proceso ejecutado para el módulo central bajo desarrollo.

Figura No.15. Proceso de ejecución de instalación



Para llevar a cabo la tarea de la implementación de menús, se decide utilizar el *bundle* de **Symfony2** `KNPMenuBundle`[35]. Este bundle provee las interfaces para consturir un árbol de menús propio, así como las herramientas para incrustarlo fácilmente en las plantillas `TWIG` del proyecto, definiendo constructores de menú dentro del inyector de dependencias, e incristándolas como se muestra en la **Figura No.16**.

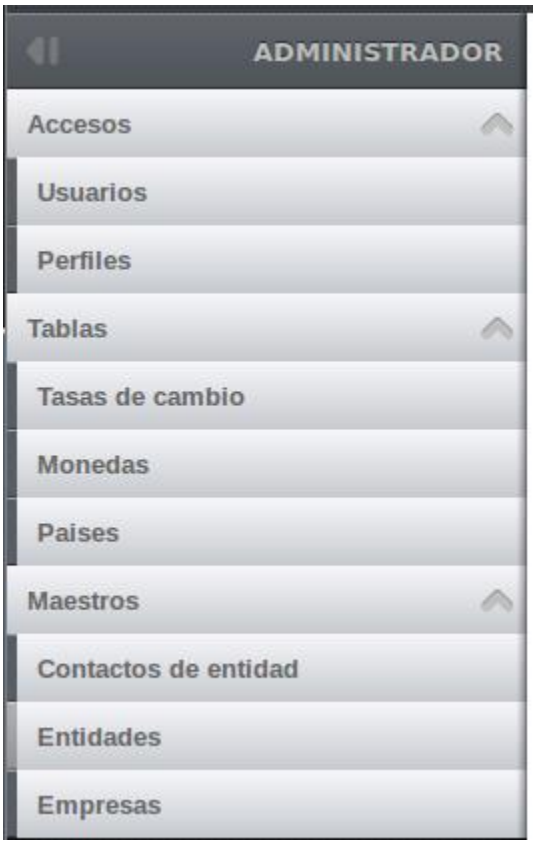
Figura No.16. Construcción de menú a través de inyector de dependencias

```
232
233     <!-- Sidebar/drop-down menu -->
234
235     {% block modulo_menu_contenedor %}
236     <section id="menu" role="complementary">
237         <div id="menu-content">
238             <header>{{ app.session.get('_gentium.perfil','layout.no_perfil'|trans({}, 'AdministracionGeneralSecurity')) }}</header>
239             <section id="modulo_menu_contenedor">
240                 {% block module_menu %}
241                 {{
242                     knp_menu_render('gentium.modulo_menu', {
243                         'template': 'GentiumAdministracionGeneralBundle:Menu:modulo_menu.html.twig',
244                         'currentClass': 'collapsible-current current grey-gradient'
245                     })
246                 }}
247                 {% endblock module_menu %}
248             </section>
249         </div>
250     </section>
251     {% endblock modulo_menu_contenedor %}
252
253     <!-- End sidebar/drop-down menu -->
254
```

El resultado de este desarrollo se visualizará más adelante en un menú dinámico generado con base al perfil del usuario tal y como se muestra en la **Figura No.17**.

[35] https://github.com/KnpLabs/KnpMenuBundle

Figura No.17. Menú dinámico generado con base a perfil de usuario



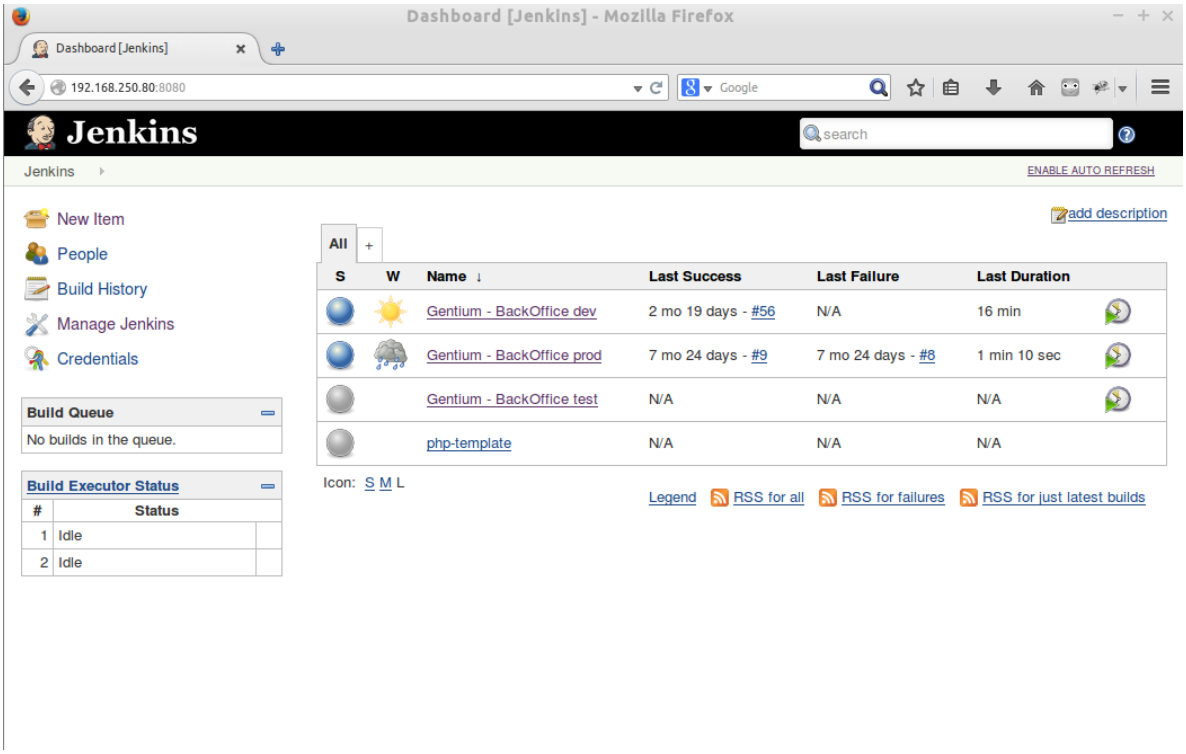
b. Instalación y configuración de automatización para entrega continua. Llegados a este punto del proyecto, se estima conveniente proceder a configurar la automatización de una entrega (*deployment*) local, que se mantenga siempre al día con el desarrollo y que pueda ser utilizada por la administración del proyecto para evaluar el estado y el desarrollo del mismo.

Con este propósito se instala dentro del servidor local la herramienta de integración continua **Jenkins**. Esta aplicación permite crear elementos (*items*) o tareas de integración, y programar para cada una la frecuencia con que se realizarán o los eventos que la dispararán. Además, a pesar de ser PHP un lenguaje interpretado que no requiere compilación, esta herramienta resulta de utilidad para ejecutar sobre el código una serie de pruebas de calidad, de acoplamiento (*compliance*) a estándares de programación y de otros factores que pueden ser de utilidad para indicar el estado del proyecto en cualquier momento.

Esta herramienta **Java** se instala dentro del servidor de desarrollo, y se configura para trabajar a través del puerto 8080 con el servidor de aplicaciones **Jetty**[36], que viene incluído dentro del paquete.

En la **Figura No.18** se muestra una captura de pantalla de la herramienta instalada en el servidor.

Figura No.18. Instalación local de entorno Jenkins



Para configurar los proyectos de PHP se utiliza la plantilla **php-template**[37], distribuida como código abierto, que nos brinda la posibilidad de *orquestar* las distintas tareas que se ejecutarán sobre cada consolidación (*build*). A pesar de que la herramienta nos permite configurar una ámplia gama de tareas, se elige configuran para el proyecto únicamente las tareas de pruebas automatizadas (utilizando **PHPUnit**), y de detección de violación de estándares de código (por medio de **PHP_CodeSniffer**).

Los elementos creados dentro de **Jenkins** se configuran para responder al evento

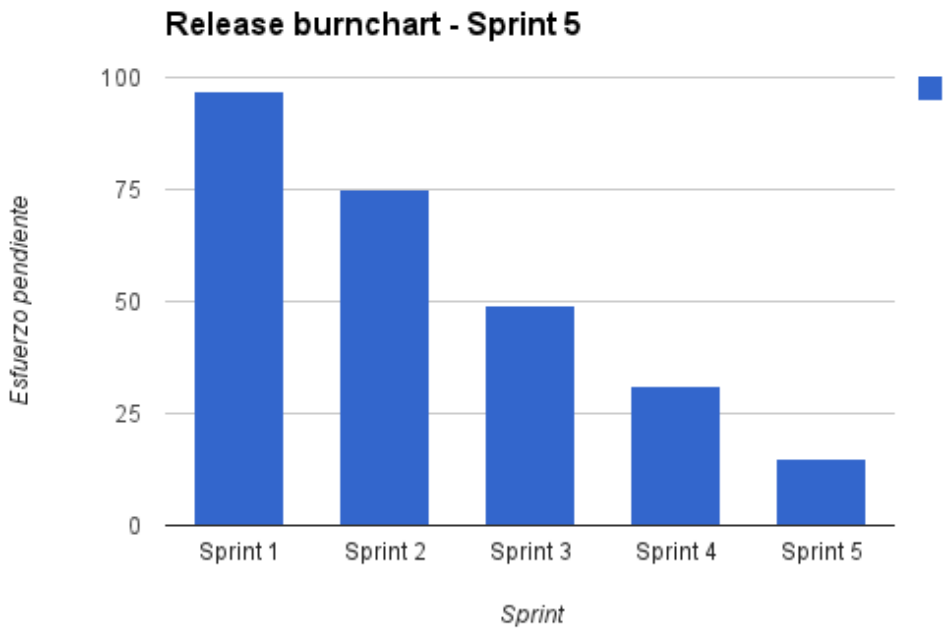
[36] http://www.eclipse.org/jetty/
[37] http://jenkins-php.org/

de un envío (*commit*) utilizando la funcionalidad de ganchos (*hooks*) de **Subversion**[38]. De acuerdo con esta configuración, al detectar el control de versiones el ingreso de un nuevo *commit*, es disparada una notificación sobre **Jenkins** que realiza un nuevo *build* sobre la versión del *software* recién actualizada.

Además, aprovechando la funcionalidad de **Jenkins** de configurar *scripts* de *bash* previa o posteriormente a la ejecución de un *build*, resulta conveniente configurar los distintos items para disparar la actualización de dos instalaciones creadas sobre el mismo servidor, que actualizan el código de acuerdo al branch configurado. Esta actualización implica un *update* de **Subversion** y una migración **Propel** de acuerdo al nuevo esquema de base de datos. Cabe destacar que cualquier otra tarea que la actualización implique deberá ser realizada manualmente en la copia de trabajo (*working copy*) correspondiente dentro del servidor.

5. Sprint 5 – Interfaz gráfica y ofuscamiento de código. Al inicio de esta iteración se presenta el *release burndown* mostrado en la **Figura No.19**.

Figura No.19. *Release burndown* de iteración 5



[38] http://svnbook.red-bean.com/nightly/en/svn.reposadmin.create.html

A pesar de que el tema del aspecto visual no es considerado dentro de las historias de usuario, se plantea como objetivo de esta iteración implementar y detallar este punto dentro de la aplicación con los módulos trabajados, dada la importancia que tiene para efectos de la interacción humano-computador.

Por otro lado, se detecta que, aunque el desarrollo está orientado a proveer el *software* como servicio (*SaaS*), una parte considerable de las negociaciones de la empresa implican la instalación de las aplicaciones dentro de las instalaciones locales de sus clientes. De aquí surge la necesidad de evaluar alternativas que permitan restringir el acceso al código fuente, para evitar modificaciones y apropiamientos no deseados. Esta necesidad se ve reflejada en la **Historia de usuario 11**.

Por tanto, la iteración puede condensarse en:

- Evaluar, adquirir e implementar los elementos de una plantilla gráfica
- Evaluar, adquirir e implementar alternativas de restricción para código fuente

a. Elementos de interfaz gráfica. Dada la orientación de la empresa a desarrollo, sin fuerte particular en cuestiones de diseño, se sigue la alternativa de adquirir una plantilla gráfica elaborada por terceros, y ajustarla a los módulos de la empresa que se van trabajando.

Con este objetivo, se trabaja a través de **Themeforest**[39] la evaluación de una serie de plantillas gráficas que se adapten a los requerimientos del proyecto. Luego de dicha evaluación se elige trabajar con la plantilla gráfica **Developr**[40], debido a que se juzga como la más completa, cuenta con extensa documentación y ejemplos de implementación y además ofrece la ventaja de ser *responsive*, es decir capaz de adaptarse al medio en donde se despliega (tableta, teléfono, computadora, etc.).

La librería se descarga como una serie de hojas de estilo (CSS) y *scripts* para navegador (Javascript) que resultan en el despliegue armonizado del diseño (*layout*) de las páginas. La integración de esta plantilla al sistema se realiza mediante la

[39] http://themeforest.net/
[40] http://themeforest.net/item/developr-fully-responsive-admin-skin/2085628

implementación de un *bundle* (módulo) adicional que incluye las dependencias y librerías. En este punto se determina que la plantilla no cuenta con una herramienta para actualizar las dependencias que requiere (otras librerías de CSS y Javascript), por lo que es desglosada dentro del módulo para poder trabajar la actualización de dichas librerías a través de **Composer**.

En la **Figura No.20** y **Figura No.21** puede observarse el resultado de la aplicación de la plantilla gráfica al desarrollo del proyecto, tanto en la parte de autenticación como dentro del módulo desarrollado.

Figura No.20. Aplicación de plantilla gráfica en autenticación

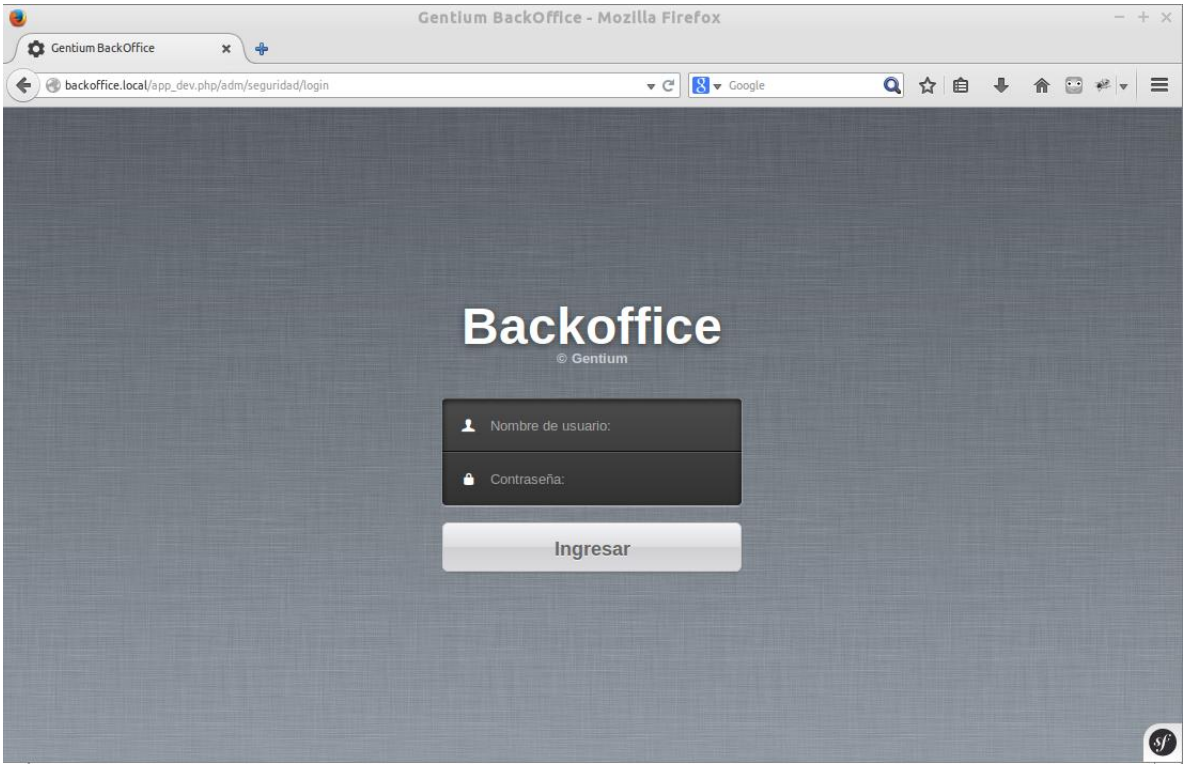
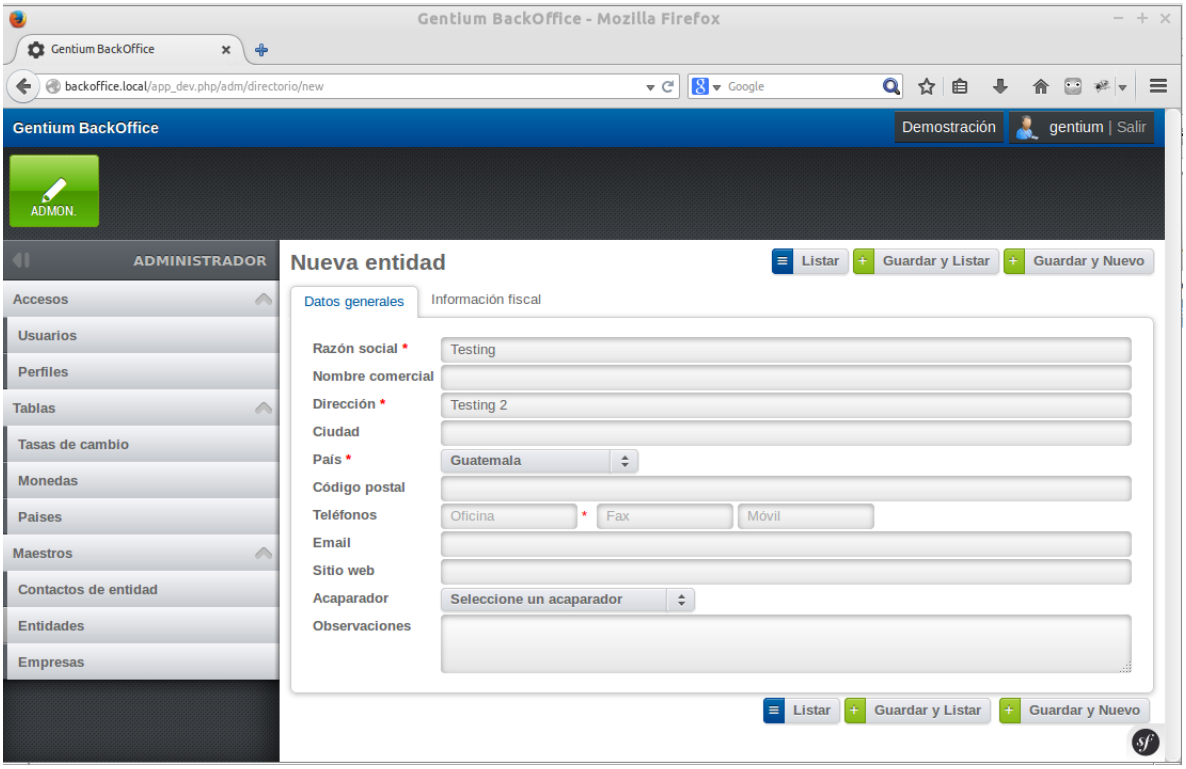


Figura No.21. Aplicación de plantilla gráfica en módulo desarrollado



b. Herramientas para restricción de código fuente. Durante esta última fase del proyecto se considera el requerimiento denotado por la **Historia de usuario 11**, según la cual se solicita una forma de proteger el código fuente de la aplicación.

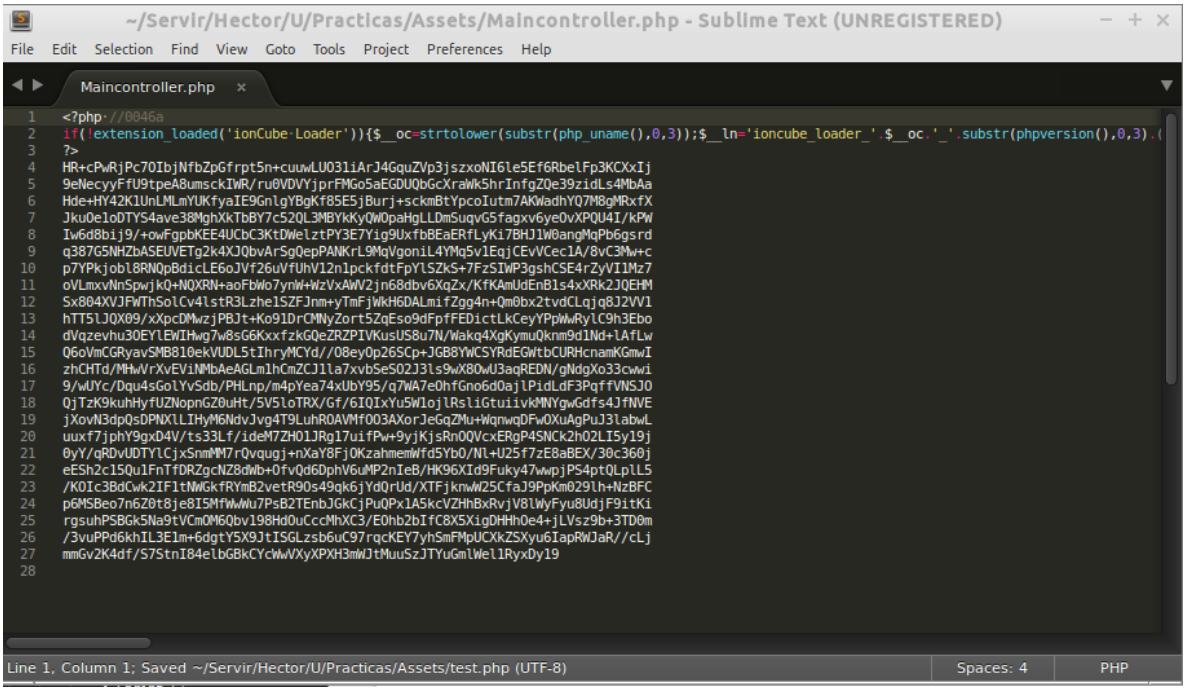
Dado que el desarrollo principal está realizado sobre PHP, se elige por priorizar la protección de este código. Esto presenta el inconveniente de que, al ser PHP un lenguaje de código abierto y orientado principalmente a proyectos de esta naturaleza, no cuenta con una herramienta por defecto para llevar a cabo su *protección*. Sin embargo, luego de evaluar el escenario se encuentra algunas herramientas que permiten cierto grado de restricción sobre el acceso y la ejecución del mismo.

La herramienta seleccionada para implementar esta tarea es **Ioncube**. Esta es una herramienta comercial que requiere la adquisición de una licencia, que ofrece amplio soporte de parte de la empresa que la promueve, y que está enfocada principalmente para trabajo sobre línea de comando.

Estas ventajas permiten desarrollar un *script* dentro del servidor que automatiza el ofuscamiento del proyecto mediante un simple comando. Este *script* desarrollado considera aquellas partes de la aplicación que no deben ser ofuscadas debido a la función que cumplen dentro del *software*, así como la configuración necesaria que debe ser aplicada según las librerías y *frameworks* utilizados.

En la **Figura No.22** es posible visualizar el resultado del proceso de ofuscamiento hacia ASCII (es posible configurar el proceso para producir binario).

Figura No.22. Aplicación de ofuscamiento ASCII a código fuente



El uso de esta herramienta implica que todo servidor en donde la aplicación vaya a ser ejecutada requiere la extensión que **Ioncube** ofrece para interpretar código ofuscado, así como un archivo extendido de parte de la empresa que representa una licencia de uso del *software*. Si estas dos condiciones no se cumplen, el servidor web devolverá un mensaje de error para cualquier petición.

Como parte de este trabajo se advierte que no todo el *software* debe ni puede ser ofuscado utilizando únicamente **Ioncube**. Por un lado, las librerías y *frameworks*

de código abierto que soportan al producto no deben ser ofuscadas debido a su misma naturaleza. Por otro lado, archivos de texto plano de configuración, hojas de cascada de estilos y archivos Javascript no pueden ser ofuscados por medio de este paquete.

La evaluación e implementación de herramientas que cubran este requerimiento, así como la integración de este proceso como parte de la consolidación (*building*) del *software* dentro de **Jenkins** no alcanzan a ser completadas dentro de esta iteración por cuestiones temporales.

VI. Conclusiones y recomendaciones

1. Durante el presente proyecto se desarrolló una base de código que incluye la funcionalidad común que la empresa ofrece a sus clientes en la mayoría de sus productos.

2. La base de código desarrollada incluye los lineamientos para el desarrollo de futuros módulos, así como la estructura para integrarlos dentro del desarrollo realizado y reutilizar así el código hecho.

3. Se realizó un análisis y descripción de las tecnologías utilizadas, así como la documentación mediante un diagrama entidad relación del modelo de base de datos implementado. Además se documentó la arquitectura modular que el sistema buscó alcanzar.

4. La base de código desarrollada fue implementada sobre herramientas de código abierto con soporte planificado a mediano plazo. de acuerdo con lo solicitado por la empresa. Esta base de código incluye un módulo inicial de administración de los objetos de dominio principales.

5. Como parte del desarrollo se instalaron y configuraron una serie de herramientas que facilitan prácticas adecuadas de ingeniería de software: sistema de control de versiones, entorno de integración continua que involucra la automatización de pruebas unitarias y funcionales y sistema de seguimiento a asuntos.

6. El desarrollo fue guiado por un subconjunto de las prácticas sugeridas por las metodologías ágiles Scrum y XP. Estas prácticas incluyen la elaboración y mantenimiento de un *product backlog*, definición y trabajo sobre *sprints*, presentación de diagramas *release burndown*, desarrollo basado en pruebas y la configuración de un entorno de integración continua.

7. Algunos aspectos de las metodologías de desarrollo elegidas no pudieron ser implementadas debido a limitaciones inherentes al desarrollo del proyecto de

práctica profesional: programación en parejas (*pair programming*) sugerida por **XP**, subdivisión en roles como *scrum master* y *developer team*, así como *planning poker* para la estimación de complejidad y los *daily standups* grupales sugeridos por **Scrum**.

8. En cuanto a los aspectos técnicos de la base de código desarrollada, se recomienda evaluar la implementación de ACLs (*Access control lists*) dentro de los módulos, para ofrecer la posibilidad de restringir acceso a objetos particulares dentro del sistema.

9. En cuanto a la integración continua del proyecto, se recomienda implementar métricas de aspectos no considerados tales como el desempeño del sistema (*performance*).

VII. Bibliografía

1. Bourque, P., Fairley, R., Abran, A., Garbajosa, J. & Keeni, G. (2014). *Guide to the software Engineering Body of Knowledge, SWEBOK v.3*. IEEE Computer Society.

2. Calo, K., Estevez, E., & Fillottrani, P. (2010). A Quantitative Framework for the Evaluation of Agile Methodologies. *Journal Of Computer Science & Technology (JCS&T)*, *10*(2), 68-73.

3. Carstensen, J., Golden, B., & Morgenthal, J. (2012). *Cloud Computing: Assessing the Risks*. Ely: IT Governance Publishing.

4. Ceri, S. (2003). *Designing Data-intensive Web Applications*. San Francisco, Calif: Morgan Kaufmann.

5. Coronado, C. (2012). *Implementación de un sistema manejador de proyectos de software para la empresa Solución Web*, 16-18.

6. Crawford, B., & de la Barra, C. (2008). Does eXtreme Programming support Collaborative Creativity? *International Multiconference Of Engineers & Computer Scientists 2008*, 1026-1032.

7. Dogša, T., & Batič, D. (2011). The effectiveness of test-driven development: an industrial case study. *Software Quality Journal*, *19*(4), 643-661. doi:10.1007/s11219-011-9130-2

8. Fowler, M., *Continuous Integration*. Consultado 1/9/2014. Sitio web: http://www.martinfowler.com/articles/continuousIntegration.html

9. Koch, A. S. (2005). *Agile Software Development: Evaluating the Methods for Your Organization*. Boston, MA: Artech House.

10. Martin, R. (2012). *The Clean Architecture*. Consultado 24/1/2014. Sitio web: http://blog.8thlight.com/uncle-bob/2012/08/13/the-clean-architecture.html

11. Martin, R. (2014). *The Domain Discontinuity*. Consultado 3/2/2014. Sitio web: http://blog.8thlight.com/uncle-bob/2014/01/27/TheChickenOrTheRoad.html

12. Martin, R. (2010). *The Land that Scrum Forgot*. Consultado 1/2/2014. Sitio web: http://www.scrumalliance.org/community/articles/2010/december/the-land-that-scrum-forgot

13. Paul C. Clements (1996). *A Survey of Architecture Description Languages*. Carnegie Mellon University, Software Engineering Institute. Consultado 30/1/2014. Sitio web: http://www.sei.cmu.edu/library/assets/Survey_of_ADLs.pdf

14. Rao, K., Naidu, G., & Chakka, P. (2011). A Study of the Agile Software Development Methods, Applicability and Implications in Industry. *International Journal Of Software Engineering & Its Applications*, *5*(2), 35-45.

15. Resnick, S., De la Maza, M., & Bjork, A. (2011). *Professional Scrum with Team Foundation Server 2010*. Indianapolis, IN: Wiley Pub.

16. Richardson, L., Ruby, S. (2007). *RESTful Web Services*. O'Reilly Media, Inc.

17. Romero, P. (2006). *Arquitectura de software, esquemas y servicios*. *Ingeniería Industrial*, 27(1), 19-21.

18. Sinnett, W. M. (2010). *Software As a Service: Experiences of SMBs*. Morristown, N.J.: Financial Executives Research Foundation.

19. Sutherland, J., *et. al.* (2011) *The Power of Scrum*. Kindle Edition. Sitio web: http://www.amazon.com/Power-Scrum-Jeff-Sutherland-ebook/dp/B007474YMC/ref=tmm_kin_swatch_0?_encoding=UTF8&sr=8-1&qid=1420733456

20. The Visual FoxPro Team. *A message to the Comunity*. Visual FoxPro Developer Center. Consultado 6/4/2014. Sitio web: http://msdn.microsoft.com/en-

us/vfoxpro/bb308952.aspx

21. Tojín, K. (2013). *Análisis y desarrollo del Sistema de Rocas para la empresa EBclosion*, 22-23.

22. Tolfo, C., Wazlawick, R., Ferreira, M., & Forcellini, F. (2011). Agile methods and organizational culture: reflections about cultural levels. *Journal Of Software Maintenance & Evolution: Research & Practice, 23*(6), 423-441. doi:10.1002/smr.483

23. W. (1999). *Dictionary of Multimedia & Internet Applications*, 345-353. Wiley Online Library. Sitio web: http://onlinelibrary.wiley.com/book/10.1002/0470841788

VIII. Apéndice: Historias de usuario

Tabla No.1. Historia de usuario 1

Historia de usuario 1:	**Manejo de múltiples módulos del sistema**
Rol:	*Desarrollador*
Prioridad:	*Alta*
Puntos de complejidad:	*5*
Descripción:	*Debo tener la posibilidad de instalar al cliente únicamente los módulos que ha adquirido, y el sistema debe reconocer esta configuración para funcionar correctamente.*

Tabla No.2. Historia de usuario 2

Historia de usuario 2:	***Manejo de múltiples empresas***
Rol:	*Usuario*
Prioridad:	*Alta*
Puntos de complejidad:	*4*
Descripción:	*Debo tener la posibilidad de manejar múltiples empresas, para llevar independientemente el control de las distintas unidades de negocio en un grupo empresarial.*

Tabla No.3. Historia de usuario 3

Historia de usuario 3:	***Manejo de permisos de acceso por módulo (autenticación y autorización)*** → ***Perfiles de usuario***
Rol:	*Administrador*
Prioridad:	*Alta*
Puntos de	*8*

complejidad:	
Descripción:	*Debo tener la posibilidad de configurar perfiles de usuarios del sistema, definiendo qué módulos y vistas en la instalación pueden acceder, para permitir crear distintos roles de trabajo en asociación con el sistema.*

Tabla No.4. Historia de usuario 4

Historia de usuario 4:	***Manejo de múltiples usuarios por empresa y perfil***
Rol:	Administrador
Prioridad:	*Alta*
Puntos de complejidad:	*6*
Descripción:	*Debo tener la posibilidad de manejar múltiples usuarios, asignados a distintas empresas, para que un usuario pueda interactuar únicamente con la información de las empresas asignadas.*

Tabla No.5. Historia de usuario 5

Historia de usuario 5:	***Manejo de directorio de entidades con contactos***
Rol:	*Usuario*
Prioridad:	*Alta*
Puntos de complejidad:	*6*
Descripción:	*Debo tener la posibilidad de manejar múltiples entidades, posiblemente utilizando una clasificación distinta en cada empresa, y asociar información de múltiples contactos a cada entidad.*

Tabla No.6. Historia de usuario 6

Historia de usuario 6:	***Manejo de menú dinámico con base a permisos asignados***
Rol:	*Usuario*

Prioridad:	*Alta*
Puntos de complejidad:	*10*
Descripción:	*Debo poder ver y acceder únicamente a los módulos configurados para mi entorno de trabajo para poder enfocar mi trabajo en las tareas que me fueron asignadas.*

Tabla No.7. Historia de usuario 7

Historia de usuario 7:	***Contar con instalación actualizada automática para revisión***
Rol:	*Administrador*
Prioridad:	*Alta*
Puntos de complejidad:	*8*
Descripción:	*Debo tener la posibilidad acceder a una instalación del sistema siempre funcional, que se actualice conforme se lleva a cabo el desarrollo para poder evaluar en todo momento los avances y realizar pruebas funcionales*

Tabla No.8. Historia de usuario 8

Historia de usuario 8:	***Acceder a historial detallado de desarrollo***
Rol:	*Administrador*
Prioridad:	*Alta*
Puntos de complejidad:	*4*
Descripción:	*Debo tener la posibilidad acceder a un historial detallado del desarrollo (configuración, código fuente, etc.) para revisar cambios, centralizar el código y poder revertir en caso de cualquier problema.*

Tabla No.9. Historia de usuario 9

Historia de usuario 9:	***Consultar estado de funcionalidad en base a pruebas automatizadas***
Rol:	*Usuario*
Prioridad:	*Alta*
Puntos de complejidad:	*10*
Descripción:	*Debo poder evaluar el avance en el desarrollo en base a herramienta que me permita llevar a cabo y obtener los resultados de un conjunto de pruebas automatizadas para garantizar la correcta orientación del proyecto.*

Tabla No.10. Historia de usuario 10

Historia de usuario 10:	***Consultar estado de problemas reportados o funcionalidades sugeridas***
Rol:	*Administrador*
Prioridad:	*Alta*
Puntos de complejidad:	*4*
Descripción:	*Debo tener la posibilidad visualizar de manera rápida y directa el estado de los distintos problemas reportados o solicitudes de funcionalidades sugeridas durante y posterior al desarrollo para darles seguimiento.*

Tabla No.11. Historia de usuario 11

Historia de usuario 11:	***Instalación y actualización de código ofuscado para clientes***
Rol:	Administrador o desarrollador
Prioridad:	Alta
Puntos de complejidad:	8

Descripción:	Debo tener la posibilidad de instalar y actualizar rápida y fácilmente código que no sea modificable por el cliente para evitar apropiamiento y modificación indebida del código.

IX. Glosario

Active record: patrón de ORM en donde un objeto que envuelve un registro en una tabla de la base de datos o vista encapsula el acceso a la base de datos y agrega lógica de dominio sobre los datos.

Aplicación web: conjunto de herramientas que los usuarios pueden utilizar accediendo a través de un navegador web a un servidor web.

Arquitectura: en el contexto de *software* se refiere a los componentes que comprenden un sistema, la especificación del comportamiento para esos componentes y los mecanismos e interacciones entre ellos.

Computación en la nube: modelo que permite acceso conveniente y sobre demanda a un conjunto compartido de recursos informáticos configurables, que pueden ser rápidamente aprovisionados y liberados con mínimo esfuerzo de administración o servicio de interacción con el proveedor.

Desarrollo orientado a pruebas (TDD): una práctica de desarrollo de *software* donde los casos de prueba son escritos incrementalmente previo a la implementación del código de producción.

Ingeniería de software: aplicación de un acercamiento sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento del *software*.

Integración continua: práctica de desarrollo de *software* en donde los miembros de un equipo integran su trabajo con frecuencia de incluso más de una vez por día.

Modelo-Vista-Controlador (MVC): patrón de diseño concebido para separar mejor y aislar las tres funciones esenciales de una aplicación interactiva: lógica empresarial (modelo), interfaz de presentación al usuario (vista) y el control de la interacción

provocada por las acciones del usuario (controlador).

***Software* como servicio (SaaS)**: es la primera capa de los tres modelos de computación en la nube, y consiste en la capacidad ofrecida al consumidor de utilizar las aplicaciones del proveedor que se ejecutan en una infraestructura en la remota.

ORM: técnica de programación que convierte datos entre sistemas incompatibles a objetos de lenguajes orientados a objetos.

Unit of work: patrón de ORM en donde se mantiene una lista de objetos afectados por una transacción y se coordina la escritura de dichos cambios y la resolución de problemas de concurrencia.