Universidad de las Ciencias Informáticas

Facultad 3

Desarrollo de la lógica de negocio en los módulos de Contratación y Administración-Configuración para el Sistema Convenio Integral de Cooperación Cuba-Venezuela.

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

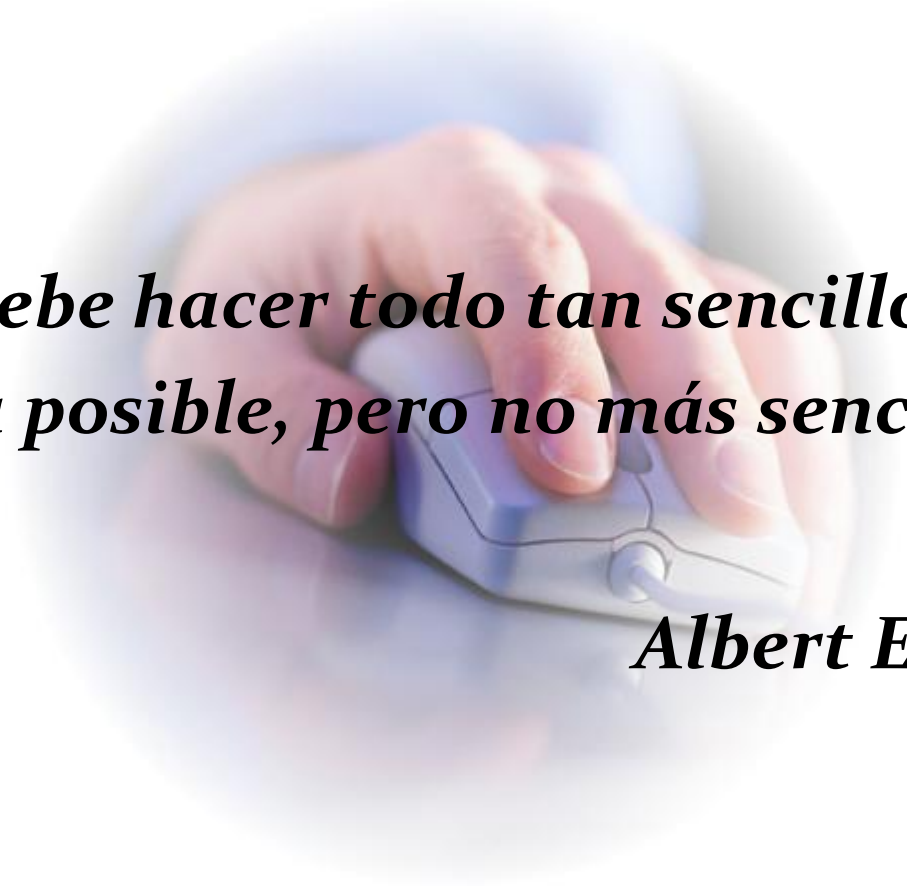
Autores: Yovany Cancio Paz

Andy Osvaldo Durán Rodriguez

Tutor: Ing. Yaniet Piñeiro Pérez

Ciudad de la Habana, Cuba

Mayo, 2009.



“Se debe hacer todo tan sencillo como sea posible, pero no más sencillo.”

Albert Einstein.

DECLARACIÓN DE AUTORÍA

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

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

Yovany Cancio Paz	**Andy Osvaldo Durán Rodriguez**	**Yaniet Piñeiro Pérez**
_____	_____	_____
Firma del Autor	Firma del Autor	Firma del Tutor

De Yovany:

…Primero a nuestro Dios, el Señor Jesucristo…, a mi mamá Cristina, gracias a ella estoy aquí…, a mi papá Francisco, a mi exnovia Yusleisy por soportarme tantos años…, a mis abuelos Olga y Pepe, a mis tías Esther, Marina y Antonia, a mi primo Roddy, a mi tía Grisel, a Yeny…por su ayuda incondicional, a Yeny Vega, a mi compañero de Tesis Andy, a mis amigos de la UCI, Tony, Ramón, Yoan, el Yoyo, Nidia, Rolian, Li@, Lilier, Yainelys, el Core, Pepe, Alberto La Rosa, Yuli, Hirán, el Primo, Marin, George, Tomás, Borroto, Ornolis, Martín, … todos los del aula…, a mis amigos Ernesto, Rafael, Ramón, Jaramillo, Kmilo…a nuestro país hermano Venezuela, a nuestra tutora Yaniet Piñeiro, a nuestros profesores y finalmente a la UCI.

De Andy:

Primeramente a Dios, sin él, sin su amor en mi vida nada de lo que he sido, lo que soy, y lo que quiero llegar a ser, sería posible… A mi mamá y mi papá, quienes son la razón de todos mis esfuerzos y alegrías, la razón de mi existencia, quienes día a día me dedican su vida, este triunfo es más de ellos que mío… A mis hermanos, quienes siempre han estado a mi lado guiándome, abriéndome nuevos caminos, apoyándome y constituyendo mi ejemplo a seguir… A Margarita por ser como una madre también para mí… A mis tíos y mis primos, quienes han sabido inculcarme el espíritu de esfuerzo; especialmente a mi primo Luis Orlando y a mi tío Jorge…. A mis abuelitas, y a María Andrea, que aunque ya no estén siento su apoyo…. A mi familia en general y los que no son pero es como si lo fueran, por siempre haber confiado en mí, por el cariño y la buena voluntad… A mis cuñadas Meibys y Dollys… A Suset… A Grettel, por adornarme muchos momentos… A todos mis maestros y profesores, por contribuir en mi formación… A esta nuestra UCI, institución que nos ha acogido como nuestro segundo hogar, y que se nos ha entregado dándonos las mejores oportunidades… A Cuba y Venezuela, que nos brindaron la posibilidad de poner en práctica nuestros conocimientos por la hermandad y la solidaridad… A mis hermanos de crianza… A mis incomparables compañeros de la vocacional… A Yovany, mi compañero de tesis y amigo, con quien he compartido durante toda la carrera, los buenos y malos momentos… A Yaniet, nuestra tutora… A mis excepcionales compañeros de grupo y mis amigos, quienes a lo largo de estos últimos cinco años me han ayudado a ser una mejor persona…A Ana María… A mis amistades todas, quienes me han enseñado a vivir con intensidad y me han regalado lo mejor de sí…

A todos los que están y los que no pudieron… en fin, a todos los que de una manera u otra han contribuido a este especial logro, y que me han hecho sentir privilegiado, de cierta forma, al formar parte en sus vidas… pues jamás dejarán de estar en mi corazón…

A todos los que están a mí alrededor… Gracias…

De Yovany:

A mis padres,
A mis abuelos,
A mis tías,
A Yusleisy,
A todos los que se consideran mis amigos.

De Andy:

A Dios,
A mi mamá y mi papá.
A mis hermanos,
A Margarita,
A mis tíos y primos,
A toda mi familia,
A mis sobrinitos,
A Grettel,
A todos mis amigos y amigas.

Resumen

El presente trabajo de diploma muestra el desarrollo enfocado al rol de programador en cuanto a la implementación de la lógica de negocio en los módulos de Contratación y Administración-Configuración para el Sistema Convenio Integral de Cooperación Cuba-Venezuela (CCV), así como los resultados de la labor desempeñada, de acuerdo con las pruebas y validaciones de los casos de usos de dichos módulos.

El desarrollo de la tarea debe cumplir con los requisitos establecidos por la parte cliente, tributando como resultado al manejo de la lógica de negocio de ambos módulos y además a la correcta integración entre las capas de presentación y acceso a datos.

Palabras Claves

Capa: Conjunto de subsistemas que comparten la misma generalidad y volatilidad de interfaces.

Implementación: Acción de implementar. Este término se refiere a la forma en que se va a codificar la capa de Negocio.

Casos de Usos: Fragmento de funcionalidad de un sistema.

Contenido

Introducción

En octubre de 2000 el Comandante en Jefe Fidel Castro y Hugo Chávez, Presidente de la hermana República de Venezuela, firmaron en el Palacio de Miraflores el Convenio Integral de Cooperación Cuba-Venezuela (CCV), mediante el cual los gobiernos de ambas naciones conscientes de su interés común por promover y fomentar el progreso de sus respectivas economías y las ventajas recíprocas que resultan de una cooperación con miras a cristalizar el avance económico y socio-cultural de ambos países y la integración de América Latina y el Caribe, acordaron elaborar programas y proyectos de cooperación.

Para la ejecución de los diferentes proyectos se pactó contar con la participación de organismos públicos y privados de ambos países, universidades y organizaciones no gubernamentales. Cuba dispuso prestar los servicios y tecnologías que estuvieran a su alcance para apoyar el programa de desarrollo económico y social en Venezuela. Estos programas son definidos cada año a través de las Comisiones Mixtas, que al fungir como mecanismo financiero para los acuerdos a nivel de gobierno, precisan el monto monetario, las especificaciones, regulaciones y modalidades en las que serán entregados. Los bienes y servicios generalmente serán pagados por Venezuela.

En la actualidad el proceso se realiza de forma manual, lo que hace más engorroso y difícil el trabajo de ambas partes en cuanto a este aspecto. Además, esto trae como consecuencia que no se tenga un control estricto y centralizado de los proyectos y contratos que se han firmado, ni del monto en efectivo relativo a estos, debido a los grandes volúmenes de información que se generan. A ello se adiciona la escasa comunicación entre las partes involucradas en la negociación por tratarse de naciones con realidades geográficas diferentes.

Ante tal situación se puede apreciar la necesidad de emplear herramientas bilaterales para efectuar las negociaciones enmarcadas en el Convenio de Colaboración, en virtud de que ampliaría la capacidad y rapidez de respuesta de las organizaciones e instituciones representadas por las partes contractuales (independientemente del nivel) que tienen bajo su responsabilidad la ejecución tanto de los contratos como de los proyectos contenidos en estos.

Con el meridiano propósito de alcanzar y mantener el éxito en las negociaciones entre los dos países es fundamental dar paso a la propuesta de lograr una colaboración estrecha a través de Sistemas de Información unitarios y estandarizados, específicamente de los Sistemas de Gestión de Proyectos (SGP),

para de este modo acelerar la mayor parte de los procesos administrativos que normalmente consumen mucho tiempo en tareas repetitivas, ritualizadas, y a la vez desburocratizar dicho aparato de gestión administrativa a partir de maximizar el flujo de información precisa y oportuna entre las personas encargadas de llevar a cabo eficientemente los diferentes proyectos y contratos.

Como resultado de un análisis minucioso de las características específicas de las distintas herramientas colaborativas o de Gestión de Proyectos ya existentes, se llegó a la conclusión de que estas no brindan las funcionalidades que le dan solución a los requerimientos exigidos por el Convenio Integral de Cooperación Cuba-Venezuela, debido a que los procesos identificados en el marco del Convenio poseen especificidades propias, que no coinciden con los procesos comunes de colaboración o gestión entre otras organizaciones o instituciones, por lo que se decidió desarrollar un SGP propio para CCV que garantice que no ocurran hechos tales como la pérdida de grandes sumas de dinero, así como el no cumplimiento o cumplimiento parcial de la ejecución de los proyectos y contratos aprobados entre ambas partes.
En consecuencia, se le ha asignado a la Universidad de las Ciencias Informáticas la tarea de crear un proyecto para automatizar las actividades que se efectúan antes, durante y después de realizada la Comisión Mixta de cada año para que el convenio fluya óptimamente. Para llevar a cabo esta solución se ha tomado en consideración utilizar la metodología Rational Unified Process (RUP) o Proceso Unificado de Software, la cual propone diferentes roles con las responsabilidades específicas para cada uno de los miembros del equipo de desarrollo. Dentro de ellos, uno de los que juega un papel crucial es el arquitecto, quien se encarga de plantear una arquitectura robusta y capaz de orientar el desarrollo del sistema, al servir de columna vertebral de este y llevar a cabo dos importantes estilos de arquitectura:

1. **El modelo vista controlador:** es la vista donde se maneja toda la visualización de la información, el controlador interpreta las acciones del ratón y el teclado, informando al modelo y/o a la vista para que cambien según resulte apropiado, y el modelo administra el comportamiento y los datos de la aplicación.

2. **La arquitectura en capas:** está basada principalmente en tres capas como son: la capa de presentación, la capa de acceso a datos y la capa intermedia o capa de negocio, encargada de llevar paso a paso toda la lógica del negocio y con ello dar cumplimiento a los requisitos del

sistema. En adición se integra con las demás capas sirviendo de intermediaria entre los datos y la interfaz. Esta capa es desarrollada por el rol de programador.

Por la importancia de dicho Sistema de Gestión es necesario desarrollar una eficiente lógica de negocio, que asegure la calidad requerida por los clientes.

Con anterioridad se hacía mención de la racionalidad que debe signar las relaciones colaborativas cubano-venezolanas y en ese sentido se develan, a raíz de las Comisiones Mixtas, una extensiva gama de contratos para financiar y llevar a cabo proyectos enmarcados en dicho Convenio. En relación con los contratos, se generan grandes volúmenes de información de procesamiento manual, soportada en papeles y almacenada en formato duro en grandes archivos, atesoramiento que obsta cualquier búsqueda o control eficiente y centralizado de los citados contratos. A esta realidad se le reconocen, por añadidura, los muy dilatados procesos para gestionar y firmar los distintos contratos, debido a que todo esto se hace de forma telefónica o vía e-mail. No está diseñado tampoco el dispositivo para la obtención de reportes fidedignos con respecto a estos contratos que puedan ilustrar los estados de ejecución en los cuales se encuentran a fin de establecer una rigurosa supervisión y seguimiento de los mismos.

Por lo ya establecido, se hace necesario construir un producto de software, cuya lógica de negocio posibilite el manejo de toda la información referente a los distintos documentos, así como sea capaz de llevar a sus usuarios por los distintos procesos necesarios y suficientes en la gestión de los diferentes contratos y que estos a su vez, terminen aprobados y firmados o rechazados, y sus proyectos queden sujetos a cronogramas para la ejecución. También es necesario ilustrar el cumplimento de estos contratos a través de simples y claros reportes. De esta forma se pueden gestionar los contratos y cronogramas de manera más eficiente y se facilita la comunicación constante por las partes involucradas, minimizando el tiempo, el costo y los posibles errores en la contratación de proyectos prescritos en el Convenio Integral de Cooperación Cuba-Venezuela.

A partir de lo antes planteado surge el siguiente **Problema Científico de la Investigación**:

¿Cómo desarrollar la lógica de negocio en los módulos de Contratación y Administración-Configuración, para el proyecto CCV?

Objeto de Estudio

Implementación en el Proceso de Desarrollo de Software.

Campo de acción

Implementación de la lógica de negocio en los módulos de Contratación y Administración-Configuración, para el proyecto CCV.

Objetivo

Implementar la capa lógica de negocio en los módulos de Contratación y Administración-Configuración, para el proyecto CCV.

Hipótesis

Si se desarrolla la capa lógica de negocio en los módulos de Contratación y Administración-Configuración, para el proyecto CCV, cumpliendo con los requisitos del sistema, entonces se valida que los proyectos y contratos sigan estrictamente sus procesos y se facilita la integración de las capas de presentación y acceso a datos.

Tareas de la investigación

- Actualización y estudio del estado del arte de los Sistemas de Gestión de Proyectos, profundizando en la lógica de negocio.
- Elaboración para la aplicación de un correcto estándar de codificación.
- Selección y utilización de patrones de diseño para garantizar que la solución cumpla con los estándares.

- Implementación de un conjunto de clases reutilizables para facilitar el posterior mantenimiento del sistema, además de lograr la integración adecuada de la capa del negocio con las capas de presentación y acceso a datos.

- Validación de los resultados obtenidos a través de las pruebas de unidad aplicadas, así como de la aplicación del sistema en general.

La estrategia de investigación a seguir es la exploratoria, pues esta se emplea cuando existe una problemática que está afectando la sociedad y no se tiene una idea clara del asunto en cuestión. Su principal objetivo es familiarizar al investigador con el tema objeto de estudio, la situación en que se encuentra y los métodos y técnicas a utilizar en su ejecución [1]. Por tanto, se realizará un estudio acerca de los sistemas de gestión de proyectos existentes en el mundo para ver las funcionalidades que estos brindan. Además se analizarán las metodologías para el desarrollo de software, los paradigmas de programación, lenguajes de programación, plataforma de desarrollo, los patrones de diseño que pueden ser utilizados, así como los estándares de codificación. Todo lo anteriormente expuesto tiene como fin dar solución de la mejor forma posible al problema planteado.

Con el objetivo de validar metodológicamente la investigación se utilizaron los siguientes métodos científicos:

Teóricos:

Histórico-Lógico: este método permite analizar y conocer la evolución y tendencias actuales en el desarrollo desde el punto de vista del negocio de los Sistemas de Gestión de Proyecto.

Analítico-Sintético: este método facilita después de un minucioso análisis, la elección del *Framework Spring* por sus potencialidades, características tecnológicas y la gran aceptación que posee para el desarrollo de sistemas robustos.

Hipotético-Deductivo: a partir de este método se reafirma que la utilización del *Framework Spring* es la mejor solución para la implementación de la Capa Lógica de Negocios de Sistemas de Gestión de Proyectos, pues brinda muchas facilidades para la integración de estas dos capas.

Inductivo – Deductivo: permite identificar la necesidad de implementar la Capa Lógica de Negocios de un producto robusto y con calidad que soporte grandes flujos de información.

Empíricos:

Observación: este método facilita la abstracción para llevar la situación del objeto de investigación a un plano más real. Ayuda a determinar que las características principales de los SGP son las que necesitaba nuestro sistema, no obstante, al adentrarse un poco más en el trabajo, se pudo observar que ningún SGP, hasta el momento, satisfacía de manera más específica los requerimientos del sistema deseado.

Capítulo 1. Fundamentación Teórica

1.1 Introducción

En el presente capítulo se aborda toda la fundamentación teórica de la investigación realizada, mayormente desde el punto de vista del rol de programador, así como un análisis comparativo sobre las metodologías y paradigmas de programación empleados en la realización del proyecto; además se muestra un estudio sobre la plataforma de desarrollo, herramientas, lenguaje de programación y *frameworks* empleados en la implementación y validación de la capa de negocio de los módulos Contratación y Administración-Configuración del proyecto CCV.

Al finalizar el capítulo se arriba a unas conclusiones donde se evidencia, según el estudio previamente realizado, el ambiente completo de trabajo donde se llevará a cabo la solución.

1.2 Sistemas de Gestión de Proyectos

Se puede afirmar que la capacidad de crecimiento económico no depende exclusivamente de la dimensión de las inversiones, sino también de la calidad de las mismas. Por tanto, se precisa contar con instrumentos idóneos que permitan identificar los proyectos de inversión y seleccionar aquellos que garanticen mayor crecimiento económico empresarial y bienestar para la sociedad. Con tal propósito surgen los Sistemas de Gestión de Proyectos (SGP).

Dichos sistemas son herramientas articuladas en el más profundo aliento de la planeación y se preocupan en la utilización adecuada de los escasos recursos, apostando siempre por objetivos de crecimiento económico y social. Por tanto, para asignar mejor los recursos se requiere mayor información sobre la rentabilidad (financiera, económica y social) de los proyectos e idear mecanismos que permitan programar la inversión en función de dichas rentabilidades. Un genuino sistema entonces, deberá ser garante de estas facilidades.

El acto de compartir información permite organizar y administrar proyectos o tareas de forma más eficiente. De ahí que los sistemas de Gestión no sólo aumentan los niveles de productividad y competitividad de una empresa o negociación, sino que también ayudan a integrar procesos del negocio que a corto plazo conllevan a una gran reducción de costos y a una mejora significativa. Estos son muy aceptados a nivel mundial pues proporcionan a las empresas nuevos métodos de colaboración e integración, a la vez que amplían las posibilidades de comunicación.

Un SGP permite organizar y administrar recursos de manera tal que se pueda culminar todo el trabajo que requiere el proyecto dentro del alcance, el tiempo, y coste definidos.

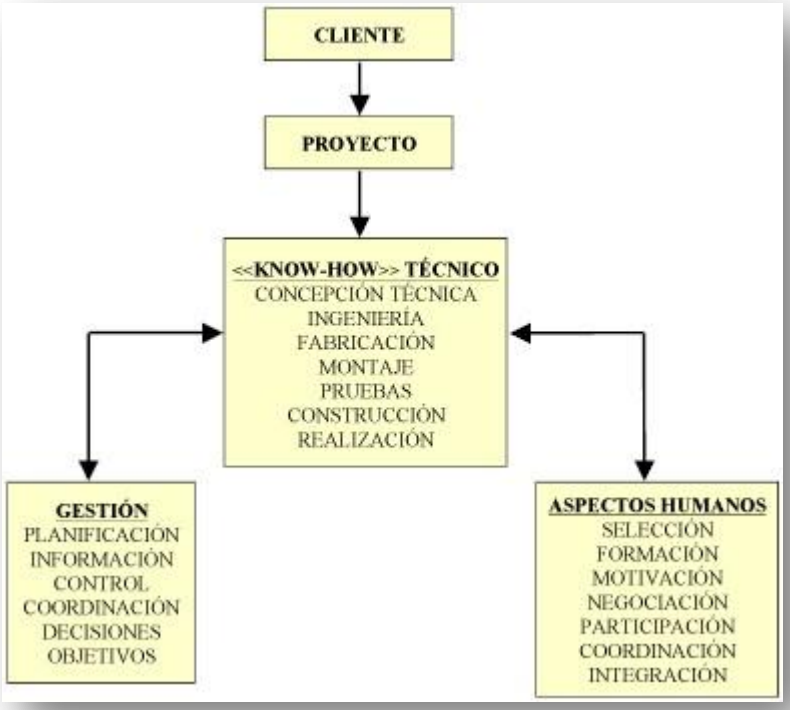


Figura 1.1 Esquema de SGP.

Mucho se ha visto y comentado sobre aplicaciones web destinadas a la gestión de proyectos y sus compromisos en términos de funcionamiento resultan muy aproximados, si se entiende en juego la misma lógica de la iniciación de un proyecto, creación del equipo y las listas de tareas, su vinculación a los

miembros que participan en él, la gestión de archivos, etc., bajo un entorno amigable y sencillo de administrar.

Utilizar las herramientas informáticas de control de proyectos apropiadas y adaptadas a sus propios métodos de trabajo añade valor al proceso de gestionar los proyectos en sí, pero sólo si son fáciles de instalar, mantener y utilizar. Aquí la operatividad resume lo viable y pertinente de su uso.

Los responsables de proyectos o negocios deben saber de inmediato si están percibiendo ganancias o pérdidas en un proyecto determinado y si cumplen con sus plazos o no. ¿Sus recursos están trabajando por debajo o por encima de sus capacidades? ¿Se necesitarán más recursos? Ellos deben conocer sobre la marcha, el estado actual de todos sus proyectos y recursos empleados en los mismos.

Durante los últimos años, muchas empresas han adquirido importantes SGP con la ilusión de que les permitirían resolver todos sus problemas de gestión, ya sea el control de los proyectos o la contabilidad. Sin embargo, al comenzar a utilizarlos, se establecen discrepancias entre los involucrados de acuerdo con el rol o papel que desempeñan estos sistemas en su trabajo como tal. Por eso, a menudo optan por adquirir un gran sistema que necesita una personalización por parte de asesores y apoyo técnico de especialistas en tecnologías de la información que manejan el sistema de gestión.

Los costes de la asesoría y del soporte técnico suelen encarecer en gran medida un sistema, lo que requiere un esfuerzo de gestión demasiado importante y poco a poco, las partes retroceden y vuelven a utilizar una gran cantidad de programas distintos, como Excel®, Word®, Access®, visualizadores y planificadores. Por eso, en la mayoría de los proyectos, los responsables de controlar el trabajo deben analizar un cúmulo de documentos diferentes los que constituyen “islas de información”. Mientras que si contasen con un sistema de gestión, pudieran ver en tiempo real la utilización de los recursos y el estado de los diferentes proyectos en cualquier momento o lugar.

Las propuestas de nuevos proyectos pueden llegar muy rápido, ya sea con carácter nacional o internacional. Sin embargo, el porcentaje de éxito o incluso realización de estos puede ser mínimo. Ello obedece a que es muy difícil llevar a cabo un estricto control, o simplemente no se controlan, debido a lo engorroso que resulta este proceso por los grandes volúmenes de información que se manejan para efectuarlo de no contarse con un SGP.

Una vez aceptadas las propuestas, se convierten en proyectos que deben ser planificados. Las tablas de planificación manuales pertenecen al pasado, puesto que un SGP eficaz se integra en tiempo real, lleva a cabo la planificación según las capacidades de los recursos de su empresa y según lo que debe cumplir. Una vez planificada la realización de todos los componentes del proyecto, empieza el control en tiempo real del mismo. Sus miembros pueden utilizar medios informáticos para establecer un control más efectivo. De este modo, los responsables saben dónde se encuentra el proyecto, ya sea a través de los hitos marcados por la ejecución financiera o ejecución física y si se cumple o no con sus plazos.

Hoy, se puede apreciar una nueva oleada de SGP los cuales se pueden convertir en sistemas de gestiones completas, rentables y según la evolución de las necesidades de las partes involucradas. [2]

Por estas razones es que se ha decidido construir un SGP que permita integrar a los actores involucrados en los diferentes proyectos de la organización, así como administrar la información de los planes operativos, monitorear la ejecución de tareas y facilitar el seguimiento y evaluación de los proyectos y contratos.

Sin dudas, los SGP presentan múltiples ventajas, las cuales son necesarias aprovechar:

- Agilizan los procesos de carga y administración de la información crítica, disminuyendo los tiempos.
- Centralizan los recursos en una base de datos on-line.
- Permiten la carga y visualización de los datos generales del proyecto, los planes operativos y los contratos totalmente on-line.
- Incorporan en un disco virtual documentos de diferente formato que sean complementarios o de utilidad para el monitoreo del proyecto.
- Pueden integrarse a otros sistemas de administración.
- Operan íntegramente a través de Internet desde cualquier computadora, evitando la instalación de software específico.
- Permiten niveles de acceso al sistema de acuerdo con las necesidades de cada uno de los actores involucrados en el sistema, así como la administración óptima tanto de la información como del sistema en sí.
- Facilitan la organización pues evitan la distorsión de la información referente a los proyectos y

contratos disponiendo del monitoreo físico de los proyectos.

- Permiten realizar un seguimiento activo de los proyectos, además disponer de alarmas tempranas e identificar las responsabilidades en el proceso de gestión.

1.3 Metodologías de Desarrollo

Las metodologías de desarrollo de software son un conjunto de procedimientos, técnicas y ayudas a la documentación para el desarrollo de productos software.

Es como un libro de recetas de cocina, en el que se van indicando paso a paso todas las actividades a realizar para lograr el producto informático deseado, indicando además qué personas deben participar en el desarrollo de las actividades y qué papel deben de tener. Además detallan la información que se debe producir como resultado de una actividad y la requerida para comenzarla. [3]

Las técnicas indican cómo debe ser realizada una actividad determinada e identificada en la metodología. Combinan el empleo de unos modelos o representaciones gráficas junto con el de unos procedimientos detallados. Se debe tener en consideración que una técnica determinada puede ser utilizada en una o más actividades de la metodología de desarrollo de software. Además se debe tener mucho cuidado cuando se pretende reemplazar una técnica por otra.

También se entiende por metodología de desarrollo una colección de documentación formal referente a los procesos, las políticas y los procedimientos que intervienen en el desarrollo del software [en Inglés *software development methodology* (SDM) o *system development life cycle* (SDLC)].

La finalidad de una metodología de desarrollo es garantizar la eficacia (ejemplo: cumplir los requisitos iniciales) y la eficiencia (ejemplo: minimizar las pérdidas de tiempo) en el proceso de generación de software. El núcleo de cualquier metodología de desarrollo se encuentra constituido por documentos escritos que detallan cada uno de los puntos expuestos.

Como se refería en la Introducción, para este caso en particular se ha decidido utilizar RUP como Metodología de Desarrollo, no obstante, también se acordó realizar un estudio del funcionamiento de otra metodología, XP, con el fin de poder contar con algunas ventajas que esta ofrece.

1.3.1 XP (Extreme Programming)

XP o Programación Extrema es un enfoque de la ingeniería de software formulado por Kent Beck, autor del primer libro sobre la materia, *Extreme Programming Explained: Embrace Change* (1999). Es el más destacado de los procesos ágiles de desarrollo de software. Al igual que estos, la programación extrema se diferencia de las metodologías tradicionales principalmente en que pone más énfasis en la adaptabilidad que en la previsibilidad. Los defensores de XP consideran que los cambios de requisitos sobre la marcha son un aspecto natural, inevitable e incluso deseable del desarrollo de proyectos. Ellos creen que ser capaz de adaptarse a los cambios de requisitos en cualquier punto de la vida del proyecto es una aproximación mejor y más realista que intentar definir todos los requisitos al comienzo del proyecto e invertir esfuerzos después en controlar los cambios en los requisitos.[4]

Se puede considerar la programación extrema como la adopción de las mejores metodologías de desarrollo según lo que se pretende llevar a cabo con el proyecto, y aplicarlo de manera dinámica durante el ciclo de vida del software.

Los principios básicos de la programación extrema son los siguientes:

- **Simplicidad:** La simplicidad es la base de la programación extrema. Se simplifica el diseño para agilizar el desarrollo y facilitar el mantenimiento. Un diseño complejo del código junto a sucesivas modificaciones por parte de diferentes desarrolladores hace que la complejidad aumente exponencialmente. Para mantener la simplicidad es necesaria la refactorización del código, esta es la manera de mantener el código simple a medida que crece. También se aplica la simplicidad en la documentación, de esta manera el código debe comentarse en su justa medida, intentando eso que el código esté autodocumentado. Para ello se deben elegir adecuadamente los nombres de las variables, métodos y clases. Los nombres largos no disminuyen la eficiencia del código ni el tiempo de desarrollo, gracias a las herramientas de autocompletado y refactorización que existen actualmente. Aplicando la simplicidad junto con la autoría colectiva del código y la programación por parejas se asegura que mientras más grande se haga el proyecto, todo el equipo conocerá más y mejor el sistema completo.

- **Comunicación:** La comunicación se realiza de diferentes formas. Para los programadores el código comunica mejor mientras más simple sea. Si el código es complejo hay que esforzarse para hacerlo inteligible. El código autodocumentado es más fiable que los comentarios ya que estos últimos pronto quedan desfasados con el código a medida que es modificado. Debe comentarse solo aquello que no va a variar, por ejemplo, el objetivo de una clase o la funcionalidad de un método. Las pruebas unitarias son otra forma de comunicación ya que describen el diseño de las clases y los métodos al mostrar ejemplos concretos de cómo utilizar su funcionalidad. Los programadores se comunican constantemente gracias a la programación por parejas. La comunicación con el cliente es fluida en tanto este forma parte del equipo de desarrollo. El cliente decide qué características tienen prioridad y siempre debe estar disponible para solucionar dudas.

- **Retroalimentación (feedback):** Al estar el cliente integrado en el proyecto, su opinión sobre el estado del mismo se conoce en tiempo real. Al realizarse ciclos muy cortos tras los cuales se muestran resultados, se minimiza el tener que rehacer partes que no cumplen con los requisitos y ayuda a los programadores a centrarse en lo que es más importante. Considérense los problemas que derivan de tener ciclos muy largos. Meses de trabajo pueden echarse por la borda debido a cambios en los criterios del cliente o malentendidos por parte del equipo de desarrollo. El código también es una fuente de retroalimentación gracias a las herramientas de desarrollo. Por ejemplo, las pruebas unitarias informan sobre el estado de salud del código. Ejecutar las pruebas unitarias frecuentemente permite descubrir fallos debidos a cambios recientes en el código.

- **Coraje o Valentía:** Los puntos anteriores parecen tener sentido común, entonces, ¿por qué coraje? Para los gerentes la programación en parejas puede ser difícil de aceptar, parece como si la productividad se fuese a reducir a la mitad, toda vez que solo la mitad de los programadores está escribiendo código. Hay que ser valiente para confiar en que la programación por parejas beneficia la calidad del código sin repercutir negativamente en la productividad. La simplicidad es uno de los principios más difíciles de adoptar. Se requiere coraje para implementar las características que el cliente quiere, ahora sin caer en la tentación de optar por un enfoque más flexible que permita futuras modificaciones. No se debe emprender el desarrollo de grandes marcos de trabajo mientras el cliente espera. En ese tiempo el cliente no recibe noticias sobre los

avances del proyecto y el equipo de desarrollo no recibe retroalimentación para saber si va en la dirección correcta. La forma de construir marcos de trabajo es mediante la refactorización del código en sucesivas aproximaciones.

1.3.2 RUP (Rational Unified Process)

RUP o Proceso Unificado de Rational, describe cómo aplicar efectivamente enfoques comprobados comercialmente para el desarrollo de software. Estos enfoques son llamados "mejores prácticas" pues son utilizados en la industria por organizaciones exitosas.

RUP provee a cada miembro del equipo de las guías de proceso, plantillas y mentores de herramientas necesarias para que el *team* completo tome ventaja de las mejores prácticas:

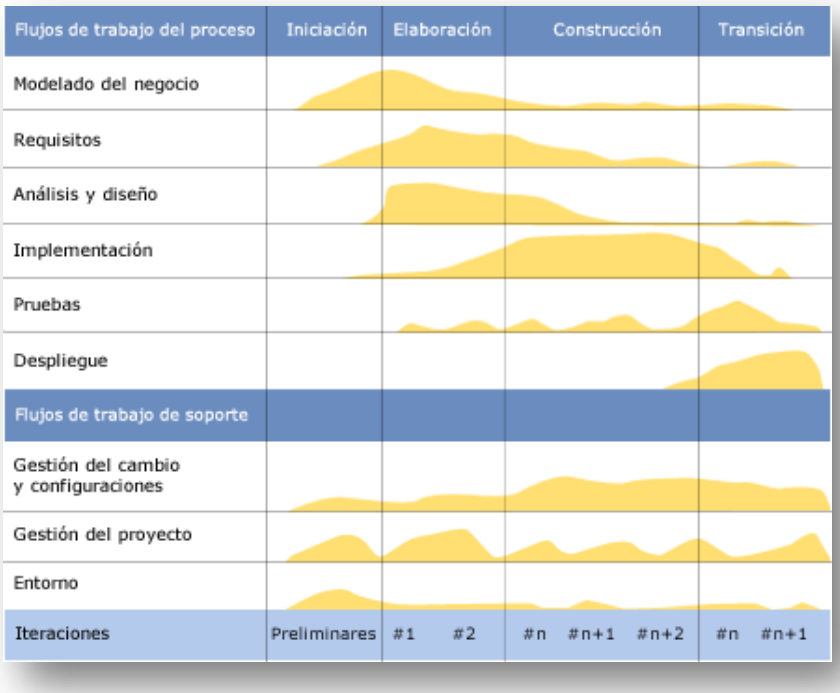


Figura 1.2 Esquema de RUP

Como se puede apreciar RUP, a partir de sus características, brinda una serie de ventajas que deben ser tomadas en cuenta por el equipo de trabajo.

En función de la cada vez mayor complejidad solicitada para los sistemas de software, ya no es posible trabajar secuencialmente: definir primero el problema completo, luego diseñar toda la solución, construir el software y finalmente, testear el producto. Es necesario un enfoque iterativo, que permita una comprensión creciente del problema a través de refinamientos sucesivos, llegando a una solución efectiva luego de múltiples iteraciones acotadas en complejidad.

RUP utiliza y soporta este enfoque iterativo que ayuda a atacar los riesgos mediante la producción de *releases* ejecutables progresivos y frecuentes que permiten la opinión e involucramiento del usuario.

A través de las iteraciones que generan *releases* ejecutables, se logra detectar en forma temprana los desajustes e inconsistencias entre los requerimientos, el diseño, el desarrollo y la implementación del sistema, manteniendo al equipo de desarrollo focalizado en producir resultados.

Los requerimientos son las condiciones o capacidades que el sistema debe conformar. RUP, por su parte, facilita la Administración de Requerimientos que no es más que un enfoque sistemático para hallar, documentar, organizar y monitorear los requerimientos cambiantes de un sistema.

Las nociones de casos de uso y de escenarios utilizadas en RUP han demostrado ser una manera excelente de capturar los requerimientos funcionales y asegurarse que direccionan el diseño, la implementación y la prueba del sistema, logrando así que el sistema satisfaga las necesidades del usuario.

El proceso de software debe focalizarse en el desarrollo temprano de una arquitectura robusta ejecutable, antes de comprometer recursos para el desarrollo a gran escala. RUP describe cómo diseñar una arquitectura flexible, que se acomode a los cambios, comprensible intuitivamente y promueve una más efectiva reutilización de software. Soporta el desarrollo de software basado en componentes: módulos no triviales que completan una función clara. RUP provee un enfoque sistemático para definir una arquitectura utilizando componentes nuevos y preexistentes.

Esta metodología muestra cómo modelar software visualmente para capturar la estructura y comportamiento de arquitecturas y componentes. Las abstracciones visuales ayudan a comunicar

diferentes aspectos del software, comprender los requerimientos, ver cómo los elementos del sistema se relacionan entre sí, mantener la consistencia entre diseño e implementación y promover una comunicación precisa. El estándar UML (Lenguaje de Modelado Unificado), creado por *Rational Software*, es el cimiento para una modelación visual exitosa.

Es necesario evaluar la calidad de un sistema respecto de sus requerimientos de funcionalidad, confiabilidad y performance. La actividad fundamental es el *testing*, que permite encontrar las fallas antes de la puesta en producción. RUP asiste en el planeamiento, diseño, implementación, ejecución y evaluación de todos estos tipos de *testing*. [5]

La capacidad de administrar los cambios es esencial en ambientes en los cuales el cambio es inevitable. RUP describe cómo controlar, rastrear y monitorear los cambios para permitir un desarrollo iterativo exitoso. Es también una guía para establecer espacios de trabajo seguros a cada desarrollador, suministrando el aislamiento de los cambios hechos en otros espacios de trabajo y controlando las transformaciones de todos los elementos de software (modelos, código, documentos, etc.). Detalla cómo automatizar la integración y administrar la conformación de *releases*.

1.3.2.1 Rol de Programador en RUP

Este rol es responsable de los componentes de desarrollo y de prueba, de acuerdo con los estándares aprobados por el proyecto para la integración en subsistemas más grandes. Cuando los componentes de prueba, como controladores o fragmentos para simulación, deben crearse para dar soporte a las pruebas, el implementador también es responsable del desarrollo y las pruebas de los componentes de prueba y los subsistemas correspondientes. [6]

Las habilidades y conocimientos apropiados para el programador incluyen: [6]

- Conocimiento del sistema o aplicación que se somete a prueba.
- Familiaridad con las herramientas de prueba y de automatización de prueba.
- Habilidades de programación.

Principales actividades del Programador: (Figura 1.3) [6]

- Implementar elementos de diseño.
- Implementar pruebas de desarrollador.
- Ejecutar pruebas de desarrollador.
- Implementar elementos de pruebas de diseño.
- Desarrollar artefactos de instalación.
- Analizar el comportamiento de componentes en tiempo de ejecución.

Principales artefactos del Programador: (Figura 1.3) [6]

- Pruebas de desarrollador.
- Elementos de implementación.
- Elementos de implementación. (Prueba)
- Artefactos de instalación.

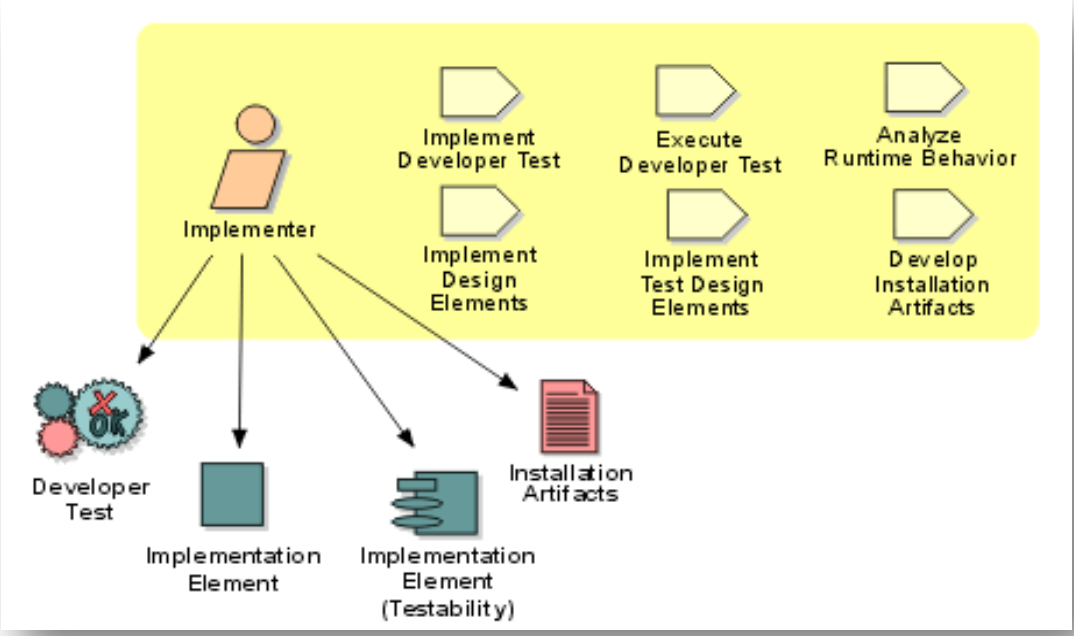


Figura 1.3 Actividades y artefactos relacionados con el rol de Programador.

1.3.3 Conclusiones sobre las metodologías

Como se ha podido apreciar en el análisis realizado anteriormente se ha decidido utilizar RUP, pues constituye una de las metodologías más usadas en el mundo entero (incluyéndose en el programa de estudio de la UCI). Para obtener este SGP con los recursos que se cuenta, RUP ha demostrado ser superior a XP. Sin dudas, XP para nada es una inapropiada metodología, solo que no conviene para este proyecto, pues al ponerla en práctica se debe dedicar todo el tiempo a la programación y como la mayoría de los miembros del equipo de proyecto son estudiantes habría grandes afectaciones para la docencia, además no hay recursos humanos suficientes para llevar a cabo la programación en parejas como lo plantea esta metodología, por lo tanto se pueden utilizar algunos criterios de la Simplicidad y la Comunicación siempre obviando la programación por parejas. También se cuenta con un buen equipo de analistas y los procesos del negocio están bien definidos como para que se haga un excelente análisis y especificación de requerimientos, realidad incomprendida por XP. RUP, por su parte, nos brinda una serie de ventajas que no debemos dejar pasar por alto y que se adecuan a las necesidades del proyecto:

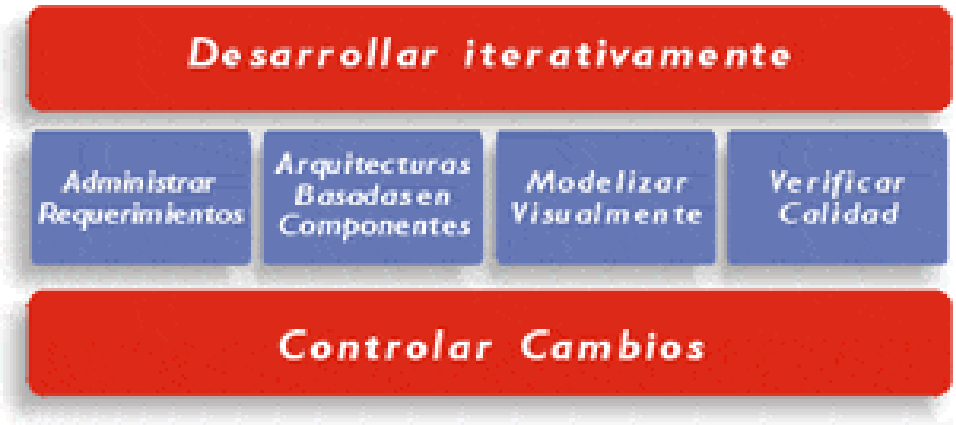


Figura 1.4 Ventajas de la Metodología RUP.

Y por último, también se ha tomado en cuenta que RUP es muy recomendado para proyectos grandes como este que se quiere llevar a cabo, además de las facilidades de protocolo que brinda.

Por estas razones es que se ha puesto de manifiesto una cierta inclinación a favor de la utilización de RUP para la construcción del SGP que se desea desarrollar.

1.4 Paradigmas de Programación

Un paradigma de programación es un modelo básico de diseño y desarrollo de programas, que permite producir programas con unas directrices específicas, tales como: estructura modular, fuerte cohesión, alta rentabilidad, etc. [7]

Un paradigma de programación representa un enfoque particular o filosofía para la construcción del software. No es mejor uno que otro sino que cada uno tiene ventajas y desventajas. También hay situaciones donde un paradigma resulta más apropiado que otro. Si bien puede seleccionarse la forma pura de estos paradigmas al momento de programar, en la práctica es habitual que se mezclen, dando lugar a la programación multiparadigma.

1.4.1 POO (Programación Orientada a Objetos)

La Programación Orientada a Objetos (POO, según sus siglas) es un paradigma de programación que usa objetos y sus interacciones para diseñar aplicaciones y programas de computadora. Está basado en varias técnicas, incluyendo herencia, modularidad, polimorfismo y encapsulamiento. Su uso se popularizó a principios de la década de 1990. Actualmente son muchos los lenguajes de programación que soportan la orientación a objetos.

La POO es una forma especial de programar, más cercana a como expresaríamos las cosas en la vida real que otros tipos de programación.

Con la POO se requiere aprender a pensar las cosas de una manera distinta, para escribir los programas en términos de objetos, propiedades, métodos y otras cosas que se verán rápidamente para aclarar conceptos y dar una pequeña base que permita ambientarse un poco con este tipo de programación.[8]

El elemento fundamental de la POO es, como su nombre lo indica, el objeto. Se puede definir un objeto como un conjunto complejo de datos y programas que poseen estructura y forman parte de una organización. [9]

Los objetos son entidades que combinan estado, comportamiento e identidad:

- El estado está compuesto de datos, será uno o varios atributos a los que se habrán asignado unos valores concretos (datos).
- El comportamiento está definido por los procedimientos o métodos con que puede operar dicho objeto, es decir, qué operaciones se pueden realizar con él.
- La identidad es una propiedad de un objeto que lo diferencia del resto, dicho con otras palabras, es su identificador (concepto análogo al de identificador de una variable o una constante).

La POO expresa un programa como un conjunto de estos objetos que colaboran entre sí para realizar tareas. Esto permite hacer los programas y módulos más fáciles de escribir, mantener y reutilizar.

De esta forma, un objeto contiene toda la información que permite definirlo e identificarlo frente a otros objetos pertenecientes a otras clases, e incluso frente a objetos de una misma clase al poder tener valores bien diferenciados en sus atributos. A su vez, los objetos disponen de mecanismos de interacción llamados métodos que favorecen la comunicación entre ellos. Esta comunicación potencia al mismo tiempo el cambio de estado en los propios objetos. Esta característica lleva a tratarlos como unidades indivisibles, en las que no se separan ni deben separarse el estado y el comportamiento.

Los métodos (comportamiento) y atributos (estado) están estrechamente relacionados por la propiedad de conjunto. Esta propiedad destaca que una clase requiere de métodos para poder tratar los atributos con los que cuenta. El programador debe pensar indistintamente en ambos conceptos, sin separar ni darle mayor importancia a ninguno de ellos, hacerlo podría producir el hábito erróneo de crear clases contenedoras de información por un lado y clases con métodos que manejen a las primeras por el otro. De esta manera se estaría realizando una programación estructurada solapada en un lenguaje de POO.

Esto difiere de la programación estructurada tradicional, en la que los datos y los procedimientos están separados y sin relación, ya que lo único que se busca es el procesamiento de unos datos de entrada para obtener otros de salida. La programación estructurada anima al programador a pensar sobre todo en términos de procedimientos o funciones, y en segundo lugar en las estructuras de datos que esos procedimientos manejan. En la programación estructurada sólo se escriben funciones que procesan datos. Los programadores que emplean este nuevo paradigma, en cambio, primero definen objetos para luego enviarles mensajes solicitándoles que realicen sus métodos por sí mismos.

En comparación con un lenguaje imperativo, una "variable", no es más que un contenedor interno del atributo, del objeto o de un estado interno, así como la "función" es un procedimiento interno del método del objeto.

Hay un cierto desacuerdo sobre exactamente qué características de un método de programación o lenguaje le definen como "orientado a objetos", pero hay un consenso general en que las características siguientes son las más importantes

- **Abstracción**: Denota las características esenciales de un objeto, donde se capturan sus comportamientos. Cada objeto en el sistema sirve como modelo de un "agente" abstracto que puede realizar trabajo, informar y cambiar su estado, y "comunicarse" con otros objetos en el sistema sin revelar cómo se implementan estas características. Los procesos, las funciones o los métodos pueden también ser abstraidos y cuando lo están, una variedad de técnicas son requeridas para ampliar una abstracción.

- **Encapsulamiento**: Significa reunir a todos los elementos que pueden considerarse pertenecientes a una misma entidad, al mismo nivel de abstracción. Esto permite aumentar la cohesión de los componentes del sistema. Algunos autores confunden este concepto con el principio de ocultación, principalmente porque se suelen emplear de conjunto.

- **Principio de ocultación**: Cada objeto está aislado del exterior, es un módulo natural, y cada tipo de objeto expone una interfaz a otros objetos que especifica cómo pueden interactuar con los objetos de la clase. El aislamiento protege a las propiedades de un objeto de su modificación por quien no tenga derecho a acceder a ellas, solamente los propios métodos internos del objeto pueden acceder a su estado. Esto asegura que otros objetos no puedan cambiar el estado interno de un objeto de maneras imprevistas, eliminando efectos secundarios e interacciones inesperadas. Algunos lenguajes relajan esto, permitiendo un acceso directo a los datos internos del objeto de una manera controlada y limitando el grado de abstracción. La aplicación entera se reduce a un agregado o rompecabezas de objetos.

- **Polimorfismo**: comportamientos diferentes, asociados a objetos distintos, pueden compartir el mismo nombre, al llamarlos por ese nombre se utilizará el comportamiento correspondiente al

objeto que se esté usando. O dicho de otro modo, las referencias y las colecciones de objetos pueden contener objetos de diferentes tipos, y la invocación de un comportamiento en una referencia producirá el comportamiento correcto para el tipo real del objeto referenciado. Cuando esto ocurre en "tiempo de ejecución", esta última característica se llama asignación tardía o asignación dinámica. Algunos lenguajes proporcionan medios más estáticos (en "tiempo de compilación") de polimorfismo, tales como las plantillas y la sobrecarga de operadores de C++.

- **Herencia**: las clases no están aisladas, sino que se relacionan entre sí, formando una jerarquía de clasificación. Los objetos heredan las propiedades y el comportamiento de todas las clases a las que pertenecen. La herencia organiza y facilita el polimorfismo y el encapsulamiento permitiendo a los objetos ser definidos y creados como tipos especializados de objetos preexistentes. Estos pueden compartir (y extender) su comportamiento sin tener que volver a implementarlo. Esto suele hacerse habitualmente agrupando los objetos en clases y estas en árboles o enrejados que reflejan un comportamiento común. Cuando un objeto hereda de más de una clase se dice que hay herencia múltiple.

La POO constituye un paradigma que utiliza objetos como elementos fundamentales en la construcción de la solución. Un objeto es una abstracción de algún hecho o elemento del mundo real que tiene atributos que representan sus características o propiedades y métodos que representan su comportamiento o acciones que realizan. Todas las propiedades y métodos comunes a los objetos se encapsulan o se agrupan en clases. Una clase es una plantilla o un prototipo para crear objetos, por eso se dice que los objetos son instancias de clases. Algunos de los lenguaje de programación que se sirven de la POO son: C++, Java, C#, VB.Net, etc.

1.4.2 POA (Programación Orientada a Aspectos)

La POA es un nuevo paradigma de programación, creado por Gregor Kiczales. El principal objetivo que persigue es encapsular, capturar, aquellos conceptos que se escapan a los métodos actuales de descomposición de sistemas. Estos conceptos reciben el nombre de aspectos. [10]

La Programación Orientada a Aspectos (POA) es un paradigma de programación relativamente reciente, cuya intención es permitir una adecuada modularización de las aplicaciones y posibilitar una mejor separación de conceptos. Gracias a la POA se pueden encapsular los diferentes conceptos que componen una aplicación en entidades bien definidas, eliminando las dependencias entre cada uno de los módulos. De esta forma se consigue razonar mejor sobre los conceptos, se elimina la dispersión del código y las implementaciones resultan más comprensibles, adaptables y reusables. Varias tecnologías con nombres diferentes se encaminan a la consecución de los mismos objetivos y así, el término POA es usado para referirse a varias tecnologías relacionadas como los métodos adaptables, los filtros de composición, la programación orientada a sujetos o la separación multidimensional de competencias.

El principal objetivo de la POA es la separación de las funcionalidades dentro del sistema:

- Por un lado, funcionalidades comunes utilizadas a lo largo de la aplicación.
- Por otro, las funcionalidades propias de cada módulo.

Cada funcionalidad común se encapsulará en una entidad.

Muchas veces, a la hora de programar, aparecen problemas que no se pueden resolver de una manera adecuada con las técnicas habituales usadas en la programación imperativa o en la programación orientada a objetos. Con estas, nos vemos forzados a tomar decisiones de diseño que repercuten de manera importante en el desarrollo de la aplicación y que nos alejan con frecuencia de otras posibilidades.

A menudo, hace falta escribir líneas de código que están distribuidas por toda o gran parte de la aplicación, para definir la lógica de cierta propiedad o comportamiento del sistema, con las consecuentes dificultades de mantenimiento y desarrollo. Este problema se conoce como *scattered code*, que podríamos traducir como código disperso. Otro problema que puede aparecer, es que un mismo módulo implemente múltiples comportamientos o aspectos del sistema de forma simultánea. Esta nueva dificultad se conoce como *tangled code*, que podríamos traducir como código enmarañado. El hecho es que hay ciertas decisiones de diseño que son difíciles de capturar, debido a que determinados problemas no se pueden encapsular claramente de igual forma que los que habitualmente se resuelven con funciones u objetos.

Existen conceptos que no pueden encapsularse dentro de una unidad funcional, debido a que atraviesan todo el sistema o varias partes de él, como lo son la sincronización, el manejo de memoria, el manejo de errores, perfiles, seguridad o redes.

El desarrollo orientado a **aspectos** requiere de tres elementos básicos:

- Un lenguaje para definir la funcionalidad básica, conocido como lenguaje base o componente. Podría ser un lenguaje como C#, C++, Java o Lisp.
- Uno o varios lenguajes de aspectos, para especificar el comportamiento de los aspectos. Como podrían ser COOL para sincronización o RIDL para distribución.
- Un tejedor de aspectos (*aspect weaver*) que produce una aplicación que integra las funcionalidades de las clases y los aspectos. Tal proceso se puede llevar a cabo en tiempo de ejecución o en tiempo de compilación.

La POA define entonces una nueva forma de interacción, provista de los puntos de enlace (*join points*). Estos brindan la interfaz entre aspectos y componentes; son lugares dentro del código donde es posible agregar el comportamiento adicional especificado en los aspectos.

Un aspecto se compone de dos partes principales, el punto de corte (*pointcut*) y el consejo (*advice code*). Este último contiene el código a ser ejecutado, mientras que el punto de corte define un punto en el programa donde este código debe ser implementado. Como se percibe, las definiciones de punto de corte y punto de enlace están estrechamente relacionadas, no obstante, los conceptos son algo diferentes. Los puntos de enlace están bien definidos en tiempo de ejecución de las entidades, mientras que los puntos de corte se definen por un conjunto de puntos de enlace y no pueden atribuirse a una estructura particular o espacio de tiempo durante la ejecución del programa.

El consejo está asociado a un punto de corte a implementar. A diferencia de un método, el consejo nunca se llama directamente sino que está tejido dentro de los puntos de enlace definidos en el punto de corte en particular.

Los lenguajes de aspectos de propósito general son diseñados para describir cualquier clase de aspecto, no solo específicos, por lo que no pueden imponer restricciones al lenguaje base. El nivel de abstracción

del lenguaje base y del lenguaje de aspectos de propósito general es el mismo. No garantizan la separación de conceptos, esto es, que la unidad de aspecto se utilice únicamente para programar el aspecto, sin embargo, los de propósito general, proveen un ambiente más adecuado para el desarrollo de aspectos. Un ejemplo es *AspectJ*, donde *Java* es el lenguaje base, y las instrucciones de los aspectos también se escriben en *Java*.

1.4.3 Conclusiones sobre los paradigmas de programación

Como se ha puesto de manifiesto en este estudio, ambos paradigmas presentan características o ventajas que, indudablemente, no solo facilitarán el desarrollo sino que se hace necesario fusionarlos para poder realizar el trabajo.

Actualmente el paradigma de programación más usado es el de la Programación Orientada a Objetos. Sin embargo, un nuevo paso en la abstracción de paradigmas de programación es la Programación Orientada a Aspectos; aunque es todavía una metodología en estado de maduración, cada vez atrae a más investigadores e incluso proyectos comerciales en todo el mundo.

En una primera impresión, la POA y la POO pareciera que son en realidad el mismo paradigma, no obstante, esta noción es errónea. En la POO los sistemas se modelan como un conjunto de objetos que interactúan entre sí, sin embargo, falla al modelar los conceptos que se entrecruzan. La diferencia radica en que mientras la POA se enfoca en los conceptos que se entrecruzan, la POO se enfoca en los conceptos comunes. Por esta razón se evitarían muchas líneas de códigos si se trataran los conceptos comunes como se hará para el sistema, pero sí y solo si se emplearan también los conceptos que se entrecruzan, pues, para realizar las **transacciones** (configuración empleada para el comportamiento a prueba de errores de los métodos), no se podría programar de una manera óptima con solo utilizar la POO; sin embargo, al vincularla con la POA no se tendría que agregar código en cada uno de los métodos, ni siquiera hacer la más mínima modificación, pues la gran ventaja de esta inversión es que hace más sencillo el trabajo de los desarrolladores. Otras prerrogativas asociadas a la asunción de este método se relacionan a continuación:

- Permite una implementación modularizada, reduciendo el acoplamiento entre sus partes.

- El código es más limpio, menos duplicado, más fácil de entender y de mantener.
- Elimina los problemas causados por el código mezclado y el código diseminado.
- Mayor reutilización, los aspectos tienen mayores probabilidades de ser reutilizados en otros sistemas con requerimientos similares.
- Los sistemas son más adaptables a cambios, la separación de conceptos permite agregar nuevos aspectos, modificarlos o removerlos fácilmente.

Teniendo en cuenta tales argumentos es que se ha decidido entonces utilizar ambos paradigmas en el desarrollo de la capa lógica de negocio en el sistema y específicamente en los módulos de Contratación y Administración-Configuración.

1.5 Lenguaje de Programación

¿Qué es un lenguaje de programación?

Un lenguaje de programación es una construcción mental del ser humano para expresar programas. Está constituido por un grupo de reglas gramaticales, un grupo de símbolos utilizables, un grupo de términos monosémicos (es decir, con sentido único) y una regla principal que resume las demás.[11]

Son herramientas que nos permiten crear programas y software de una forma más flexible y portable. Estos facilitan la tarea de programación, ya que disponen de formas adecuadas que permiten ser leídas y escritas por personas, a su vez resultan independientes del modelo de computador a utilizar.[12]

Los lenguajes de programación se dividen en 3 grandes grupos principalmente. Estos son: [13]

- **Lenguajes Máquina:** es el lenguaje de programación que entiende directamente la computadora o máquina. Este lenguaje de programación utiliza el alfabeto binario, es decir, el 0 y el 1.

- **Lenguajes de programación de bajo nivel:** Son mucho más fáciles de utilizar que el lenguaje máquina, pero dependen mucho de la máquina o computadora como sucedía con el lenguaje máquina.

- **Lenguajes de programación de alto nivel:** Este tipo de lenguajes de programación son independientes de la máquina, lo podemos usar en cualquier computador con muy pocas modificaciones o sin ellas, son muy similares al lenguaje humano, pero precisan de un programa intérprete o compilador que traduzca este lenguaje de programación de alto nivel a uno de bajo nivel como el lenguaje de máquina que la computadora pueda entender.

Existen otras clasificaciones de los lenguajes de programación como son por nivel de abstracción, propósito, evolución histórica, manera de ejecutarse, paradigma de programación, concurrencia, interactividad, determinismo y productividad. [14]

En la investigación se hace un estudio de Java, un lenguaje de programación de alto nivel, ya que este fue el escogido para desarrollar el proyecto CCV.

1.5.1 Java

Java es un lenguaje de programación diseñado e implementado por Sun Microsystem en los años 90, pensado en un inicio para programar pequeños aparatos (electrodomésticos y artefactos electrónicos). [15]

Entre sus principales características tenemos que es: [16]

- **Orientado a Objetos:** Los objetos agrupan en estructuras encapsuladas tanto sus datos como los métodos (o funciones) que manipulan esos datos. La tendencia del futuro, a la que Java se suma, apunta hacia la programación orientada a objetos, especialmente en entornos cada vez más complejos y basados en red.

- **Distribuido**: Java proporciona una colección de clases para su uso en aplicaciones de red, que permiten abrir sockets y establecer y aceptar conexiones con servidores o clientes remotos, facilitando así la creación de aplicaciones distribuidas.

- **Interpretado y compilado a la vez:** Java es compilado, en la medida en que su código fuente se transforma en una especie de código máquina, los bytecodes, semejantes a las instrucciones de ensamblador.
  Por otra parte, es interpretado, ya que los bytecodes se pueden ejecutar directamente sobre cualquier máquina a la cual se hayan portado el intérprete y el sistema de ejecución en tiempo real (run-time).

- **Indiferente a la arquitectura:** diseñado para soportar aplicaciones que serán ejecutadas en los más variados entornos de red, desde Unix a Windows Nt, pasando por Mac y estaciones de trabajo, sobre arquitecturas distintas y con sistemas operativos diversos. Para acomodar requisitos de ejecución tan disímiles, el compilador de Java genera bytecodes: un formato intermedio indiferente a la arquitectura diseñada para transportar el código eficientemente a múltiples plataformas hardware y software. El resto de problemas los soluciona el intérprete de Java, llamado *Java Virtual Machine*. (Maquina Virtual de Java)

- **Portable:** La indiferencia a la arquitectura representa sólo una parte de su portabilidad. Además, Java especifica los tamaños de sus tipos de datos básicos y el comportamiento de sus operadores aritméticos, de manera que los programas son iguales en todas las plataformas.

- **Multihebra:** Hoy en día ya se ven como terriblemente limitadas las aplicaciones que solo pueden ejecutar una acción a la vez. Java soporta sincronización de múltiples hilos de ejecución (*multithreading*) a nivel de lenguaje, especialmente útiles en la creación de aplicaciones de red distribuidas. Así, mientras un hilo se encarga de la comunicación, otro puede interactuar con el usuario mientras otro presenta una animación en pantalla y otro realiza cálculos.

- **Dinámico:** El lenguaje Java y su sistema de ejecución en tiempo real son dinámicos en la fase de enlazado. Las clases solo se enlazan a medida que son necesitadas. Se pueden enlazar nuevos módulos de código bajo demanda, procedente de fuentes muy variadas, incluso desde la red.

- **Produce Applets:** puede ser usado para crear dos tipos de programas: aplicaciones independientes y applets. Las aplicaciones independientes se comportan como cualquier otro programa escrito en cualquier lenguaje, como por ejemplo el navegador de Web HotJava, escrito íntegramente en Java. Por su parte, las applets son pequeños programas que aparecen embebidos en las páginas Web, como aparecen los gráficos o el texto, pero con la capacidad de ejecutar acciones muy complejas, como animar imágenes, establecer conexiones de red, presentar menús y cuadros de diálogo para luego emprender acciones, etc.

Para poder desarrollar programas en este lenguaje, se necesita, al menos, un entorno de desarrollo mínimo que es el JDK.

JDK es el acrónimo de “Java Development Kit” (Kit de Desarrollo de Java), que se puede definir como un conjunto de herramientas, utilidades, documentación y ejemplos para desarrollar aplicaciones Java. Incluye componentes esenciales como la librería de clases de JAVA, el compilador y la maquina virtual llamada “java”. [15]

El lenguaje fue seleccionado para el desarrollo de la solución por los arquitectos, puesto que la versatilidad y eficiencia de la tecnología, la portabilidad de su plataforma y la seguridad que aporta, la han convertido en la tecnología ideal para su aplicación a redes. De portátiles a centros de datos, de consolas de juegos a súperequipos científicos, de teléfonos móviles a Internet; además ha sido probado, mejorado, ampliado y probado por una comunidad especializada de más de 6,5 millones de desarrolladores, la mayor y más activa del mundo. [17]

Otros aspectos tenidos en cuenta para su selección por los analistas fueron:

- Desarrollar software en una plataforma y ejecutarlo en prácticamente cualquier otra plataforma.
- Crear programas para que funcionen en un navegador web y en servicios web.

- Combinar aplicaciones o servicios que usan el lenguaje Java para crear servicios o aplicaciones totalmente personalizados.
- Desarrollar potentes y eficientes aplicaciones para teléfonos móviles, procesadores remotos, productos de consumo de bajo coste y prácticamente cualquier tipo de dispositivo digital.
- La mayoría de las tecnologías Java han sido liberadas por Sun Microsystems bajo la licencia GNU GPL (Licencia Pública General de GNU), de modo que forma parte del software libre.

1.6 Plataformas de Desarrollo

El desarrollo de SGP está muy vinculado a la colaboración entre entidades y departamentos, a esto se le conoce como programación distribuida, la cual ha tenido un importante auge en los últimos años. Ante este auge se han creado varias plataformas de desarrollo para implementarlas y entre las más reconocidas e innovadoras se encuentra Java 2 Plataform, Enterprise Edition (según sus siglas, J2EE), la cual se expone a continuación.

1.6.1 J2EE (Java 2 Plataform, Enterprise Edition)

J2EE es una especificación estándar basada en Java, que provee un conjunto de frameworks, librerías e interfaces que facilitan una infraestructura para desarrollar arquitecturas empresariales, además de brindar el entorno necesario para el desarrollo en n-capas distribuidas. Cuenta con una vasta experiencia y recibe soporte mundialmente.

Está basada en características de la plataforma J2SE (Java 2 Standard Edition) como el acceso a bases de datos, modelos de seguridad, portabilidad e interacción con recursos empresariales existentes. [18]

Además agrega las capacidades necesarias para proveer un sistema completo, estable, seguro y rápido para el nivel empresarial. [19]

Se aprecia que esta plataforma posee muchas características importantes como que:

- Está mayormente destinada a la programación orientada a objetos.
- Su propósito principal es facilitar el desarrollo de sistemas empresariales.
- Proporciona las herramientas necesarias para crear servicios web.
- Proporciona un modelo de acceso de componentes a datos y de lógica de negocio, separados por una capa intermedia de presentación implementada mediante *Servlets*.

También de esta plataforma se derivan algunas desventajas, como que no es muy sencilla de aprender y que su velocidad de desarrollo es más lenta que con otras plataformas como .NET de Microsoft. A pesar de estas limitaciones tiene sus ventajas, como el coste, pues la mayoría de las herramientas son gratuitas, además de operar bien y estar soportada en cualquier plataforma y tener comunidades formadas por muchos grupos independientes de desarrolladores.

1.7 IDEs (Integrated Development Enviroment)

¿Qué es un IDE?

Un IDE (Entorno de Desarrollo Integrado en español) es una aplicación que reúne varios programas necesarios para el desarrollador, está compuesto por varios componentes como: editor, compilador, intérprete, depurador, entre otros programas. Además se encarga de proveer un marco de trabajo amigable para la mayoría de los lenguajes de programación. Es posible que un IDE pueda funcionar con varios lenguajes de programación mediante la adición de plugins a este. Algunos ejemplos de IDEs son:

- Emacs.
- Anjuta.
- e-Des.
- Eclipse.
- Visual Studio .Net.

- NETBeans.
- KDevelop.
- C++Builder.

Al elegirse a Java como lenguaje de programación y a la plataforma J2EE para desarrollar la solución propuesta, se realiza un estudio sobre los IDEs que se emplean para desarrollar en esta tecnología, ya que las JDKs no ofrecen un ambiente de desarrollo para proyectos complejos, quizá para compilar una o dos clases serían de ayuda, mas si el proyecto contara con muchas clases ya este sería deficiente.

Los IDEs brindan un ambiente gráfico amigable y una gran cantidad de herramientas no ofrecidas en los JDKs, como son los debuggers más elaborados, check-points dentro de la compilación, creación de WAR's (Web-Archives), "Wizards" para acelerar el desarrollo, entre otras.

Entre los más conocidos para desarrollar en Java podemos encontrar a :

- Eclipse.
- NetBeans.
- JBuilder.
- WebSphere Studio.
- JDeveloper.
- Sun Java Studio.
- Microsoft Visual J++.
- IntelliJ IDEA.

De los mencionados anteriormente se analizan a continuación Eclipse y NetBeans por ser gratuitas, *open source* y su curva de aprendizaje bastante intuitiva además de ser los preferidos por las comunidades de desarrolladores en Java. [20]

1.7.1 Eclipse

Eclipse es una plataforma de desarrollo *open source* basada en Java. Es un desarrollo de IBM cuyo código fuente fue puesto a disposición de los usuarios. En sí mismo es un marco y un conjunto de servicios para construir un entorno de desarrollo a partir de componentes conectados (plug-in). [21]

Actualmente es desarrollado por la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios.

Eclipse presenta una arquitectura de plug-ins, la cual permite integrar diversos lenguajes e introducir otras aplicaciones accesorias. Además conserva el registro de las versiones, genera y mantiene la documentación de cada etapa del proyecto. [22]
Algunas de sus características son:

- Editor de texto.
- Compilación en tiempo real.
- Pruebas unitarias con JUnit.
- Asistentes (*wizards*) para creación de proyectos, clases, tests, etc.
- Refactorización.

Asimismo, a través de plug-ins libremente disponibles es posible añadir:

- Control de versiones con Subversion. [23]
- Integración con Hibernate. [24]

Además se puede instalar en varios sistemas operativos como:

- Microsoft Windows.
- GNU/Linux.
- MAC OS X.

1.7.2 NetBeans

NetBeans es un entorno de desarrollo fundado en el año 2000 por Sun Microsystems como un proyecto de código abierto; es una herramienta para que los programadores puedan escribir, compilar, depurar y ejecutar programas. Está escrito en Java pero puede servir para cualquier otro lenguaje de programación. Existe además un número importante de módulos para extender el IDE. Es un producto libre y gratuito sin restricciones de uso. [25]

Entre sus características principales se encuentran un sistema de proyectos basado en Ant, control de versiones y refactorización.

La última versión estable, NetBeans IDE 6.5 lanzada en noviembre del 2008, extiende las características existentes del J2EE. Adicionalmente, el NetBeans Enterprise Pack soporta el desarrollo de aplicaciones empresariales con Java EE 5, incluyendo herramientas de desarrollo visuales de SOA, herramientas de esquemas XML, orientación a web services, y modelado UML.

1.7.3 Conclusiones sobre los IDEs

Según lo analizado estos dos entornos de desarrollo tienen una gran competencia dentro de la comunidad de desarrolladores Java, ya que ambos poseen muchas facilidades e integraciones con otras aplicaciones, lo cual es de gran ayuda para el desarrollo, además de ser ambas de código abierto.

A pesar de esto se utiliza Eclipse porque al momento de concebir el proyecto ese IDE estaba más establecido en el mercado, se contaba con más documentación y era el más empleado en la Universidad.

Además de contar con las siguiente características:

- Soporta perfectamente la plataforma de desarrollo J2EE.
- Es fácilmente integrable con la herramienta CASE Visual Paradigm.
- Soporta el desarrollo de aplicaciones basadas en GUI y non-GUI.
- Soporta la construcción de una variedad de herramientas para el desarrollo de aplicaciones.

1.8 Frameworks utilizados

A grandes rasgos, un framework, en el proceso de desarrollo de software, es una estructura de soporte definida mediante la cual otro proyecto de software puede ser desarrollado y organizado; además los frameworks pueden incluir soporte de programas, bibliotecas y un lenguaje interpretado, entre otras aplicaciones para ayudar a desarrollar y unir los componentes de un proyecto.

En la solución se desarrolla la capa lógica de negocio con Spring Framework, y se hacen pruebas de validación con JUnit. A continuación se analizan las características principales de ambos:

1.8.1 Spring Framework

Spring es un framework de código abierto para el desarrollo de aplicaciones basadas en la tecnología Java. Fue escrito por Rod Jonhson, quien lo lanzó primero en su libro *Expert One-on-One Java EE Design and Development* en el año 2002. Existe además una versión para la plataforma .NET, llamada Spring.NET.

Este Framework se sustenta en dos características básicas en su núcleo: Inversión de Control (Inversion of Control, IoC) y la Programación Orientada a Aspectos. [26]

La Programación Orientada a Aspectos es usada en *Spring* para: [27]

- Proveer servicios de aplicación (enterprise services) declarativos. Ejemplo: declarar el manejo de transacciones.
- Permitir a los usuarios la facilidad de implementar sus propios aspectos personalizados.

Spring está diseñado con una serie de módulos que pueden trabajar sin dependencias entre ellos. Además intenta mantener un mínimo acoplamiento entre la aplicación y el mismo, de forma que podrían ser desvinculados sin demasiada dificultad.

Los principales módulos de este framework se muestran en la Figura 1.5 y son: [27]

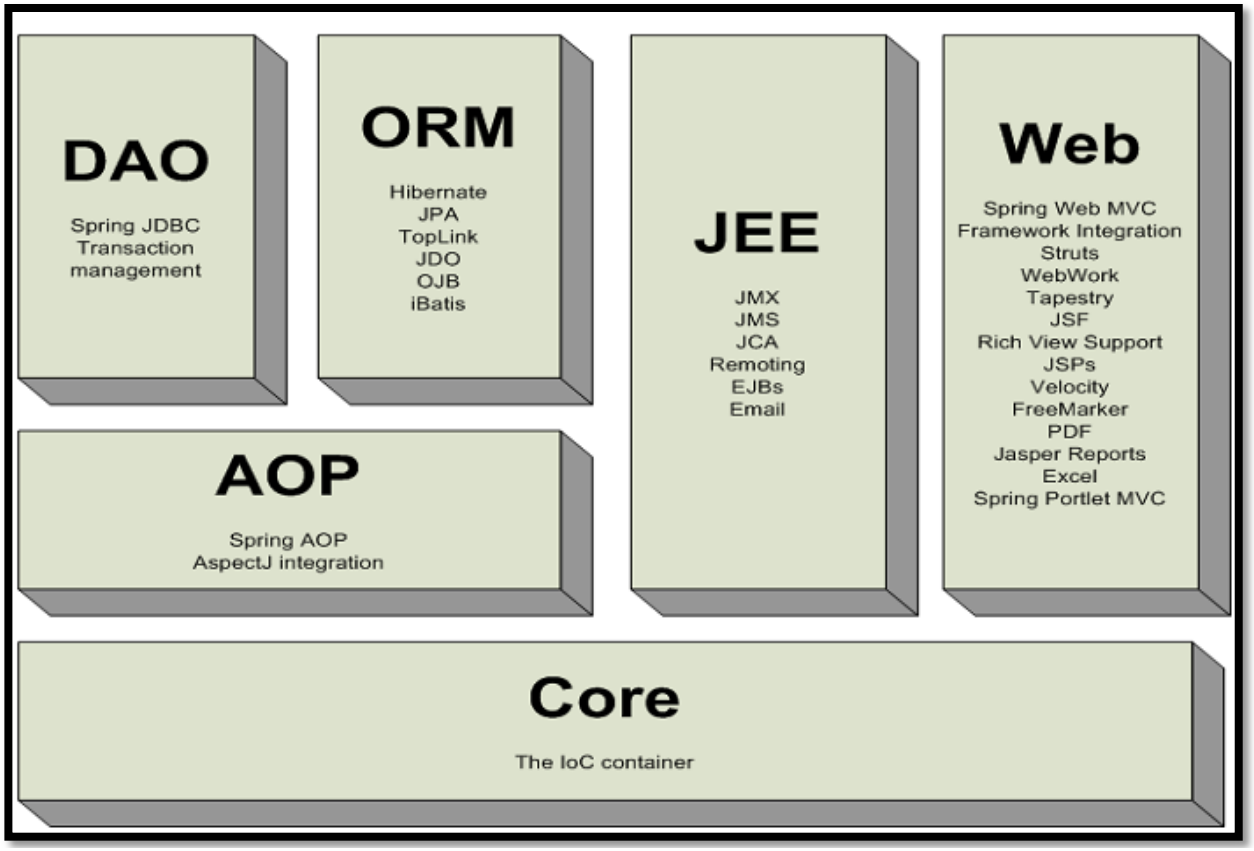


Figura 1.5: Módulos que conforman al Spring Framework.

- **Core:** Como su nombre indica, es el núcleo de Spring. Permite técnicas de Inversión del Control (IoC) como la inyección de dependencias.
- **JEE:** Proporciona herramientas para acceder a los beans y da soporte a propagación de eventos, resource bundles, carga de recursos y creación transparente de contextos por parte de los contenedores.
- **DAO:** Proporciona una capa de abstracción JDBC (Java Database Connectivity) y una forma de administrar transacciones.
- **ORM:** Provee capas de integración para APIs de mapeo objeto-relacional.
- **AOP:** Proporciona una implementación de programación orientada a aspectos, permitiendo definir puntos de corte e interceptores.

- **Web:** Provee de características de integración orientadas a la web, como funcionalidad multipartes, inicialización de contextos mediante servlet listeners y un contexto de aplicación orientada a la web. También permite integrar de forma sencilla otros frameworks como Struts, JSF o WebWork. Además provee una implementación Modelo-Vista-Controlador que permite el uso del resto de funcionalidades del Spring Framework.

1.8.2 JUnit

JUnit es un framework de código abierto creado por Erich Gamma y Kent Beck para hacer pruebas unitarias de aplicaciones Java. Este permite la ejecución de clases Java de manera controlada, para poder evaluar si el funcionamiento de cada uno de los métodos de la clase se comporta como se espera.

Entre sus principales ventajas se destacan que: [28]

- **JUnit es elegantemente simple.**
  Si escribir test es demasiado complejo o lleva demasiado tiempo, no existe ningún incentivo para empezar a escribir test en primer lugar. Con JUnit, se puede escribir rápidamente los tests que ejerciten el código e incrementalmente añadir test según va creciendo el software. Con JUnit, ejecutar tests es tan fácil como compilar el código. El compilador "testea" la sintaxis del código y los tests "validan" la integridad del código.

- **Los test JUnit chequean sus propios resultados y proporcionan feedback inmediato**.
  Testear no es divertido si se tiene que comparar manualmente los resultados esperados y obtenidos del test, y atrasa. Los tests JUnit se pueden ejecutar automáticamente y chequean sus propios resultados. Cuando se ejecutan los tests, se obtiene un feedback visual inmediato indicando si se ha pasado o fallado el test. No existe la necesidad de leer un informe para comparar los resultados.

- **Los tests JUnit pueden componerse como un árbol de suites de tests.**
  Los tests JUnit se pueden organizar en suites de tests que contienen ejemplares de tests e incluso

otras suites. El comportamiento compuesto de los tests JUnit te permite ensamblar colecciones de tests y automáticamente hacer un test regresivo de toda la suite de tests con un solo paso. También puede ejecutar los tests de cualquier capa dentro del árbol de suites.

- **Escribir tests JUnit no es costoso.**
  Utilizando el marco de trabajo JUnit, puedes escribir tests de forma barata y disfrutar de las conveniencias ofrecidas por el marco de trabajo. Escribir tests es tan simple como escribir un método que pruebe el código que se va a testear y definir el resultado esperado. El marco de trabajo proporciona el contexto para ejecutar los tests automáticamente y como parte de una colección de otros tests. Esta pequeña inversión en testear continuará beneficiando en tiempo y calidad.

- **Los tests JUnit incrementan la estabilidad del software**.
  Cuanto menos tests se escriban, menos estable estará el código. Los tests validan la estabilidad del software y dan la confianza de que los cambios no causarán efectos negativos en el software. Los tests forman el pegamento de la integridad estructural del software.

- **Los tests JUnit son tests del desarrollador.**
  Los tests JUnit son tests altamente localizados escritos para mejorar la productividad del desarrollador y la calidad del código. Al contrario de los tests funcionales, que tratan al sistema como una caja negra y aseguran que el software funciona como una totalidad. Las unidades de tests están escritas para probar los bloques de construcción básicos del sistema desde dentro. Los desarrolladores escriben y poseen los tests JUnit. Cuando se completa una iteración de desarrollo, los tests se convierten en parte y parcela del producto entregado como una forma de comunicación.

- **Los tests JUnit se escriben en Java.**
  Testear software Java utilizando tests Java forma una similitud entre el test y el código testeado. El test se convierte en una extensión del software general y el código se puede reconstruir partiendo de los tests. El compilador Java ayuda al proceso de testeo realizando el chequeo de la sintaxis

estática de las unidades de testeo y asegurándose de que el contrato de interface del software se está obedeciendo.

Actualmente los IDEs como NetBeans y Eclipse cuentan con plug-ins que permiten que la generación de las plantillas necesarias para la creación de las pruebas de una clase Java se realice automáticamente, facilitando al programador enfocarse en la prueba y el resultado esperado, y dejando a la herramienta la creación de las clases que permiten coordinar las pruebas.

1.9 Estándares de Codificación

Un estándar de codificación completo comprende todos los aspectos de la generación de código. Si bien los programadores deben implementar un estándar de forma prudente, este debe tender siempre a lo práctico [29]. Un código fuente completo debe reflejar un estilo armonioso, como si un único programador hubiera escrito todo el código de una sola vez. Al comenzar un proyecto de software, es necesario establecer un estándar de codificación para asegurarse de que todos los programadores del proyecto trabajen de forma coordinada. Cuando el proyecto de software incorpore código fuente previo, o bien cuando realice el mantenimiento de un sistema de software creado anteriormente, el estándar de codificación deberá establecer cómo operar con la base de código existente.

La legibilidad del código fuente repercute directamente en lo bien que un programador comprende un sistema de software. La mantenibilidad del código es la facilidad con que el sistema de software puede modificarse para añadirle nuevas características, modificar las ya existentes, depurar errores, o mejorar el rendimiento. Aunque la legibilidad y la mantenibilidad son el resultado de muchos factores, una faceta del desarrollo de software en la que todos los programadores influyen especialmente es en la técnica de codificación. El mejor método para asegurarse de que un equipo de programadores mantenga un código de calidad es establecer un estándar de codificación sobre el que se efectuarán luego revisiones del código de rutinas.

Usar técnicas de codificación sólidas y realizar buenas prácticas de programación con vistas a generar un código de alta calidad es de gran importancia para la calidad del software y para obtener un buen

rendimiento. Además, si se aplica de forma continuada un estándar de codificación bien definido, se utilizan técnicas de programación apropiadas y, posteriormente, se efectúan revisiones del código de rutinas, caben muchas posibilidades de que un proyecto de software se convierta en un sistema de software fácil de comprender y de mantener.

Aunque el propósito principal para llevar a cabo revisiones del código a lo largo de todo el desarrollo es localizar defectos en el mismo, las revisiones también pueden afianzar los estándares de codificación de manera uniforme. La adopción de un estándar de codificación sólo es viable si se sigue desde el principio hasta el final del proyecto de software. No es práctico, ni prudente, imponer un estándar de codificación una vez iniciado el trabajo.

Las técnicas de codificación incorporan muchos aspectos del desarrollo del software. Aunque generalmente no afectan la funcionalidad de la aplicación, sí contribuyen a una mejor compresión del código fuente. En esta fase se tienen en cuenta todos los tipos de código fuente, incluidos los lenguajes de programación, de marcado o de consulta.

En general, una técnica de codificación no pretende formar un conjunto inflexible de estándares de codificación. Más bien intenta servir de guía en el desarrollo de un estándar de codificación para un proyecto específico de software.

1.10 Patrones de Diseño

Los patrones de diseño son soluciones estándares para un problema común de programación, son técnicas para flexibilizar el código haciéndolo satisfacer ciertos criterios y son además una manera más práctica de describir ciertos aspectos de la organización de un programa.

Hay patrones que abarcan las distintas etapas del desarrollo, desde el análisis hasta el diseño y desde la arquitectura hasta la implementación.

En 1994, aparece el libro "Design Patterns: Elements of Reusable Object Oriented Sofware" escrito por los famosos Gang of Four (GoF, en español, la pandilla de los cuatro) formada por Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides. Ellos recopilaron y documentaron veintitrés patrones de diseño aplicados usualmente por expertos diseñadores de software orientado a objetos. [30]

El grupo de GoF clasificaron los patrones en tres grandes categorías (Creacionales, Estructurales, Comportamiento) basadas en su propósito y en un segundo nivel clasificaron los patrones en 2 ámbitos: Clases y Objetos, obteniendo seis clasificaciones: [30]

- **Creacionales**: tratan con las formas de crear instancias de objetos. El objetivo de estos patrones es de abstraer el proceso de instanciación y ocultar los detalles de cómo los objetos son creados o inicializados.

    - **Creacional de la Clase:** usan la herencia como un mecanismo para lograr la instanciación de la clase. Por ejemplo, el método Factoría.

    - **Creacional del objeto:** son más escalables y dinámicos comparados con los patrones creacionales de Clases. Por ejemplo, la Factoría abstracta y el patrón Singleton.

- **Estructurales**: describen cómo las clases y objetos pueden ser combinados para formar grandes estructuras y proporcionar nuevas funcionalidades. Estos objetos adicionados pueden ser incluso objetos simples u objetos compuestos.

    - **Estructural de la Clase:** usan la herencia para proporcionar interfaces más útiles combinando la funcionalidad de múltiples clases. Por ejemplo, el patrón Adaptador (Clase).

    - **Estructural de Objetos:** crean objetos complejos agregando objetos individuales para construir grandes estructuras. La composición de este puede ser cambiada en tiempo de ejecución. El patrón da flexibilidad adicional sobre los patrones estructurales de clases. Por ejemplo, el Adaptador (Objeto), Facade, Bridge, Composite.

- **Comportamiento**: ayudan a definir la comunicación e iteración entre los objetos de un sistema. El propósito de este patrón es reducir el acoplamiento entre los objetos.

    - **Comportamiento de Clase:** usan la herencia para distribuir el comportamiento entre Clases. Por ejemplo, Interpreter.

    - **Comportamiento de Objeto:** nos permite analizar los patrones de comunicación entre objetos interconectados, como objetos incluidos en un objeto complejo. Ejemplo, Iterator, Observer, Visitor.

1.10.1 Patrones de Diseño para J2EE

Además de los patrones de diseño vistos anteriormente, existen otros patrones más específicos enfocados para la plataforma J2EE.

Según el libro "J2EE PATTERNS Best Practices and Design Strategies", existen cinco capas en la arquitectura J2EE:

- Cliente.
- Presentación.
- Negocios.
- Integración.
- Recurso.

El libro explica quince patrones J2EE que están divididos en tres de las capas: presentación, negocios e integración.

De estos se exponen los relacionados con la capa lógica de negocio: [30]

- **Business Delegate:** Un objeto que reside en la capa de presentación y en beneficio de los otros componentes de la capa de presentación llama a métodos remotos en los objetos de la capa de negocios.

- **Value Object/ Data Transfer Object/ Replicate Object:** Un objeto serializable para la transferencia de datos sobre la red.

- **Session Facade/ Session Entity Facade/ Distributed Facade:** El uso de un bean de sesion como una fachada (facade) para encapsular la complejidad de las interacciones entre los objetos de negocio y participantes en un flujo de trabajo. El Session Facade maneja los objetos de negocio y proporciona un servicio de acceso uniforme a los clientes.

- **Aggregate Entity**: Un bean entidad que es construido o es agregado a otros beans de entidad.

- **Value Object Assembler:** Un objeto que reside en la capa de negocios y crea Value Objets cuando es requerido.

- **Value List Handler/ Page-by-Page Iterator/ Paged List:** Es un objeto que maneja la ejecución de consultas SQL, caché y procesamiento del resultado. Usualmente implementado como beans de sesión.

- **Service Locator:** Consiste en utilizar un objeto Service Locator para abstraer toda la utilización JNDI y para ocultar las complejidades de la creación del contexto inicial, de búsqueda de objetos home EJB y recreación de objetos EJB. Varios clientes pueden reutilizar el objeto Service Locator para reducir la complejidad del código, proporcionando un punto de control.

1.11 Conclusiones del capítulo

En el presente capítulo se realizó un estudio sobre los SGP, las metodologías de desarrollo de software, paradigmas y lenguajes de programación, plataformas de desarrollo, IDEs y *frameworks* para desarrollar capas lógicas de negocio; por lo cual se decide desarrollar el Proyecto CCV siguiendo la metodología RUP, utilizando la fusión de la POO y la POA como paradigmas, con la plataforma J2EE para el lenguaje de programación Java, el Framework Spring, para implementar la capa lógica de negocio, JUnit para las pruebas y el IDE Eclipse en el desarrollo.

Capítulo 2. Descripción de la solución propuesta

2.1 Introducción

A lo largo de este capítulo se mostrará descriptivamente la solución propuesta para lograr el objetivo planteado en el diseño teórico de la investigación; se presentan las características principales de la capa lógica de negocio, el estándar de codificación empleado, los patrones de diseños enfocados a J2EE que fueron utilizados, las configuraciones de los ficheros XML, el artefacto Modelo de Implementación generado durante el flujo de implementación para cada uno de los módulos, así como la implementación de las clases del diseño y los algoritmos y estructuras de datos empleados.

2.2 Capa Lógica de Negocio

Para el desarrollo del proyecto se utilizó una arquitectura separada por varias capas lógicas con Spring como framework arquitectónico base, sirviendo de guía su contenedor de objetos y su técnica de inyección de instancias para enlazar la capa de presentación y la capa de acceso a datos con la capa lógica de negocio, logrando el menor acoplamiento posible y haciendo uso de su módulo orientado a aspectos para manejar las transacciones y la seguridad. Esta arquitectura se ilustra en la Figura 2.1.

En la capa lógica de negocio radican los objetos de negocio o Business Objects. Los objetos de negocio separan los datos y la lógica de negocio usando un modelo de objetos.

Esta capa presenta las siguientes funcionalidades:

- **Lógica de negocio específica de procesos de negocios:** En ocasiones es más oportuno para los objetos de dominio contener lógica de negocio aplicable a muchos casos de uso específicos. En esta definición de arquitectura los objetos de dominio no presentan ningún tipo de lógica de negocio, sino que esta responsabilidad recae sobre los objetos de negocio, permitiendo usar a los objetos de dominio como objetos de transferencia que se mueven entre las capas arquitectónicas de la aplicación.

- **Puntos de entrada muy bien definidos para las operaciones de negocio implementadas:** Los objetos de negocio brindan las interfaces usadas por la capa de presentación.
- **Control de transacciones:** Las políticas de transaccionales de la aplicación son planteadas sobre los objetos de negocio.
- **Ejecución de restricciones de seguridad:** Las restricciones de seguridad en esta capa se realiza en los puntos de entradas a la capa media, es decir en los objetos de negocio.
- **Ejecución de la auditoría:** Sobre esta capa se lleva a cabo la auditoría sobre los métodos de negocio.

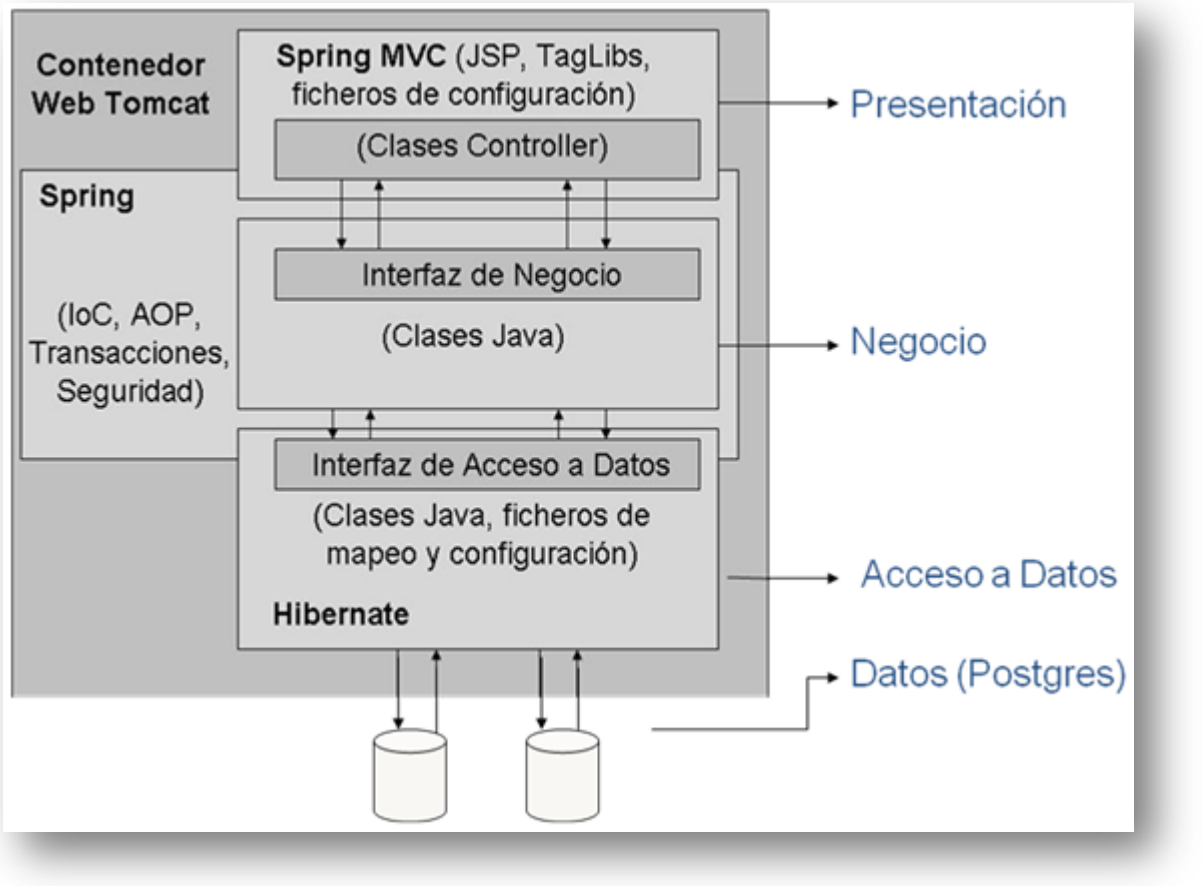


Figura 2.1: Arquitectura del proyecto CCV.

Por cada módulo se define una o más fachadas, en caso de que se requiera, que agrupen los métodos de negocio implementados en los manejadores o Managers. La fachada de un módulo se basa en el patrón Facade para permitir una clara división entre las capas arquitectónicas. Los Managers son las clases que se especializan en un conjunto de funcionalidades que representan el negocio sobre una o varias entidades. Los Managers son las únicas clases en la aplicación que tienen lógica de negocio mientras que las fachadas se limitan solamente a agrupar las funcionalidades para ser expuestas a capas superiores.

En la Figura 2.2 se muestra cómo se encuentran estructurados los paquetes definidos previamente por el arquitecto para la implementación de los módulos Contratación y Administración-Configuración:

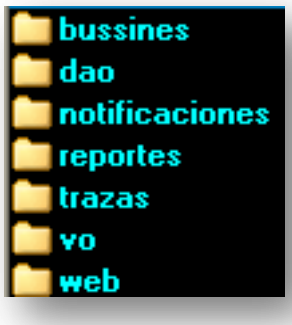


Figura 2.2: Estructura de los paquetes de cada módulo para la implementación.

De estos paquetes el que concierne a la investigación es el paquete **bussines**, el cual corresponde con la capa lógica de negocio de cada módulo. En este se encuentran un conjunto de interfaces llamadas **services** y otro paquete **impl** con las clases managers que implementan las interfaces, como se muestra en la Figura 2.3:

Figura 2.3: Paquetes **bussines** de los módulos Contratación y Administración-Configuración.

La mayor parte de las interfaces contenidas en el paquete **bussines** heredan principalmente de dos interfaces, ProcesoService y DocumentoService, contenidas dentro del paquete **bussines** del módulo **common,** el cual se encarga de manejar un conjunto de clases comunes y genéricas de manera que los módulos restantes puedan heredar de ellas para hacer uso de sus facilidades.

Estas dos interfaces son mostradas en las Figuras 2.4 y 2.5, respectivamente:

Figura 2.4: Interface ProcesoService.

Figura 2.5: Interface DocumentoService.

2.3 Estándar de Codificación

Como se había abordado, un estándar de codificación comprende todos los aspectos de la generación de código. Anteriormente se exponía que los programadores deben implementar un estándar tendiendo siempre a lo práctico. Los estándares de codificaciones resultan esenciales para los programadores, pues en la mayoría de las ocasiones quienes fabrican el software no son los que se encargarán de hacer que funcione posteriormente; mejoran la lectura del software al permitir la interpretación del código de manera más rápida y profunda [29]. Un código fuente completo debe resultar un entorno familiar para cada desarrollador, como si hubiese sido escrito por él mismo. El mejor método para asegurarse de que un equipo de programadores mantenga un código de calidad es establecer un estándar de codificación sobre el que se efectuarán luego revisiones del código de rutinas y su mantenimiento.

En el caso específico de este proyecto se decidió desarrollar la capa lógica de negocio de los módulos de Contratación y Administración-Configuración siguiendo el estándar de codificación definido para el lenguaje de programación de Java:

- Nombres de ficheros.
- Organización de los ficheros.
- Indentación.
- Comentarios.
- Declaraciones.
- Sentencias.
- Espacios en blanco.

La aplicación mantiene un lenguaje común y uniforme, pues se especifican convenciones de nombres y estándares de código para los distintos recursos. Las clases Java adoptan convenciones estándares presentadas en la Especificación del Lenguaje Java por la compañía Sun Microsystem.

No obstante, se tomaron una serie de acuerdos a raíz de nuevas proposiciones de reglas a cumplir efectuadas por los desarrolladores con el fin de establecer un estándar de codificación propio para CCV. Seguidamente se brindan varios aspectos que resultaron muy convenientes y se tuvieron en cuenta a la hora de enriquecer el estándar de codificación que se ha utilizado:

- Asegurar que el tamaño de una variable no sea excesivamente grande (para conservar los recursos es necesario ser selectivo en el tipo de dato).
- Mantener el tiempo de vida de las variables tan corto como sea posible.
- Mantener el *scope* de las variables tan corto como sea posible, lo cual evita confusiones y mejora la mantenibilidad, además de minimizar la dependencia, es decir, si por algún motivo se comete el error de borrar una variable es más fácil de encontrar el error si esta tiene un *scope* más pequeño.
- Utilizar los procedimientos y variables con un solo propósito, evitando así crear procedimientos multipropósito que lleven a cabo una variedad de funciones no relacionadas.

- Dentro de una clase, evitar el uso de variables públicas, utilizar en cambio procedimientos y propiedades que accedan a dichas variables (privadas), así se provee una capa de encapsulación y se brinda la posibilidad de validar valores de cambio sobre las mismas, antes de manipularlas directamente.
- Minimizar el *scope* y la duración de las transacciones.
- Evitar el uso de conversión de tipos o *casting*, ya que esto puede generar resultados imprevistos, sobre todo cuando dos variables están involucradas en una sentencia, el *cast* se debe utilizar solo en situaciones triviales.
- Usar siempre rutinas de manejo de excepciones.
- Especificar cuando se declaren objetos que puedan generar colisión, por ejemplo, si se tienen dos métodos con el mismo nombre en diferentes paquetes se deben escribir con el nombre completo incluyendo el del paquete.
- Usar siempre sentencias *Case* o *Switch* en lugar de utilizar sentencias *if-then* repetitivas.
- Liberar explícitamente las referencias a objeto. (variable = null)
- Siempre que sea posible se debe utilizar polimorfismo en vez de cláusulas *Switch*.

Las técnicas de codificación que se estuvieron utilizando están divididas en tres secciones:

- Nombrado.
- Documentación Interna. (Comentarios)
- Formato.

Nombrado

El esquema de nombres es una de las ayudas más importantes para entender el flujo lógico de la aplicación. Los nombres expresan más bien el "qué" que el "cómo", por lo tanto se utilizan nombres que evitan referirse a la implementación para de esta manera conservar la abstracción de la estructura, ya que la implementación está sujeta a cambios, así se describe qué hace la estructura y no cómo lo hace.

Otra directiva que se utiliza es la de que los nombres sean tan largos como para que se entiendan pero a la vez tan cortos como para que no den información irrelevante.

- Estructuras (paquetes, procedimientos, clases, interfaces y propiedades):

  - Los nombres de todas las estructuras de código deben ser en español.
  - Los paquetes deben empezar por el país, la institución donde se fabrica el software, el nombre del proyecto seguido del nombre del módulo y la capa correspondiente:

    cu.uci.ccv.contratacion.business

    [país].[institución].[proyecto].[modalidad].[capa]
  - El nombre de la clase y el archivo fuente deben ser iguales.
  - Evitar nombres imprecisos que permitan interpretaciones subjetivas. Tales nombres contribuyen más a la ambigüedad que a la abstracción.
  - En la POO es redundante incluir nombres de clases en el nombre de las propiedades de clases.
  - Utilizar la técnica verbo-sustantivo para nombrar procedimientos que ejecuten alguna operación en un determinado objeto, por ejemplo aceptarEnvio().
  - Terminar el nombre de las clases interfaces del Negocio con *Service* al final, pues cada una de ellas constituyen un servicio, por ejemplo: BusquedaService.
  - Terminar el nombre de las clases implementadoras con el sufijo Impl, por ejemplo: BusquedaServiceImpl.
  - No utilizar el guión bajo "_" (con la excepción de las variables privadas), y lo menos posible las abreviaturas, pues causan confusiones.

- Variables:

  - Es recomendado que las variable booleanas contengan una palabra que describa su estado: puedeEliminarse(), existe(), etc. Y siempre se debe referir al estado verdadero.
  - Incluso para el caso de una variable de poco uso, que deba aparecer sólo en unas cuantas líneas de código, emplear un nombre descriptivo. Utilizar nombres de variables de una sola letra, como i ó j sólo para índices (ciclos *for*).

- Parámetros:
  - Los parámetros siguen el mismo estándar de las variables.

- Varios
  - Para términos largos o utilizados con frecuencia, se deben emplear abreviaturas que mantengan las longitudes de los nombres dentro un límite razonable. En general, los nombres de variables con más de treinta y dos caracteres son difíciles de leer. Además, las abreviaturas deben ser coherentes a lo largo de toda la aplicación.
  - Minimizar el uso de abreviaturas, pero si se emplean, deben utilizarse coherentemente. Una abreviatura sólo debe tener un significado y, del mismo modo, a cada palabra abreviada sólo debe corresponder una abreviatura.
  - Los archivos y los nombres de carpetas, al igual que los nombres de los métodos, deben describir claramente su finalidad.
  - No reutilizar nombres para elementos diferentes.
  - Evitar el uso de caracteres como “$” o “%”.
  - No usar nombres que sean dependientes del tipo de variable, control o clase, en cambio, se debe utilizar, como ya se dijo, nombres que describan el propósito de la variable, control o propiedad.

Comentarios

En CCV existen dos tipos de documentación: externa e interna. La documentación externa, por ejemplo: las especificaciones, los archivos de ayuda y los documentos de diseño, se ha mantenido fuera del código fuente. La documentación interna es a la que se le llama comentarios, y que los programadores han escrito dentro del código fuente durante la fase de desarrollo.

Uno de los problemas de los comentarios es garantizar que se mantengan y actualicen al mismo tiempo que el código fuente. Aunque unos buenos comentarios en el código fuente no tienen ningún valor en el tiempo de ejecución, resultan valiosísimos para un programador que tenga que mantener una parte de software particularmente compleja.

A la hora de desarrollar CCV se siguieron las siguientes pautas en sus comentarios:

- Son en español.
- Se realizan al principio de algunas rutinas importantes y complejas que indican el propósito de la misma, las suposiciones y las limitaciones, lo que consiste en una breve introducción que explica por qué existe y qué puede hacer.
- Se evitan los comentarios recargados, como las líneas enteras de asteriscos. En su lugar, se utilizan espacios para separar los comentarios y el código.
- Se usan frases completas cuando se escriben y deben aclarar el código, no añadirle ambigüedad.
- Se usan para explicar el propósito del código, no como si fueran traducciones literales.
- Deben estar ortográficamente bien escritos.
- Se evitan los comentarios que expliquen cosas obvias, en la mayoría de las cosas el código debe ser autoexplicativo.

Formato

El formato de código en CCV hace que la organización lógica del código sea más clara, pues para establecer el formato se tomaron los siguientes lineamientos:

- Se evitó albergar múltiples clases en un solo archivo.
- Se estableció un tamaño estándar de sangría y se alinearon las secciones de código mediante la sangría predeterminada.
- Se declaró una sola variable por línea.
- Se aislaron las interfaces de la implementación.
- Se inicializaron las variables en el momento de la declaración.

Adicionales

Las excepciones se trataron también siguiendo algunas premisas:

- No se declaró ninguna sentencia *catch* vacía.

- Se evitó anidar los bloques *try/catch*.
- Se ordenó la captura de excepciones *catch* siempre en orden descendente desde la más particular hasta la más genérica.

2.4 Configuraciones e Integración

Las configuraciones XML de los beans de la capa lógica de negocio se realizaron con el esquema propuesto por el Framework Spring como se muestra en la Figura 2.7:

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xmlns:lang="http://www.springframework.org/schema/lang"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xmlns:util="http://www.springframework.org/schema/util"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
        http://www.springframework.org/schema/aop
        http://www.springframework.org/schema/aop/spring-aop-2.0.xsd
        http://www.springframework.org/schema/lang
        http://www.springframework.org/schema/lang/spring-lang-2.0.xsd
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx-2.0.xsd
        http://www.springframework.org/schema/util
        http://www.springframework.org/schema/util/spring-util-2.0.xsd">
```

Figura 2.7: Esquema para los XML propuesto por el Framework Spring.

Las configuraciones e integraciones de las diferentes capas de los módulos se encuentran ubicadas en el paquete **conf** dentro de la carpeta **WEB-INF** del **WebContent;** y se les localiza mediante el **web.xml** como se muestra a continuación:

```
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>
        /WEB-INF/conf/administracion/administracion-applicationContext-dao.xml
        /WEB-INF/conf/administracion/administracion-applicationContext-notificaciones.xml
        /WEB-INF/conf/administracion/administracion-applicationContext-bussines.xml
        /WEB-INF/conf/administracion/administracion-applicationContext-remoteServer.xml


        /WEB-INF/conf/contratacion/contratacion-applicationContext-dao.xml
        /WEB-INF/conf/contratacion/contratacion-applicationContext-bussines.xml
        /WEB-INF/conf/contratacion/contratacion-applicationContext-reporte.xml
        /WEB-INF/conf/contratacion/contratacion-applicationContext-remoteServer.xml
    </param-value>
</context-param>
```

Figura 2.8: Configuración de la ubicación de los XML de cada módulo dentro del **web.xml**.

Integración Capa Lógica de Negocio/Capa Acceso a Datos

La integración de las capas lógica de negocio y acceso a datos se implementa mediante configuraciones XML, en las cuales, mediante la inyección de dependencia brindada por el framework Spring, cada **bean** de las clases **Service** referencia a través de una **property** a un **bean** que se encuentra en la configuración de XML de la capa de acceso a datos, siendo necesario tener en las clases del negocio un atributo con el mismo nombre del cual se hace referencia en el **bean,** conjuntamente con su método Set() para poder llevar a cabo la inyección de dependencias.

Las Figuras 2.9 y 2.10 muestran una parte del XML principal que integra la capa lógica de negocio con la capa de acceso a datos y evidencia lo explicado anteriormente:

```
<bean id="BusquedaService"
    class="cu.uci.ccv.contratacion.bussines.impl.BusquedaServiceImpl">
    <property name="contratoDao" ref="contratoDao" />
    <property name="cronogramaDao" ref="cronogramaDao" />
    <property name="desembolsoInvDao" ref="desembolsoInvDao" />
    <property name="desembolsoTransfDao" ref="desembolsoTransfDao" />
    <property name="rechazoDao" ref="rechazoDao"></property>
</bean>

<bean id="ContractService"
    class="cu.uci.ccv.contratacion.bussines.impl.ContractServiceImpl"
    parent="DocumentoService">
    <property name="contratoDao" ref="contratoDao" />
</bean>

<bean id="CronogramEFService"
    class="cu.uci.ccv.contratacion.bussines.impl.CronogramServiceImpl"
    parent="DocumentoService">
    <property name="cronogramaDao" ref="cronogramaDao" />
</bean>
```

Figura 2.9: Integración de las capas lógica de negocio y acceso a datos. (Módulo Contratación)

```
<bean id="tipoRecursoService"
    class="cu.uci.ccv.administracion.bussines.impl.TipoRecursoServiceImpl">
    <property name="tipoRecursoDao">
        <ref bean="tipoDeRecursoDao"/>
    </property>
</bean>


<bean id="viaticoService"
    class="cu.uci.ccv.administracion.bussines.impl.ViaticoServiceImpl">
    <property name="viaticoDao">
        <ref bean="viaticoDao"/>
    </property>
</bean>
```

Figura 2.10: Integración de las capas lógica de negocio y acceso a datos. (Módulo Administración-Configuración)

Además los **beans** creados en la capa lógica de negocio son referenciados por la capa de presentación de la misma forma.

Esta capa cuenta con otras configuraciones como la encargada de integrar la capa con las distintas funcionalidades que ofrece el paradigma de Programación Orientada a Aspectos (POA) también utilizado para el desarrollo del sistema. En la Figura 2.11 se muestran partes de esta configuración:

```
<bean id="transactionManager"
    class="org.springframework.orm.hibernate3.HibernateTransactionManager">
    <property name="sessionFactory">
        <ref bean="sessionFactory" />
    </property>
    <property name="nestedTransactionAllowed" value="true"/>
</bean>

<tx:advice id="txAdvice" transaction-manager="transactionManager">
    <tx:attributes>
        <tx:method name="crear*" propagation="REQUIRED" />
        <tx:method name="modificar*" propagation="REQUIRED" />
        <tx:method name="comprobar*" propagation="REQUIRED" />
        <tx:method name="eliminar*" propagation="REQUIRED" />
        <tx:method name="aceptar*" propagation="REQUIRED" />
        <tx:method name="rechazar*" propagation="NOT_SUPPORTED" />
    </tx:attributes>
</tx:advice>

<aop:config>
    <aop:advisor pointcut="execution(* cu.uci.ccv.administracion.bussines.*.*(..))"
    advice-ref="txAdvice"/>
    <aop:advisor pointcut="execution(* cu.uci.ccv.contratacion.bussines.*.*(..))"
    advice-ref="txAdvice"/>
    <aop:advisor pointcut="execution(* cu.uci.ccv.contratacion.dao.*.*(..))"
    advice-ref="txAdvice"/>
</aop:config>
```

Figura 2.11: Configuración de transacciones y AOP.

2.5 Diseño de Clases

Un diagrama de clases del diseño modela las clases que contendrá el sistema y sus relaciones. Además se declaran sus atributos y métodos dependiendo siempre del lenguaje de programación con el cual se implementará el sistema. Precisamente el punto de entrada para una implementación eficiente y eficaz en la capa lógica de negocio, es tener el diseño de clases previamente realizado por los diseñadores y arquitectos, este suele ser un artefacto generado después de todo un ciclo de desarrollo por donde pasa el software, comenzando desde la fase de inicio con el modelamiento del negocio, después por los requerimientos y por último la descripción del análisis. Para el desarrollo de estos diagramas se respetan los patrones de diseño, previamente utilizados por el diseñador con la ayuda de los desarrolladores, como son:

- Bajo acoplamiento, el cual se encarga de asignar las responsabilidades de modo que se mantenga un bajo acople, o sea, el modo de dar soporte a poca dependencia y a una mayor reutilización.

- Facade, que consiste en proveer una interfaz única a un conjunto de interfaces en un subsistema, o sea, define una interface a alto nivel que hace más fácil el uso del subsistema.

- Experto, el cual asigna una determinada responsabilidad a la clase que cuenta con la información necesaria para cumplir con la responsabilidad.

- Alta cohesión, que se encarga de asignar las responsabilidades de modo que la cohesión siga siendo alta, o sea, que las clases se caractericen por estar estrechamente relacionadas y que no realicen un trabajo enorme.

El diseño de clases además proporciona a los programadores un mejor entendimiento de los problemas a solucionar, además de brindar una visión más clara de todos los casos de uso que se implementarán.

En la presente investigación se muestran los diagramas de clases del diseño, específicamente de la capa lógica de negocio, no siendo estos los del sistema en general. Se ilustran estos diagramas para los módulos Contratación y Administración-Configuración, respectivamente en las Figuras 2.12 y 2.13

2.5.2 Módulo Contratación

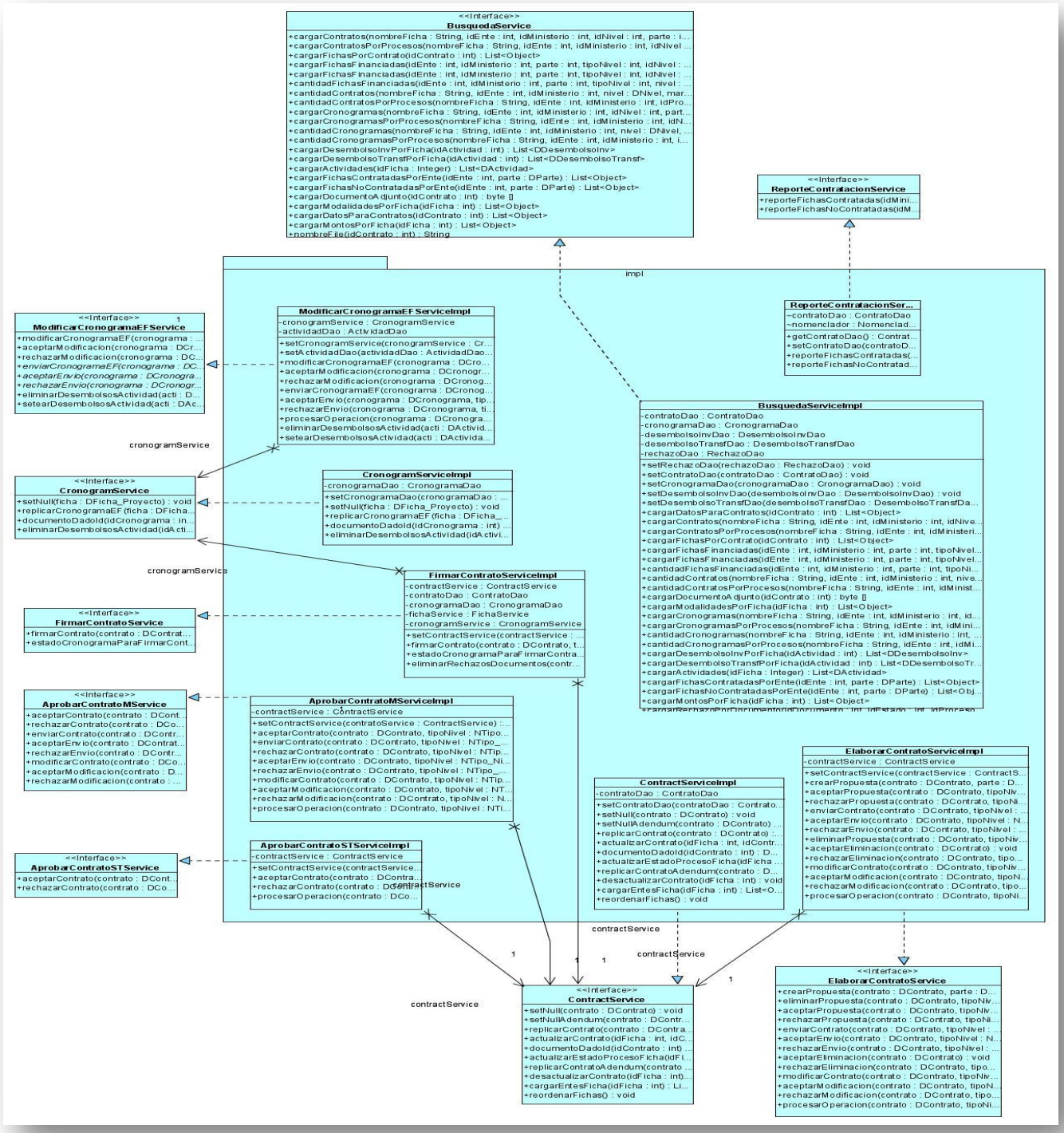


Figura 2.12: Diagrama de clases del diseño para la capa lógica de negocio, del módulo Contratación.

2.5.3 Módulo Administración-Configuración

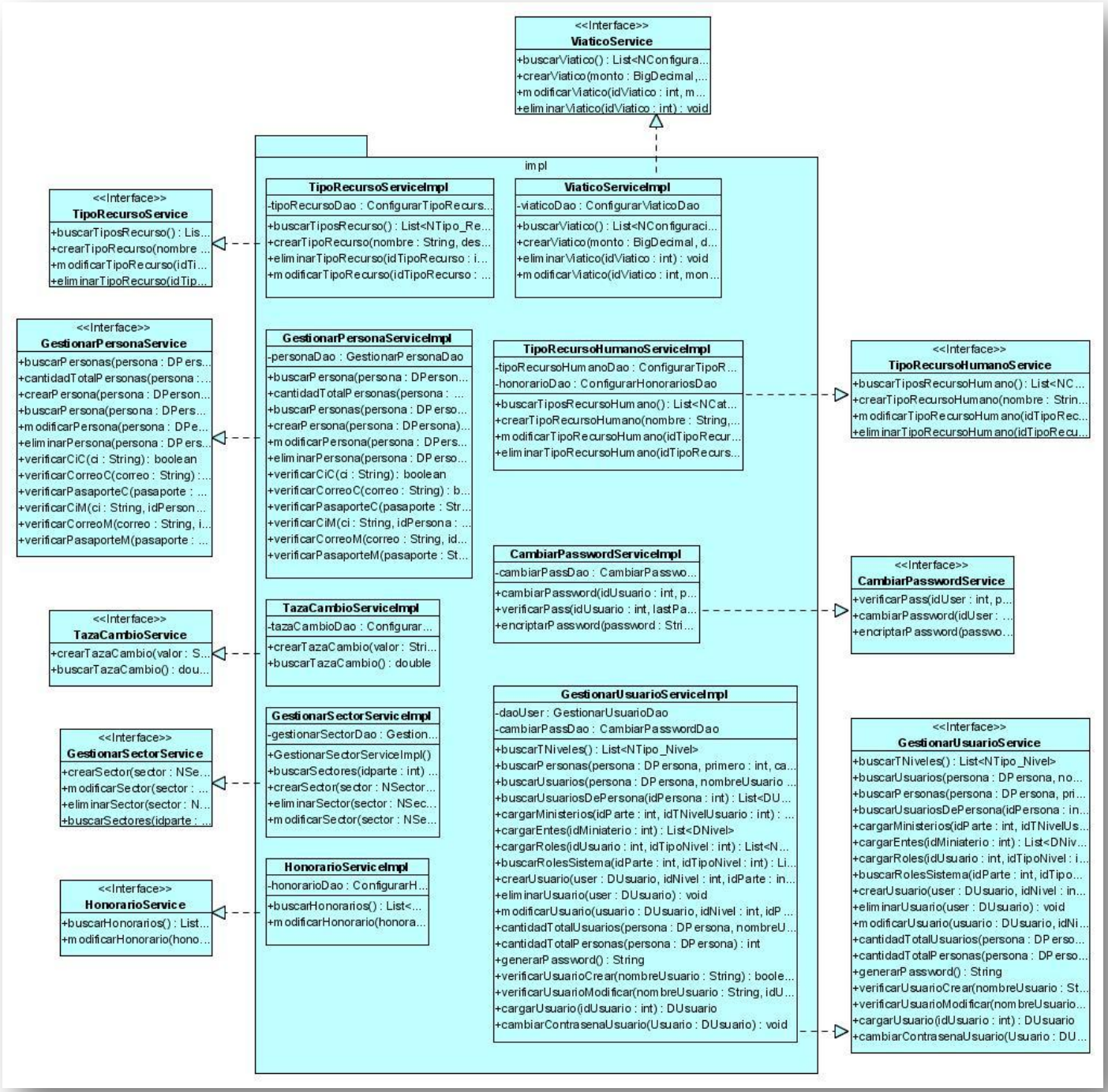


Figura 2.13: Diagrama de clases del diseño para la capa lógica de negocio, del módulo Administración-Configuración.

2.6 Modelo de Implementación

El modelo de Implementación está compuesto principalmente por los diagramas de componentes, los cuales describen cómo los artefactos, clases y otros elementos de bajo nivel, se integran para formar componentes de alto nivel, así como las conexiones entre ellos. En este epígrafe se muestra cómo están estructurados los diagramas de componentes para cada uno de los módulos a los cuales concierne la investigación.

El diagrama de componentes en ambos módulos queda estructurado como presentan las Figuras 2.14 y 2.16, respectivamente. En estas se evidencia como el paquete **bussines** correspondiente a la capa lógica de negocio contiene todas las interfaces, y los componentes **services** (managers) del paquete **impl** las implementan. Además se hace necesario que el paquete **bussines** importe los paquetes **DAO** (Data Access Object) y **VO** (Value Object) correspondientes a la capa de Acceso a Datos, y en el caso específico del módulo de Contratación se importa también del paquete **COMMON** que contiene todas las clases comunes al resto del proyecto.

2.6.1 Módulo Contratación

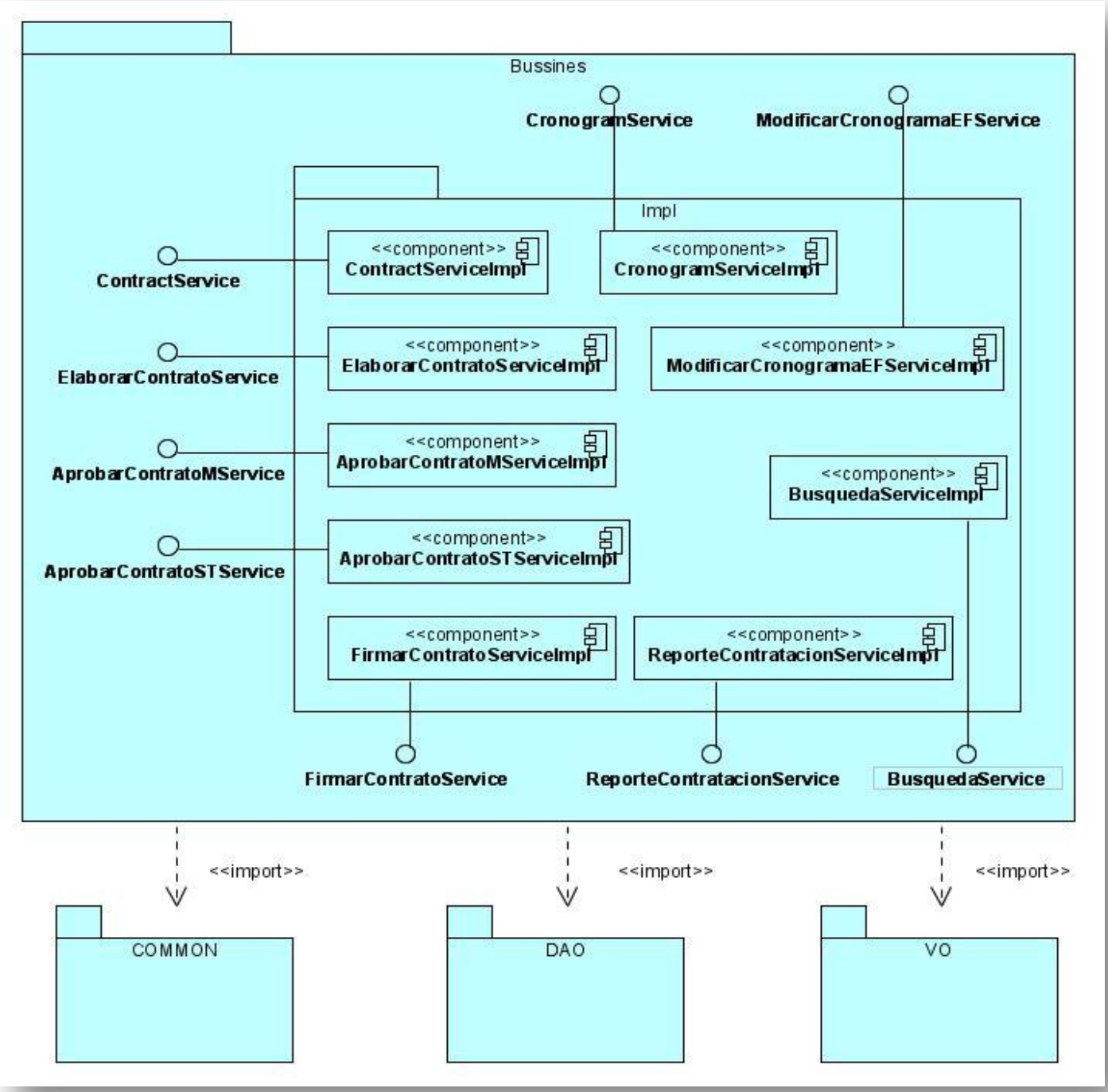


Figura 2.14: Diagrama de componentes para el módulo Contratación.

Seguidamente, en la Figura 2.15 se puede apreciar el diagrama de componentes de uno de los casos de usos fundamentales del Módulo de Contratación, pues se corresponde con el caso de uso de Gestionar Contrato.

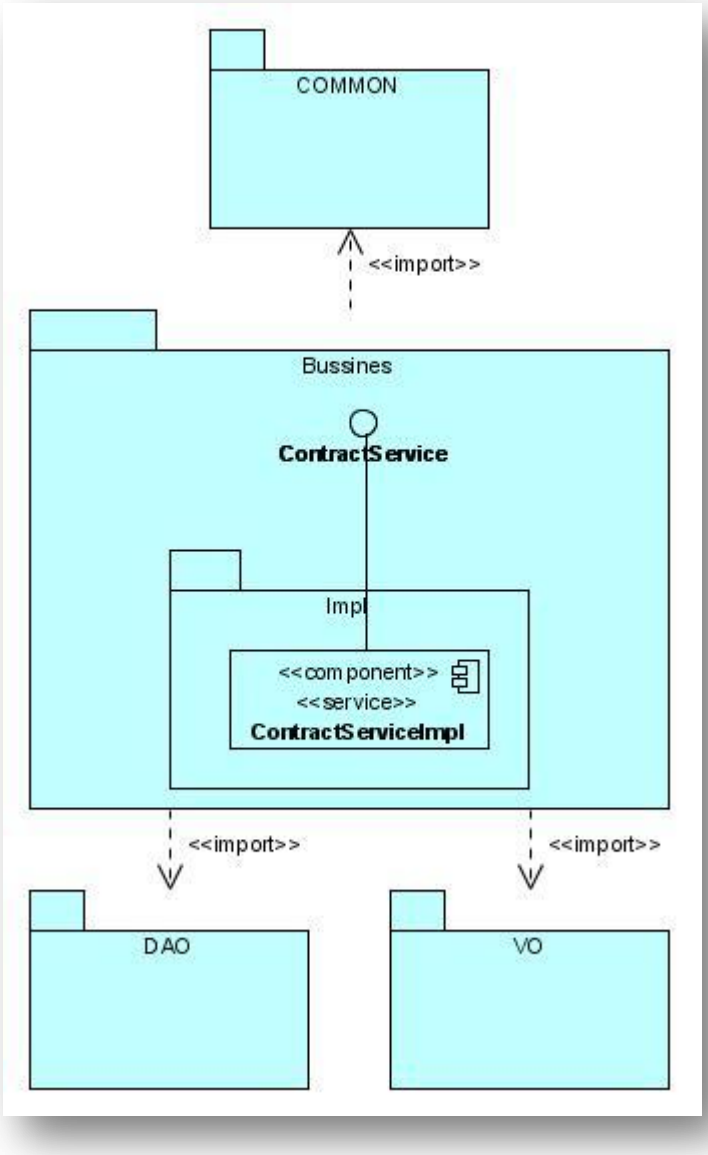


Figura 2.15: Diagrama de componentes para el CU Gestionar Contrato.

2.6.2 Módulo Administración-Configuración

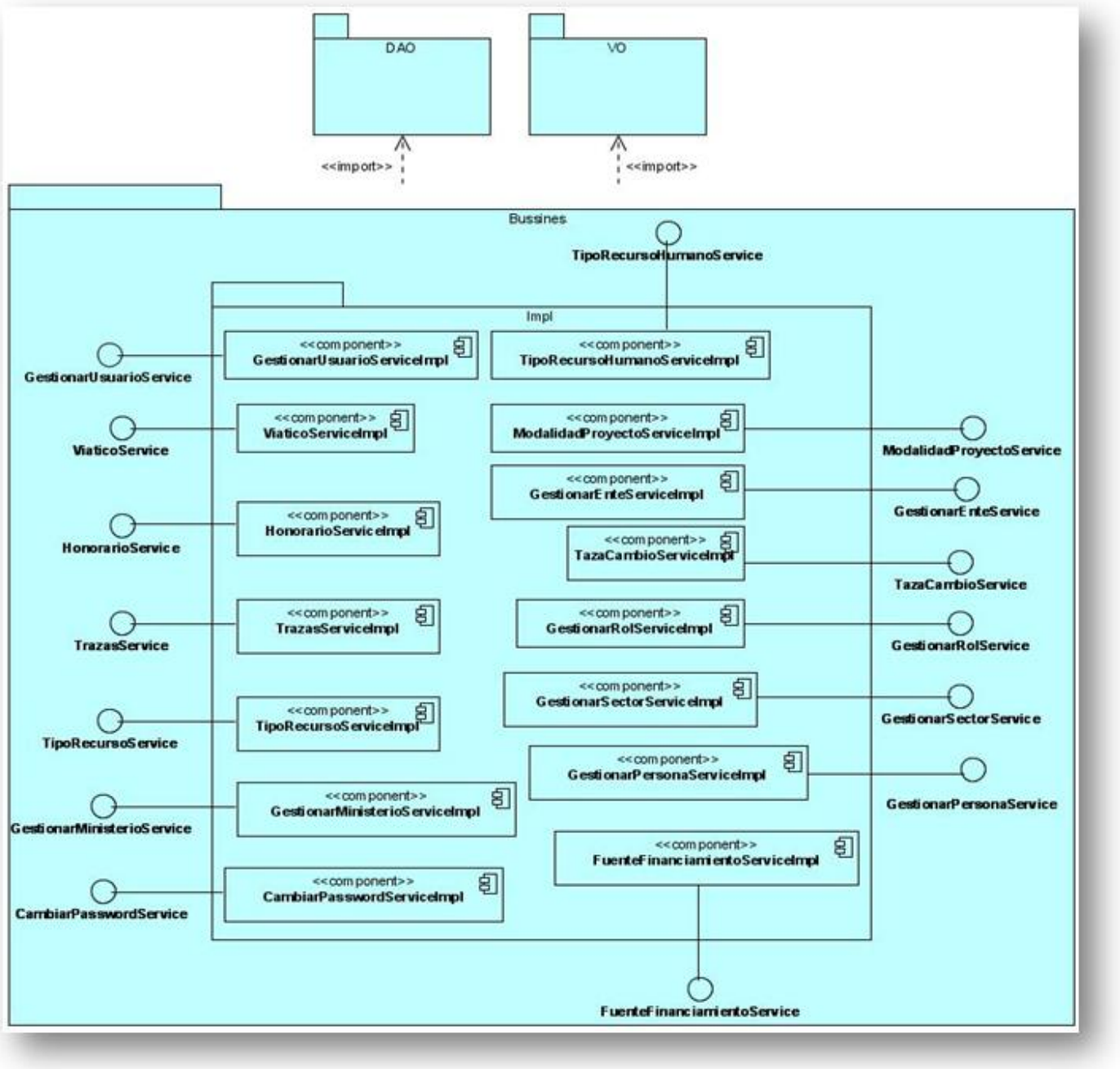


Figura 2.16: Diagrama de componentes para el módulo Administración-Configuración.

A continuación se muestran dos diagramas de componentes más específicos, correspondientes a los casos de uso, Gestionar Rol y Gestionar Usuario, Figuras 2.17 y 2.18, respectivamente:

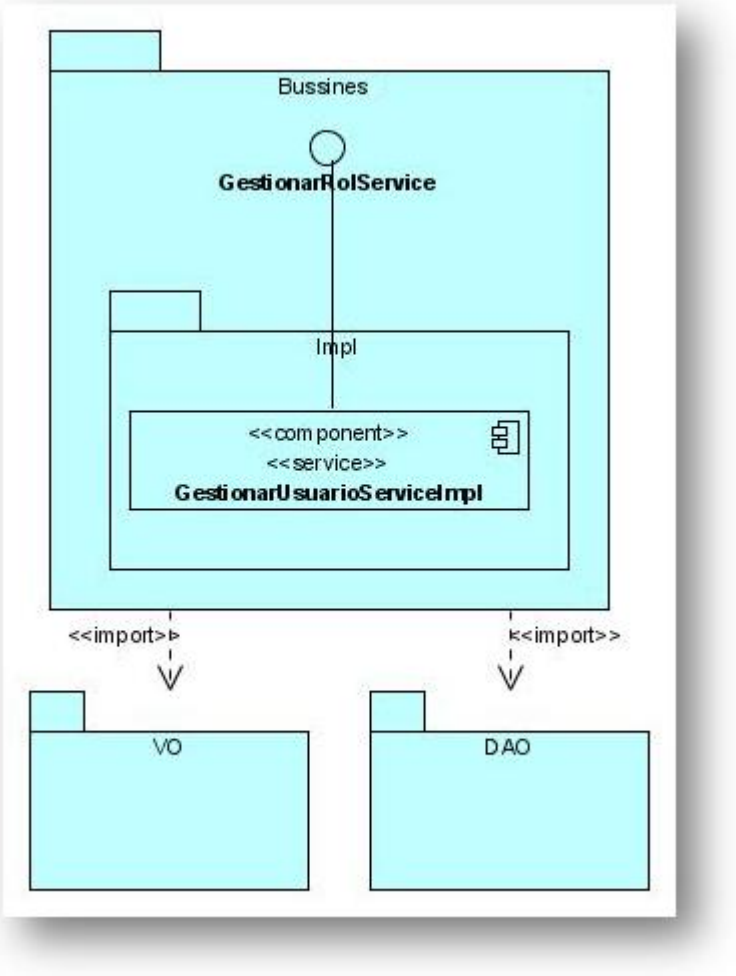


Figura 2.17: Diagrama de componentes para CU Gestionar Roles.

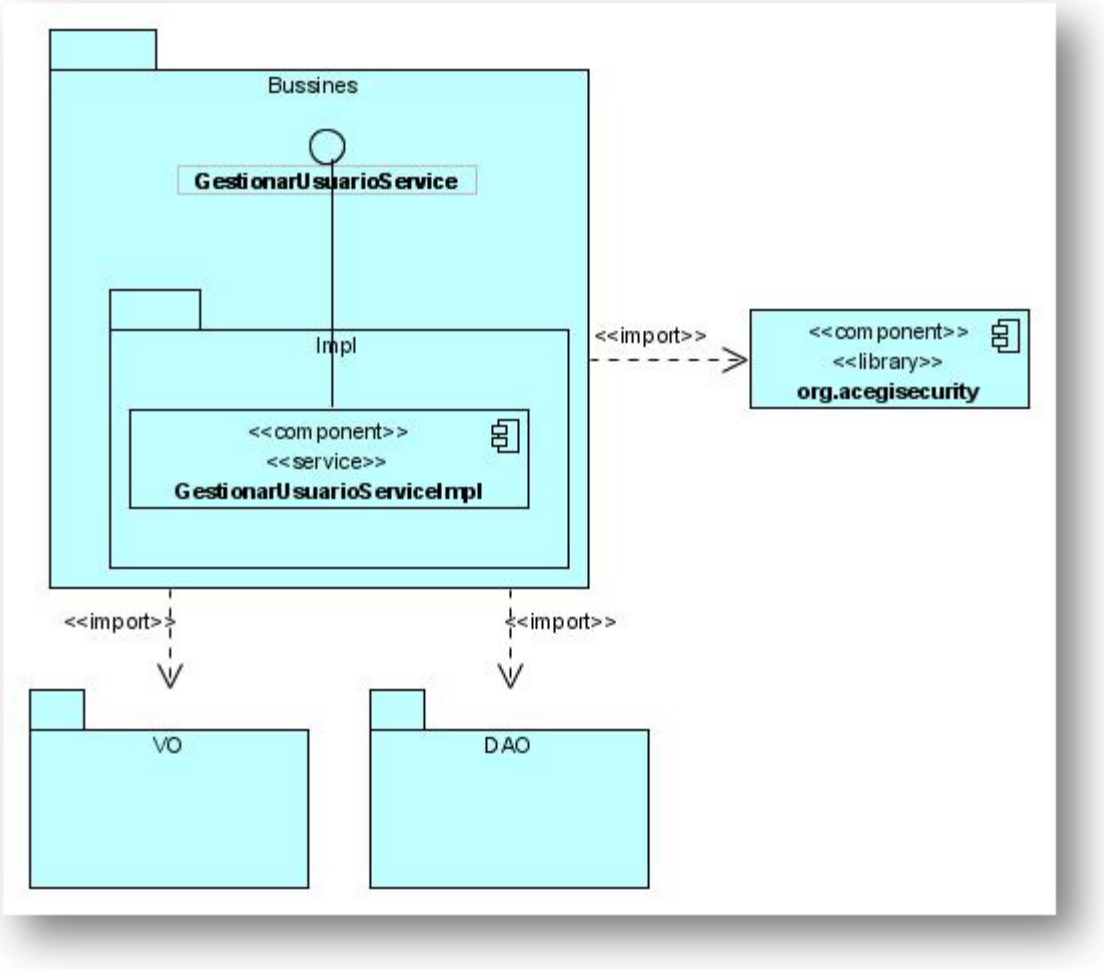


Figura 2.18: Diagrama de componentes para CU Gestionar Usuario.

2.7 Implementación y descripción de las clases del diseño

En este epígrafe se pormenorizan las clases más cardinales de la capa lógica de negocio tocantes a cada módulo que involucra la presente investigación.

Además durante la implementación de los módulos se emplearon algunos de los patrones específicamente para J2EE, estudiados en el capítulo anterior, como son:

- **Business Delegate:** En la mayoría de las clases implementadas en la capa de presentación del proyecto CCV, se incluye uno o varios objetos de la capa lógica de negocios, los cuales se encargan de llamar a las operaciones del negocio en beneficio de los componentes de la presentación.

- **Value Object/ Data Transfer Object/ Replicate Object:** Las clases persistentes VO (VALUE OBJECTS) importan al **java.io.Serializable** para que estas puedan tener un contrato con la clase Serializable, o sea, implementar esta para la transferencia de datos sobre la red.

- **Session Facade/ Session Entity Facade/ Distributed Facade:** Todas las clases de la capa lógica de negocio implementan una interfaz correspondiente para encapsular la complejidad de las interacciones de los objetos del negocio, las cuales actúan como una fachada (facade), y son las que se exponen a las clases de la capa de presentación.

- **Service Locator:** Este patrón viene implícito con el framework Spring, además de traer consigo la utilización del patrón de diseño Singleton, el cual es responsable de que una clase tenga solo una instancia proporcionando un punto de acceso global a ella, ejemplo de su uso se evidencia en el proyecto, en los XML ***-applicationContext-remoteClient*** y ***-applicationContext-remoteServer*** de cada módulo, reduciendo así la complejidad del código y proporcionando un punto de control.

2.7.2 Módulo Contratación

Las siguientes tablas describen las clases más importantes dentro del paquete **bussines/impl** que implementan a las interfaces referentes al módulo Contratación.

En lo sucesivo se presentarán las descripciones de las clases más relevantes con sus respectivos casos de uso. Específicamente, la clase que se exhibe a continuación, por su carácter especial, se utilizará en casi todos los casos de uso del módulo correspondiente:

Nombre de la Clase: BusquedaServiceImpl	
Atributos	
Nombre:	**Tipo:**
contratoDao	ContratoDao
cronogramaDao	CronogramaDao
desembolsoInvDao	DesembolsoInvDao
desembolsoTransfDao	DesembolsoTransfDao
rechazoDao	RechazoDao
Responsabilidades	
Nombre:	**Descripción:**
cargarDatosParaContratos(int idContrato): List<Object[]>	Devuelve una lista con todos los objetos que contiene un contrato en sí.
cargarContratos(String nombreFicha, int idEnte, int idMinisterio, int idNivel, int parte, int tipoNivel, int primerResultado, int cantidad, int marcoAprobacion): List<DContrato>	Carga los contratos con sus principales atributos en forma de objetos por separado con el objetivo de optimizar la consulta y arma el contrato como tal en esta capa del negocio nuevamente correspondiente al usuario logueado en el sistema.
cargarContratosPorProcesos(String nombreFicha,int idEnte, int idMinisterio, int idNivel, int parte,int tipoNivel, int idProceso, int primerResultado, int cantidad,int marcoAprobacion): List<DContrato>	Carga los contratos también con sus principales atributos en forma de objetos con el objetivo de obtener todos contratos que están en un proceso específico y que tiene el usuario logueado en el sistema.
cargarFichasPorContrato(int idContrato): List<Object[]>	Devuelve una lista de fichas para el contrato.
cargarFichasFinanciadas(int idEnte,int idMinisterio, int parte, int tipoNivel, int idNivel,int[] idFichas, int primerResultado, int cantidad,int marcoAprobacion):	Carga las fichas financiadas pasándole el arreglo con los id de las cuales es necesario cargar y le asigna a cada una de ellas un cronograma y un documento con los datos que ellos necesitan de la

List<DFicha_Proyecto>	ficha.
cargarFichasFinanciadas(int idEnte,int idMinisterio, int parte, int tipoNivel, int idNivel,int primerResultado, int cantidad, int marcoAprobacion): List<DFicha_Proyecto>	Carga las fichas financiadas y le asigna a cada una de ellas un cronograma y un documento con los datos que ellos necesitan de la ficha.
cantidadFichasFinanciadas(int idEnte, int idMinisterio,int parte, int tipoNivel, DNivel nivel, int marcoAprobacion): int	Devuelve un número entero con la cantidad de fichas financiadas que tiene el usuario logueado en el sistema.
cantidadContratos(String nombreFicha, int idEnte,int idMinisterio, DNivel nivel, int MarcoAprobacion): int	Devuelve el número entero con la cantidad de contratos correspondiente al usuario logueado en el sistema.
cantidadContratosPorProcesos(String nombreFicha, int idEnte,int idMinisterio, int idProceso, DNivel nivel, int marcoAprobacion): int	Devuelve el número entero con la cantidad de contratos que están en un proceso determinado que le corresponden al usuario logueado en el sistema.
cargarDocumentoAdjunto(int idContrato): byte[]	Devuelve un arreglo de bytes que contiene el documento adjunto de un contrato determinado.
cargarModalidadesPorFicha(int idFicha): List<Object[]>	Devuelve una lista de objetos que contiene las modalidades de una ficha determinada.
nombreFile(int idContrato): String	Devuelve el nombre del fichero del documento adjunto de un contrato específico.
cargarCronogramas(String nombreFicha, int idEnte,int idMinisterio, int idNivel, int parte, int tipoNivel,int primerResultado, int cantidad, int marcoAprobacion): List<Object[]>	Carga todos los cronogramas en forma de Lista de objetos relacionados con el usuario logueado en el sistema.
cargarCronogramasPorProcesos(String NombreFicha, int idEnte, int idMinisterio, int idNivel, int parte, int tipoNivel, int idProceso, int primerResultado, int cantidad, int marcoAprobacion): List<Object[]>	Carga todos los cronogramas que están en un proceso determinado en forma de Lista de objetos relacionados con el usuario logueado en el sistema.
cantidadCronogramas(String nombreFicha, int	Devuelve el número entero con la cantidad de

idEnte,int idMinisterio, DNivel nivel, int MarcoAprobacion): int	cronogramas correspondiente al usuario logueado en el sistema.
cantidadCronogramasPorProcesos(String nombreFicha, int idEnte,int idMinisterio, int idProceso, DNivel nivel, int marcoAprobacion): int	Devuelve el número entero con la cantidad de cronogramas que están en un proceso determinado que le corresponden al usuario logueado en el sistema.
cargarDesembolsoInvPorFicha(int idActividad): List<DDesembolsoInv>	Carga una lista con los Desembolsos de Inversión de una actividad determinada de una ficha de proyecto.
cargarDesembolsoTransfPorFicha(int idActividad): List<DDesembolsoTransf>	Carga una lista con los Desembolsos de Transferencia de una actividad determinada de una ficha de proyecto.
cargarActividades(Integer idFicha): List<DActividad>	Carga una lista con las actividades que contiene una ficha determinada.
cargarFichasContratadasPorEnte(int idEnte, DParte parte): List<Object[]>	Carga las fichas incluidas en algún contrato que tenga un ente determinado.
cargarFichasNoContratadasPorEnte(int idEnte, DParte parte): List<Object[]>	Carga las fichas que no se encuentran incluidas en ningún contrato de los que le pertenece a un ente determinado.
cargarMontosPorFicha(int idFicha): List<Object[]>	Carga en una lista de objetos los montos de una ficha determinada.
cargarRechazoPorDocumento(int idDocumento, int idEstado, int idProceso): List<Object[]>	Carga en una lista de objetos, para hacer la consulta mas optima, todos los rechazos de un documento determinado.

Tabla 1: Descripción de la clase BusquedaServiceImpl.

CU Gestionar Contrato

Nombre de la Clase: ContractServiceImpl	
Atributos	
Nombre:	**Tipo:**
contratoDao	ContratoDao
Responsabilidades	
Nombre:	**Descripción:**
setNull(DContrato contrato): void	Este método restablece todos los valores de los atributos del contrato y los pone en nulo.
setNullAdendum(DContrato contrato): void	Restablece todos los valores de los atributos pero esta vez de un adendum perteneciente a un contrato dado y los pone en nulo.
replicarContrato(DContrato contrato): DContrato	Devuelve un contrato clonado o replicado con respecto al que se le pasa por parámetros.
actualizarContrato(int idFicha, int idContrato): void	Referencia una ficha dada a un contrato dado.
desactualizarContrato(int idFicha): void	Le borra a la ficha la referencia al contrato que pertenece
documentoDadoId(int idContrato): DContrato	Carga un contrato dado el id.
actualizarEstadoProcesoFicha(int idFicha, int idEstado): void	Modifica el proceso en el que se encuentra una ficha con un id dado y en un estado dado perteneciente a un contrato.
replicarContratoAdendum(DContrato contrato): DContrato	Devuelve un contrato clonado o replicado con respecto al que se le pasa por parámetros pero esta vez crea también la referencia de los adendum a dicho contrato.

cargarEntesFicha(int idFicha): List<Object[]>	Carga los entes que tienen las fichas que tiene el contrato.
reordenarFichas():void	Reordena o redirecciona las fichas referenciándolas nuevamente al contrato.

Tabla 2: Descripción de la clase ContractServiceImpl.

Nombre de la Clase: ElaborarContratoServiceImpl	
Atributos	
Nombre:	**Tipo:**
contractService	ContractService
Responsabilidades	
Nombre:	**Descripción:**
crearPropuesta(DContrato contrato, DParte parte): void	Crea la propuesta de contrato poniéndolo desde ya en el estado y el proceso correspondientes
aceptarPropuesta(DContrato contrato, NTipo_Nivel tipoNivel,DParte parte): void	Acepta o aprueba la propuesta de contrato anteriormente realizada llamando al método Procesar Operación que a su vez llama al Siguiente Flujo implementado en la clase padre ProcesoServiceImpl.
rechazarPropuesta(DContrato contrato, NTipo_Nivel tipoNivel,DRechazo rechazo, DParte parte): void	Rechaza la propuesta de contrato anteriormente realizada llamando al método Procesar Operación pero además crea un rechazo para ese documento explicando el por qué.
enviarContrato(DContrato contrato, NTipo_Nivel tipoNivel,DParte parte): void	Envía la propuesta de contrato anteriormente realizada al próximo nivel llamando al método Procesar Operación.
aceptarEnvio(DContrato contrato, NTipo_Nivel	Acepta el envío del contrato al próximo nivel

tipoNivel,DParte parte): void	llamando al método Procesar Operación.
rechazarEnvio(DContrato contrato, NTipo_Nivel tipoNivel, DRechazo rechazo, DParte parte): void	Rechaza el envío del contrato llamando al método Procesar Operación pero además crea un rechazo para ese documento explicando el por qué.
eliminarPropuesta(DContrato contrato, NTipo_Nivel tipoNivel, DParte parte): void	Elimina la propuesta de contrato llamando al método Procesar Operación.
aceptarEliminacion(DContrato contrato): void	Acepta la eliminación de la propuesta de contrato anteriormente realizada llamando al método Procesar Operación.
rechazarEliminacion(DContrato contrato, NTipo_Nivel tipoNivel, DRechazo rechazo, DParte parte): void	Rechaza la eliminación de la propuesta de contrato anteriormente realizada llamando al método Procesar Operación.
modificarContrato(DContrato contrato, NTipo_Nivel tipoNivel, DParte parte): void	Modifica el contrato llamando al método Procesar Operación, pero como se debe guardar un anterior se crea un nuevo contrato con los nuevos atributos modificados.
aceptarModificacion(DContrato contrato, NTipo_Nivel tipoNivel, DParte parte): void	Acepta la modificación del contrato actualizando las referencia de las fichas al nuevo contrato creado a las hora de modificar y se pone de anterior el contrato del que se partió para hacer la modificación.
rechazarModificacion(DContrato contrato, NTipo_Nivel tipoNivel, DRechazo rechazo, DParte parte): void	Rechaza la modificación del contrato actualizando las referencia de las fichas al mismo que referenciaban anteriormente y se pone de anterior el contrato clonado o candidato a ser el modificado.
procesarOperacion(DContrato contrato, NTipo_Nivel tipoNivel, int operacion, DParte parte): void	Procesa las operaciones llamando al método Siguiente Flujo implementado en la clase padre ProcesoServiceImpl el cual le cambia el estado y el proceso a los documentos.

Tabla 3: Descripción de la clase ElaborarContratoServiceImpl.

CU Gestionar Cronograma

Nombre de la Clase: CronogramServiceImpl	
Atributos	
Nombre:	**Tipo:**
cronogramaDao	CronogramaDao
Responsabilidades	
Nombre:	**Descripción:**
setNull(DFicha_Proyecto ficha): void	Restablece todos los valores de los atributos del cronograma y los pone en nulo.
replicarCronogramaEF(DFicha_Proyecto ficha): DFicha_Proyecto	Replica el Cronograma de Ejecucion Financiera restableciendo los atributos de la ficha y poniéndolos en null y creándola nuevamente pues el cronograma depende íntegramente de la ficha.
documentoDadoId(int idCronograma): DCronograma	Carga un cronograma dado el id.
eliminarDesembolsosActividad(int idActividad): void	Elimina los desembolsos de una actividad contenida por el cronograma.

Tabla 4: Descripción de la clase CronogramServiceImpl.

Nombre de la Clase: ModificarCronogramaEFServiceImpl	
Atributos	
Nombre:	**Tipo:**
cronogramService	CronogramService

actividadDao	ActividadDao
Responsabilidades	
Nombre:	**Descripción:**
modificarCronogramaEF(DCronograma cronograma, NTipo_Nivel tipoNivel, DParte parte): void	Modifica el cronograma llamando al método Procesar Operación, pero como se debe guardar un anterior se crea un nuevo cronograma con los nuevos atributos modificados.
aceptarModificacion(DCronograma cronograma, NTipo_Nivel tipoNivel, DParte parte): void	Acepta la modificación del cronograma y se pone de anterior el contrato del que se partió para hacer la modificación.
rechazarModificacion(DCronograma cronograma, NTipo_Nivel tipoNivel, DRechazo rechazo, DParte parte): void	Rechaza la modificación del cronograma y se pone de anterior el cronograma clonado o candidato a ser el modificado.
procesarOperacion(DContrato contrato, NTipo_Nivel tipoNivel, int operacion, DParte parte): void	Procesa las operaciones llamando al método Siguiente Flujo implementado en la clase padre ProcesoServiceImpl el cual le cambia el estado y el proceso a los documentos.
eliminarDesembolsosActividad(DActividad acti): void	Elimina todos los desembolsos de una actividad.
setearDesembolsosActividad(DActividad acti, Set<DDesembolsoInv> invs, Set<DDesembolsoTransf> tranfs): void	Le pasa nuevos desembolsos tanto de inversión como de transferencia a una actividad.

Tabla 5: Descripción de la clase ModificarCronogramaEFServiceImpl.

2.7.3 Módulo Administración-Configuración

Las siguientes tablas describen las clases más importantes dentro del paquete **bussines/impl** que implementan a las interfaces referentes al módulo Administración-Configuración:

CU Gestionar Usuario

Nombre de la Clase: GestionarUsuarioServiceImpl	
Atributos	
Nombre:	**Tipo:**
daoUser	GestionarUsuarioDao
cambiarPassDao	CambiarPasswordDao
Responsabilidades	
Nombre:	**Descripción:**
buscarTNiveles():List<NTipo_Nivel>	Devuelve una lista con todos los tipos de niveles que tienen los usuarios.
buscarPersonas (DPersona persona, int primero, int cantidad):List<DPersona>	Devuelve una lista de personas con paginado, filtrando por la persona pasada por parámetro.
buscarUsuarios (DPersona persona, String nombreUsuario, int nivel, int idMinisterio, int idEnte, int primero, int cantidad, DUsuario usuario): List<DUsuario>	Devuelve una lista de usuarios con paginado, filtrando por determinados parámetros como persona, nombre de usuario, nivel, identificador del ministerio, identificador del ente y usuario.
buscarUsuariosDePersona(int idPersona): List<DUsuario>	Devuelve una lista de usuarios, filtrando por el identificador de una persona.
cargarMinisterios (int idParte, int idTNivelUsuario): List<DNivel>	Devuelve una lista de ministerios (DNivel) filtrando por el identificador de la Parte y el identificador del Nivel de Usuario.
cargarEntes(int idMiniaterio): List<DNivel>	Devuelve una lista de entes(DNivel) pertenecientes a un Ministerio, dado su identificador.
cargarRoles(int idUsuario, int idTipoNivel): List<NRol>	Devuelve una lista de roles, filtrando por un identificador de usuario, y un identificador de tipo de nivel.
buscarRolesSistema(int idParte, int	Devuelve una lista de roles, filtrando por un

idTipoNivel): List<NRol>	identificador de Parte, y un identificador de tipo de nivel.
crearUsuario(DUsuario user, int idNivel, int idParte): void	Crea un usuario, con un determinado nivel y perteneciendo a una Parte.
eliminarUsuario(DUsuario user): void	Elimina un usuario pasado por parámetro.
modificarUsuario(DUsuario usuario, int idNivel, int idParte): void	Modifica un usuario pasado por parámetro, además de su nivel y Parte.
cantidadTotalUsuarios(DPersona persona, String nombreUsuario, int nivel, DUsuario usuario, int idMinisterio, int idEnte):int	Devuelve la cantidad total de usuarios dados una persona, un nombre de usuario, un identificador de nivel, un usuario, un identificador de ministerio y un identificador de ente.
cantidadTotalPersonas(DPersona persona): int	Devuelve la cantidad total de personas dada una persona pasada por parámetro.
generarPassword(): String	Genera y devuelve un password.
verificarUsuarioCrear(String nombreUsuario): Boolean	Verifica si existe un usuario para crearlo, filtrando con el nombre de usuario pasado por parámetro, devuelve **true** en caso afirmativo y **false** en caso contrario.
verificarUsuarioModificar(String nombreUsuario, int idUsuario): Boolean	Verifica si existe un usuario para modificarlo, filtrando con el nombre de usuario e identificador pasado por parámetro, devuelve **true** en caso afirmativo y **false** en caso contrario.
cargarUsuario(int idUsuario):DUsuario	Carga un usuario dado el identificador de este pasado por parámetro.
cambiarContrasenaUsuario(DUsuario Usuario): void	Cambia la contraseña del usuario pasado por parámetro.

Tabla 6: Descripción de la clase GestionarUsuarioServiceImpl.

CU Gestionar Roles

Nombre de la Clase: GestionarRolServiceImpl	
Atributos	
Nombre:	**Tipo:**
dao	GestionarRolDao
Responsabilidades	
Nombre:	**Descripción:**
eliminarRol(NRol rol): String	Elimina un rol pasado por parámetro, devuelve “si” si se logra eliminar, y “no” en caso contrario.
cargarFuncionalidadesRol(int idRol): Set<NFuncionalidad>	Devuelve una colección de funcionalidades, dado un identificador de rol.
cargarFuncionalidades(int idRol): List<NFuncionalidad>	Devuelve una lista con las funcionalidades que no posean el rol pasado por parámetro.
cargarRoles(DUsuario usuario, int inicio, int total): List<NRol>	Devuelve una lista de roles con paginado, filtrando por un usuario pasado por parámetro.
cargarTiposNivel(): List<NTipo_Nivel>	Devuelve una lista de los tipos de niveles.
modificarRol(NRol rol): Boolean	Modifica el rol, pasado por parámetro.
crearRol(NRol rol): Boolean	Crea un rol, pasado por parámetro.
cantidadTotalRoles (int idParte): int	Devuelve la cantidad total de roles que tiene una determinada Parte.
cargarRol(int idRol): NRol	Devuelve un rol, dado su identificador.

Tabla 7: Descripción de la clase GestionarRolServiceImpl.

CU Gestionar Persona

Nombre de la Clase: GestionarPersonaServiceImpl	
Atributos	
Nombre:	**Tipo:**
personaDao	GestionarPersonaDao
Responsabilidades	
Nombre:	**Descripción:**
buscarPersona(DPersona persona): DPersona	Devuelve un objeto persona, dado otro objeto persona pasado por parámetro con su identificador.
cantidadTotalPersonas(DPersona persona): int	Devuelve la cantidad total de personas dada una persona pasada por parámetro.
buscarPersonas(DPersona persona, int primero, int cantidad): List<DPersona>	Devuelve una lista de personas con paginado dada una persona pasada por parámetro.
crearPersona(DPersona persona): void	Crea la persona pasada por parámetro.
modificarPersona (DPersona persona): void	Modifica la persona pasada por parámetro.
eliminarPersona(DPersona persona): Boolean	Elimina la persona pasada por parámetro, devuelve **true** si se elimina y **false** en caso contrario.
verificarCiC(String ci): Boolean	Verifica si existe un CI dado este, devuelve **true** en caso afirmativo y **false** en caso contrario.
verificarCorreoC(String correo): Boolean	Verifica si existe un correo dado este, devuelve **true** en caso afirmativo y **false** en caso contrario.
verificarPasaporteC (String pasaporte): Boolean	Verifica si existe un pasaporte dado este, devuelve **true** en caso afirmativo y **false** en caso contrario.
verificarCiM(String ci, int idPersona): Boolean	Verifica si existe una determinada persona dado su identificador, que posea el CI pasado por parámetro, devuelve **true** en caso afirmativo y **false**

	en caso contrario.
verificarCorreoM(String correo, int idPersona): Boolean	Verifica si existe una determinada persona dado su identificador, que posea el correo pasado por parámetro, devuelve **true** en caso afirmativo y **false** en caso contrario.
verificarPasaporteM(String pasaporte, int idPersona): Boolean	Verifica si existe una determinada persona dado su identificador, que posea el pasaporte pasado por parámetro, devuelve **true** en caso afirmativo y **false** en caso contrario.

Tabla 8: Descripción de la clase GestionarPersonaServiceImpl.

CU Gestionar Datos de Ministerios

Nombre de la Clase: GestionarMinisterioServiceImpl	
Atributos	
Nombre:	**Tipo:**
ministerioDao	GestionarMinisterioDao
Responsabilidades	
Nombre:	**Descripción:**
buscarMinisterios(int idParte, String siglas, String nombreMinisterio,int idSector, int primero, int cantidad): List<DNivel>	Devuelve una lista de Ministerios (DNivel) con paginado, filtrando por el identificador de la Parte, las siglas, el nombre del Ministerio y el identificador del Sector.
crearMinisterio(DNivel nivel):void	Crea un ministerio dado este.
eliminarMinisterio(DNivel nivel): Boolean	Elimina un ministerio dado este, devuelve **true** en caso afirmativo y **false** en caso contrario.

modificarMinisterio(DNivel nivel): void	Modifica un ministerio dado este.
cantidadTotalM(int idparte, String siglas, String nombreMinisterio ,int idsector): int	Devuelve la cantidad total que existen de Ministerios dado el identificador de la Parte, las siglas, el nombre y el identificador del Sector.
cantidadTotalC(DPersona persona): int	Devuelve la cantidad de coordinadores dada una persona.
devolverSectores(int idParte): List<NSector>	Devuelve una lista de Sectores dado el identificador de una Parte.

Tabla 9: Descripción de la clase GestionarMinisterioServiceImpl.

2.8 Algoritmos y Estructuras de Datos empleados

Con el objetivo de darle solución efectiva a diferentes problemas presentados en el negocio del sistema fue necesario la utilización de distintos algoritmos y estructuras de datos. Indudablemente de vital importancia resultó el uso de las estructuras de datos List y ArrayList, pues casi todas las funcionalidades dependen de estas estructuras debido a las facilidades que brindan para el almacenamiento y tratamiento de datos en general, con las cuales se hace con más calidad el trabajo, y además, son propias del lenguaje Java obteniéndose con solo incluir en la implementación los paquetes **java.util.List** y **java.util.ArrayList**. Sin embargo, también tuvo gran aceptación el uso del Set y del HashSet, otras estructuras muy parecidas a las listas solo que permiten más especificidades, ya que como son una colección que no contiene elementos duplicados, esto posibilita obtener listas sin ninguno de los elementos repetidos, viniendo con el Java también se puede contar con ellas al incluir los paquetes **java.util.Set** y **java.util.HashSet**.

Con la necesidad de extrema seguridad en el sistema, se constató el imperativo de crear usuarios cuyas contraseñas fueran generadas para que viajaran y fueran guardadas encriptadas en la base de datos a fin de impedir el robo de claves a los distintos usuarios, sabida la importancia que tienen a la hora de gestionar recursos. Por lo tanto, fue necesario utilizar un algoritmo para encriptar las contraseñas a **SHA**

(lenguaje de encriptación) que es muy usado y se obtiene a partir de la copia de un .**jar** que contiene las clases para el cifrado. Se logra correctamente la encriptación mediante el uso de la clase **ShaPasswordEncoder** que provee un importante framework encargado de brindarle seguridad al sistema, se trata del **AcegiSecurity**, el cual se vincula a Spring a través de distintas configuraciones. Finalmente se utilizó en el algoritmo encargado de generar las contraseñas e implementó de la siguiente manera:

```
public String generarPassword()throws Exception {

    Random rand = new Random();
    String pass = "";
    String alfabeto = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ";
    int size = alfabeto.length()-1;
    int ran = 0;

    for (int i = 0; i < 4; i++) {

          pass += Integer.toString(rand.nextInt(9));
          ran = rand.nextInt(52);

        if ((ran + 5) > size)
        pass += alfabeto.substring(ran-1, ran);
        else
        pass += alfabeto.substring(ran, ran+1);
       }

    ShaPasswordEncoder encripter = new ShaPasswordEncoder();

    String contra = encripter.encodePassword(pass, null);

    return contra;
}
```

Figura 2.19: Implementación del método generarPassword.

En el Sistema CCV la información contiene datos redundantes o datos no muy relevantes para la información que se necesita guardar, pues por ejemplo, se hace necesario archivar en la base de datos el documento Word que constituye la preforma del contrato. Esto produce grandes cantidades de datos que se transfieren entre el cliente y el servidor de aplicaciones. Un método para reducir el almacenamiento de los datos irrelevantes y la transferencia de la información es a través de representaciones de los datos con un código más eficiente, precisamente a través de la compresión y descompresión de datos, usando el paquete **java.util.zip**. [31]

Hay muchos beneficios con la compresión de los datos. No obstante, la ventaja principal es reducir requisitos de almacenamiento. También, para comunicaciones de los datos, la transferencia de datos comprimidos aumenta la cantidad de información transmitida.

Java proporciona el paquete **java.util.zip** para trabajar con archivos compatibles con el formato **.zip**. Este paquete proporciona clases que permiten leer, crear, y modificar archivos con los formatos ZIP y GZIP, pero además da la posibilidad de tratar datos comprimidos según esos estándares. Utiliza el algoritmo de compresión Deflate, el cual, al ser un algoritmo deflación, es un sistema de compresión de datos sin pérdidas que usa una combinación del algoritmo LZ77 y la codificación Huffman. También proporciona clases para comprobar las sumas de control de flujos de entrada y puede ser usado para validar entradas de datos. [32]

Este paquete en el sistema brinda la posibilidad de crear un archivo **.zip** con los documentos necesarios para que funcione disminuyendo el tamaño del mismo. Otra razón por la que se usó es que la compresión de datos en informática es un problema bastante complejo pues para estas operaciones se utilizan algoritmos que se basan en el álgebra de los números. También ofrece la posibilidad de comprimir y descomprimir estos datos, lo que contribuyó a reducir el tráfico de la red y mejorar el desempeño de la aplicación cliente/servidor. Seguidamente se ilustra esta clase con sus dos métodos para comprimir y descomprimir los archivos con el documento de la preforma de Contrato:

```java
import java.io.ByteArrayOutputStream;
import java.util.zip.DataFormatException;
import java.util.zip.Deflater;
import java.util.zip.Inflater;

public class Compresor
{
    public byte [] ComprimirArchivo (byte[] value) {
        ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
        byte[] buffer = new byte[1024];
        Deflater deflater = new Deflater(Deflater.BEST_COMPRESSION);
        deflater.setInput(value);
        deflater.finish();
        int bytes = 0;
        while((bytes = deflater.deflate(buffer)) > 0)
            outputStream.write(buffer, 0, bytes);
        return outputStream.toByteArray();
    }

    public byte[] DescomprimirArchivo(byte[] value) throws DataFormatException {
        ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
        byte[] buffer = new byte[1024];
        Inflater inflater = new Inflater();
        inflater.setInput(value);
        int bytes = 0;
        while((bytes = inflater.inflate(buffer)) > 0)
            outputStream.write(buffer, 0, bytes);
        return outputStream.toByteArray();
    }
}
```

Figura 2.20: Clase Compresor.

2.9 Conclusiones del Capítulo

En este capítulo se presentó una propuesta para darle solución a la capa lógica de negocio de los módulos Contratación y Administración-Configuración, donde, por supuesto, se mostró la estructura en paquetes del sistema y específicamente de la capa del negocio, la cual también está estructurada por

paquetes. Conjuntamente se ilustró el estándar de codificación y las razones por las cuales fue utilizado. También se detallaron los patrones de diseño empleados, las configuraciones y la política seguida para la integración con las otras capas de presentación y acceso a datos. Además, se pudo apreciar el diseño de las clases del negocio, el modelo de implementación del negocio, y la implementación llevada a cabo mediante la descripción de las clases del diseño del negocio de ambos módulos. Por otro lado, se trataron las estructuras de datos utilizadas, así como algunos algoritmos empleados con sus fines específicos.

Después de haber desarrollado dicha propuesta de solución se hace necesario llevar a cabo un conjunto de pruebas para verificar y validar que la misma se puede ejecutar correctamente y de la manera más óptima, pero además que cumpla con los objetivos por los cuales se desarrolló.

Capítulo 3. Validación de la solución propuesta

3.1 Introducción

En el presente capítulo se efectúa la validación de la solución propuesta en el capítulo 2, para ello se realiza la descripción de las clases de pruebas, así como pruebas de unidad con el framework JUnit para validar el código de los módulos Contratación y Administración-Configuración.

3.2 Descripción de las clases de pruebas

A continuación se describen las clases de prueba más importantes, separadas por casos de uso, que se utilizaron para validar la implementación de los módulos que concierne a la investigación.

3.2.1 Módulo Contratación

CU Gestionar Contrato

Caso de Prueba	**cargarDatosParaContratosTest**			
Condiciones	Para que esta prueba se cumpla satisfactoriamente debe estar configurada la base de datos correspondiente y deben existir contratos ya almacenados.			
Clases Válidas	**Clases Inválidas**	**Resultado Esperado**	**Resultado de la Prueba**	**Observaciones**
idContrato=1	idContrato=null	- El contrato está almacenado en la base de datos y se espera que el método de prueba devuelva los datos	Clases Válidas: - Se devuelven los datos que pertenecen al contrato.	En el caso de que el identificador del contrato sea nulo, o no coincida con ningún id almacenado, se levanta una

		del mismo. - En el caso de que el contrato no tenga los datos que se piden, se espera una lista vacía.	Clases Inválidas: - Se devuelve una lista vacía.	excepción y se trata. El resultado de la prueba fue satisfactorio.

Caso de Prueba	**cargarDocumentoAdjuntoTest**			
Condiciones	Para que esta prueba se cumpla satisfactoriamente debe estar configurada la base de datos correspondiente y deben existir contratos con sus respectivos documentos adjuntos ya almacenados.			
Clases Válidas	**Clases Inválidas**	**Resultado Esperado**	**Resultado de la Prueba**	**Observaciones**
idContrato=1	idContrato=null	- El contrato está almacenado en la base de datos con su documento adjunto y se espera que el método de prueba cargue este documento adjunto. - En el caso de que el contrato no tenga los datos que se piden, se espera una lista vacía.	Clases Válidas: - Se devuelve la lista con los ministerios filtrados. Clases Inválidas: - Se devuelve una lista vacía.	En el caso de que el identificador del contrato sea nulo, o no coincida con ningún id almacenado, se levanta una excepción y se trata. El resultado de la prueba fue satisfactorio.

Caso de Prueba	cargarFichasPorContratoTest			
Condiciones	Para que esta prueba se cumpla satisfactoriamente debe estar configurada la base de datos correspondiente y debe tener contratos y fichas ya almacenados			
Clases Válidas	**Clases Inválidas**	**Resultado Esperado**	**Resultado de la Prueba**	**Observaciones**
idContrato=1	idContrato=null	- El contrato y las fichas están almacenadas en la base de y se espera que el método de prueba cargue todas las fichas de ese contrato. - En el caso de que el contrato no tenga fichas, se espera una lista vacía.	Clases Válidas: - Se devuelve la lista con las fichas pertenecientes a este contrato. Clases Inválidas: - Se devuelve una lista vacía.	En el caso de que el identificador del contrato sea nulo, o no coincida con ningún id almacenado, se levanta una excepción y se trata. El resultado de la prueba fue satisfactorio.

Caso de Prueba	cantidadFichasFinanciadasTest			
Condiciones	Para que esta prueba se cumpla satisfactoriamente debe estar configurada la base de datos correspondiente y deben haber fichas financiadas almacenadas.			
Clases Válidas	**Clases Inválidas**	**Resultado Esperado**	**Resultado de la Prueba**	**Observaciones**
idEnte=122 **idMinisterio=24** **parte=1** **tipoNivel=2**	idEnte= null idMinisterio= null parte= null tipoNivel= null	- El ente y el ministerio se encuentran en la base de datos y coinciden con la parte, el tipo de nivel, el nivel y el	Clases Válidas: - Se pudo conocer la cantidad de fichas financiadas que contienen o el ente o el ministerio.	En caso de que solo el ente sea null y todos los demás parámetros estén bien, se obtiene la cantidad de fichas

nivel= NivelPrueba **marcoAprobacion=2**	nivel= null marcoAprobacion= null	marco de aprobación, de esta manera se retorna la cantidad de fichas financiadas elimina. - En caso de que el ente y el ministerio no existan debe retornar una excepción anunciando el problema.	Clases Inválidas: - Se lanza una excepción.	financiadas del ministerio en cuestión. El resultado de la prueba fue satisfactorio.

Caso de Prueba	**replicarContratoTest**			
Condiciones	Para que esta prueba se cumpla satisfactoriamente debe estar configurada la base de datos correspondiente y lista para almacenar nuevos contratos.			
Clases Válidas	**Clases Inválidas**	**Resultado Esperado**	**Resultado de la Prueba**	**Observaciones**
contrato= ContratoPrueba	contrato=null	- Se espera un nuevo contrato clonado e insertado en la base de datos, por supuesto con casi todos los atributos iguales a los del contrato por el cual fue clonado. - En caso de que el contrato no pueda ser clonado se lanzaría una	Clases Válidas: - El contrato pudo ser clonado o replicado correctamente Clases Inválidas: - El contrato no pudo ser clonado o replicado	En caso de que el contrato sea null se levanta una excepción y no se hace ninguna verificación. El resultado de la prueba fue satisfactorio.

		excepción.		

Caso de Prueba	**replicarContratoAdendumTest**			
Condiciones	Para que esta prueba se cumpla satisfactoriamente debe estar configurada la base de datos correspondiente y lista para almacenar nuevos contratos con sus adendums.			
Clases Válidas	**Clases Inválidas**	**Resultado Esperado**	**Resultado de la Prueba**	**Observaciones**
contrato= ContratoPrueba	contrato=null	- Se espera un nuevo contrato clonado e insertado en la base de datos, por supuesto con casi todos los atributos, específicamente él o los adendums, iguales a los del contrato por el cual fue clonado. - En caso de que el contrato no pueda ser clonado se lanzaría una excepción.	Clases Válidas: - El contrato pudo ser clonado o replicado con sus adendums correctamente. Clases Inválidas: - El contrato no pudo ser clonado o replicado con sus adendums.	En caso de que el contrato que se pretende replicar sea null se levanta una excepción y no se hace ninguna verificación. El resultado de la prueba fue satisfactorio.

Caso de Prueba	**crearPropuestaTest**			
Condiciones	Para que esta prueba se cumpla satisfactoriamente debe estar configurada la base de datos correspondiente y lista para almacenar nuevos contratos.			
Clases Válidas	**Clases Inválidas**	**Resultado Esperado**	**Resultado de la Prueba**	**Observaciones**

contrato= ContratoPrueba **parte= PartePrueba**	contrato= null parte= null	- Se espera un nuevo contrato insertado en la base de datos y perteneciente a la parte a la que corresponde. - En caso de que la parte no se corresponda el contrato no podrá ser insertado en la base de datos.	Clases Válidas: - Se inserta correctamente la propuesta de contrato en la base de datos. Clases Inválidas: - No se puede insertar el contrato.	En caso de que la parte sea null tampoco se podría insertar el contrato. El resultado de la prueba fue satisfactorio.

Caso de Prueba	**modificarContratoTest**			
Condiciones	Para que esta prueba se cumpla satisfactoriamente debe estar configurada la base de datos correspondiente y lista para almacenar nuevos contratos al estos ser modificados.			
Clases Válidas	**Clases Inválidas**	**Resultado Esperado**	**Resultado de la Prueba**	**Observaciones**
contrato= ContratoPrueba **tipoNivel= TipoNivelPrueba** **parte= PartePrueba**	contrato= null tipoNivel= null parte= null	- Se espera el contrato modificado e insertado nuevamente en la base de datos y perteneciente a la parte a la que corresponde. - En caso de que la parte y el tipo de nivel no se correspondan, pero además si el	Clases Válidas: - Se modifica e inserta correctamente el contrato en la base de datos. Clases Inválidas: - No se puede modificar el contrato.	En caso de que la parte y el tipoNivel sean null tampoco se podría modificar el contrato. El resultado de la prueba fue satisfactorio.

		contrato no tiene valores, este no podrá ser modificado.		

CU Firmar Contrato

Caso de Prueba	**firmarContratoTest**			
Condiciones	Para que esta prueba se cumpla satisfactoriamente debe estar configurada la base de datos correspondiente.			
Clases Válidas	**Clases Inválidas**	**Resultado Esperado**	**Resultado de la Prueba**	**Observaciones**
contrato= ContratoPrueba **tipoNivel= TipoNivelPrueba** **parte= PartePrueba**	contrato= null tipoNivel= null parte= null	- Se espera que el contrato quede en estado firmado y sea de esta forma modificado en la base de datos nuevamente en la base de datos. - En caso de que la parte y el tipo de nivel no se correspondan, pero además si el contrato no tiene valores, este no podrá ser firmado.	Clases Válidas: - Se firma y se modifica correctamente el contrato en la base de datos. Clases Inválidas: - No se puede modificar el contrato.	En caso de que la parte y el tipoNivel sean null tampoco se podría modificar el contrato. El resultado de la prueba fue satisfactorio.

CU Gestionar Cronograma

Caso de Prueba	replicarContratoTest			
Condiciones	Para que esta prueba se cumpla satisfactoriamente debe estar configurada la base de datos correspondiente y lista para almacenar nuevos cronogramas.			
Clases Válidas	**Clases Inválidas**	**Resultado Esperado**	**Resultado de la Prueba**	**Observaciones**
ficha= FichaPrueba	ficha=null	- Se espera un nuevo cronograma clonado e insertado en la base de datos, a partir de replicar la ficha a la que pertenece, y por supuesto, prácticamente con los mismos atributos. - En caso de que el cronograma y la ficha no puedan ser clonados o replicados se lanzaría una excepción.	Clases Válidas: - El cronograma y la ficha a la que pertenece pudieron ser clonados o replicados correctamente. Clases Inválidas: - El cronograma y la ficha no pudieron ser clonados o replicados.	En caso de que la ficha sea null se levanta una excepción. El resultado de la prueba fue satisfactorio.

Caso de Prueba	modificarCronogramaEFTest			
Condiciones	Para que esta prueba se cumpla satisfactoriamente debe estar configurada la base de datos correspondiente y lista para almacenar nuevos cronogramas al estos ser modificados.			
Clases Válidas	**Clases Inválidas**	**Resultado Esperado**	**Resultado de la Prueba**	**Observaciones**

cronograma= cronogramaPrueba **tipoNivel= TipoNivelPrueba** **parte= PartePrueba**	cronograma= null tipoNivel= null parte= null	- Se espera el cronograma modificado e insertado nuevamente en la base de datos y perteneciente a la parte a la que corresponde. - En caso de que la parte y el tipo de nivel no se correspondan, pero además si el cronograma no tiene valores, este no podrá ser modificado.	Clases Válidas: - Se modifica e inserta correctamente el cronograma en la base de datos. Clases Inválidas: - No se puede modificar el cronograma.	En caso de que la parte y el tipoNivel sean null tampoco se podría modificar el cronograma. El resultado de la prueba fue satisfactorio.

CU Gestionar Reportes

Caso de Prueba	**reporteFichasContratadasTest**			
Condiciones	Para que esta prueba se cumpla satisfactoriamente debe estar configurada la base de datos correspondiente y deben haber fichas contratadas almacenadas.			
Clases Válidas	**Clases Inválidas**	**Resultado Esperado**	**Resultado de la Prueba**	**Observaciones**
idMinisterio=24 **idEnte=122** **idficha= 321** **parte=**	idMinisterio=null idEnte=null idficha= null parte=null	- El ente y el ministerio se encuentran en la base de datos y coinciden con la parte, de esta manera se retorna la cantidad de fichas	Clases Válidas: - Se pudieron cargar las fichas contratadas que contienen o el ente o el ministerio.	En caso de que solo el ente sea null y todos los demás parámetros estén bien, se obtiene la cantidad de fichas

PartePrueba		contratadas. - En caso de que el ente y el ministerio no existan debe retornar una excepción anunciando el problema.	Clases Inválidas: - Se lanza una excepción.	contratadas del ministerio en cuestión. El resultado de la prueba fue satisfactorio.

3.2.2 Módulo Administración-Configuración

CU Gestionar Usuario

Caso de Prueba	**buscarUsuariosDePersonaTest**			
Condiciones	Para que esta prueba se cumpla satisfactoriamente debe estar configurada la base de datos correspondiente y debe tener personas almacenadas.			
Clases Válidas	**Clases Inválidas**	**Resultado Esperado**	**Resultado de la Prueba**	**Observaciones**
idPersona=1	idPersona=null	- La persona está almacenada en la base de datos y se espera que el método de prueba devuelva los usuarios de esta. - En el caso de que la persona no tenga usuarios, se espera una lista vacía.	Clases Válidas: - Se devuelve los usuarios que tiene la persona. Clases Inválidas: - Se devuelve una lista vacía.	En el caso de que el identificador de la persona sea nulo, se levanta una excepción y se trata. El resultado de la prueba fue satisfactorio.

Caso de Prueba	cargarMinisteriosTest			
Condiciones	Para que esta prueba se cumpla satisfactoriamente debe estar configurada la base de datos correspondiente y debe tener partes, ministerios y niveles de usuario almacenados.			
Clases Válidas	**Clases Inválidas**	**Resultado Esperado**	**Resultado de la Prueba**	**Observaciones**
idParte=1 **idTNivelUsuario=1**	idParte=null idTNivelUsuario=null idParte=99 idTNivelUsuario=99	- Se cargan los ministerios correspondientes al identificador de la parte y al nivel de usuario. - En el caso de que no existan ministerios que coincidan con los parámetros, se espera una lista vacía.	Clases Válidas: - Se devuelve la lista con los ministerios filtrados. Clases Inválidas: - Se devuelve una lista vacía.	En el caso de que algún parámetro de la prueba sea nulo, se levanta una excepción y no se cargan los ministerios. El resultado de la prueba fue satisfactorio.

Caso de Prueba	eliminarUsuarioTest			
Condiciones	Para que esta prueba se cumpla satisfactoriamente debe estar configurada la base de datos correspondiente y debe de tener algún usuario almacenado.			
Clases Válidas	**Clases Inválidas**	**Resultado Esperado**	**Resultado de la Prueba**	**Observaciones**
User=”ybyanez”	User=null	- El usuario se encuentra en la base de datos y se elimina. - En caso de que no exista el usuario	Clases Válidas: - El usuario pudo ser eliminado correctamente Clases Inválidas:	En caso de que el usuario sea null o no exista no podrá ser eliminado El resultado de la

		en la base de datos, este no podrá ser eliminado.	- El usuario no pudo ser eliminado	prueba fue satisfactorio.

Caso de Prueba	**verificarUsuarioCrearTest**			
Condiciones	Para que esta prueba se cumpla satisfactoriamente debe estar configurada la base de datos correspondiente y debe de tener algún usuario almacenado.			
Clases Válidas	**Clases Inválidas**	**Resultado Esperado**	**Resultado de la Prueba**	**Observaciones**
nombreUsuario= ”ybyanez”	nombreUsuario= null	- Se chequea que el nombre de usuario pertenezca a algún usuario de la base de datos. - En caso de que no exista el usuario con ese nombre de usuario en la base de datos, entonces se devuelve **false**.	Clases Válidas: - El nombre de usuario pudo ser chequeado correctamente Clases Inválidas: - El nombre de usuario no pudo ser chequeado	En caso de que el nombre de usuario sea null, se levanta una excepción y no se hace la verificación. El resultado de la prueba fue satisfactorio.

CU Gestionar Roles

Caso de Prueba	**eliminarRolTest**			
Condiciones	Para que esta prueba se cumpla satisfactoriamente debe estar configurada la base de datos correspondiente y debe de tener algún rol almacenado.			
Clases Válidas	**Clases Inválidas**	**Resultado**	**Resultado de la**	**Observaciones**

		Esperado	Prueba	
Rol=”coordinador”	Rol=null	- El rol se encuentra en la base de datos y se elimina. - En caso de que no exista el rol en la base de datos, este no podrá ser eliminado.	Clases Válidas: - El rol pudo ser eliminado correctamente Clases Inválidas: - El rol no pudo ser eliminado	En caso de que el rol sea null o no exista no podrá ser eliminado El resultado de la prueba fue satisfactorio.

Caso de Prueba	**cargarFuncionalidadesTest**			
Condiciones	Para que esta prueba se cumpla satisfactoriamente debe estar configurada la base de datos correspondiente y debe de tener algún rol y sus funcionalidades almacenados.			
Clases Válidas	**Clases Inválidas**	**Resultado Esperado**	**Resultado de la Prueba**	**Observaciones**
idRol=1	idRol=null	- El rol se encuentra en la base de datos y devuelve una lista con todas las funcionalidades menos las de ese rol. - En caso de que no exista el rol en la base de datos, se devuelve una lista con todas las funcionalidades.	Clases Válidas: - Se devuelve correctamente la lista con todas las funcionalidades menos las de ese rol. Clases Inválidas: - Se devuelve una lista con todas las funcionalidades.	En caso de que el rol sea null o no exista se lanza una excepción y se trata devolviendo una lista con todas las funcionalidades. El resultado de la prueba fue satisfactorio.

Caso de Prueba	cantidadTotalRolesTest			
Condiciones	Para que esta prueba se cumpla satisfactoriamente debe estar configurada la base de datos correspondiente y debe de tener alguna parte almacenada.			
Clases Válidas	**Clases Inválidas**	**Resultado Esperado**	**Resultado de la Prueba**	**Observaciones**
idParte=1	idParte=null	- Se devuelve la cantidad de roles que posee esa determinada parte. - En caso de que no exista la parte o no tenga roles en la base de datos, se devuelve valor 0.	Clases Válidas: - Se devuelve correctamente la cantidad de roles que posee esa determinada parte. Clases Inválidas: - Se devuelve valor 0.	En caso de que la parte sea null o no exista se devuelve valor 0. El resultado de la prueba fue satisfactorio.

Caso de Prueba	modificarRolTest			
Condiciones	Para que esta prueba se cumpla satisfactoriamente debe estar configurada la base de datos correspondiente y debe de tener algún rol almacenado.			
Clases Válidas	**Clases Inválidas**	**Resultado Esperado**	**Resultado de la Prueba**	**Observaciones**
Rol="coordinador"	Rol=null	- El rol se encuentra en la base de datos y se modifica - En caso de que no exista el rol en la base de datos, no se modifica.	Clases Válidas: - Se modifica correctamente el rol. Clases Inválidas: - No se modifica el rol.	En caso de que el rol sea null o no exista se lanza una excepción y se trata. El resultado de la prueba fue satisfactorio.

CU Gestionar Persona

Caso de Prueba	**buscarPersonaTest**			
Condiciones	Para que esta prueba se cumpla satisfactoriamente debe estar configurada la base de datos correspondiente y debe de tener alguna persona almacenada.			
Clases Válidas	**Clases Inválidas**	**Resultado Esperado**	**Resultado de la Prueba**	**Observaciones**
Persona="Yovany"	Persona=null	- Se encuentra y se devuelve la persona en la base de datos. - En caso de que no exista la persona en la base de datos se devuelve la persona sin los datos necesarios del objeto.	Clases Válidas: - Se encuentra y se devuelve correctamente la persona. Clases Inválidas: - No se encuentra la persona y se devuelve un objeto persona sin los datos necesarios	En caso de que la persona sea null se lanza una excepción y se trata. El resultado de la prueba fue satisfactorio.

Caso de Prueba	**crearPersonaTest**			
Condiciones	Para que esta prueba se cumpla satisfactoriamente debe estar configurada la base de datos correspondiente.			
Clases Válidas	**Clases Inválidas**	**Resultado Esperado**	**Resultado de la Prueba**	**Observaciones**
Persona="Yovany"	Persona=null	- Se crea la nueva persona en la base de datos	Clases Válidas: - Se creó correctamente la persona en la base	En caso de que la persona sea null se lanza una excepción y no se

			de datos. Clases Inválidas: - No se crea la persona en la base de datos.	ejecuta el método. El resultado de la prueba fue satisfactorio.

Caso de Prueba	**verificarCiMTest**			
Condiciones	Para que esta prueba se cumpla satisfactoriamente debe estar configurada la base de datos correspondiente y debe de tener alguna persona almacenada.			
Clases Válidas	**Clases Inválidas**	**Resultado Esperado**	**Resultado de la Prueba**	**Observaciones**
Ci=”86012106420” **idPersona=”1”**	Ci=null idPersona=null	- Se verifica que la persona con ese identificador tenga ese Ci. - En caso de que no exista o no coincida la persona con el Ci en la base de datos se devuelve **false**.	Clases Válidas: - Se verifica correctamente que la persona con ese identificador tiene ese Ci. Clases Inválidas: - No se hace la correspondiente verificación.	En caso de que alguno de los parámetros sea null se lanza una excepción y no se hace la verificación El resultado de la prueba fue satisfactorio.

Caso de Prueba	**verificarPasaporteMTest**			
Condiciones	Para que esta prueba se cumpla satisfactoriamente debe estar configurada la base de datos correspondiente y debe de tener alguna persona almacenada.			
Clases Válidas	**Clases Inválidas**	**Resultado**	**Resultado de la**	**Observaciones**

		Esperado	Prueba	
Pasaporte= ”B4950881” **idPersona=”1”**	Pasaporte=null idPersona=null	- Se verifica que la persona con ese identificador tenga ese pasaporte. - En caso de que no exista o no coincida la persona con el pasaporte en la base de datos se devuelve **false**.	Clases Válidas: - Se verifica correctamente que la persona con ese identificador tiene ese pasaporte. Clases Inválidas: - No se hace la correspondiente verificación.	En caso de que alguno de los parámetros sea null se lanza una excepción y no se hace la verificación El resultado de la prueba fue satisfactorio.

3.3 Pruebas unitarias de validación con JUnit

Para llevar a cabo la validación de la implementación de los módulos correspondientes a la investigación se hizo necesario el uso del Framework JUnit, estudiado en el Capítulo 1, con el objetivo de realizar las pruebas unitarias correspondientes, guiadas por los casos de pruebas. Estas pruebas unitarias son una vía de probar la implementación de cada módulo por separado y más que eso, delimitados por las diferentes capas.

Con el uso de JUnit, se hace más fácil realizar los casos de prueba, pues este permite que sean ejecutados devolviendo visualmente al programador los resultados que se esperan, y en caso de que existan errores, devuelve una traza con los detalles de estos.

A continuación se muestra para cada uno de los módulos algunos de los ejemplos de realización de los casos de prueba usando JUnit:

3.3.1 Módulo Contratación

Con el fin de lograr la validación de los métodos se pusieron en práctica los casos de prueba anteriormente especificados. A continuación en la Figura 3.1 se muestra el resultado de probar el método crearPropuesta() cuando primeramente se prueba con las clases válidas, o sea pasándole los parámetros que necesita. Sin embargo, en la Figura 3.2 se prueba el mismo método pero con las clases inválidas, por lo que se efectúa la prueba con los parámetros cuyos valores son nulos. Seguidamente se constataron resultados satisfactorios, pues en sendos casos fueron los esperados:

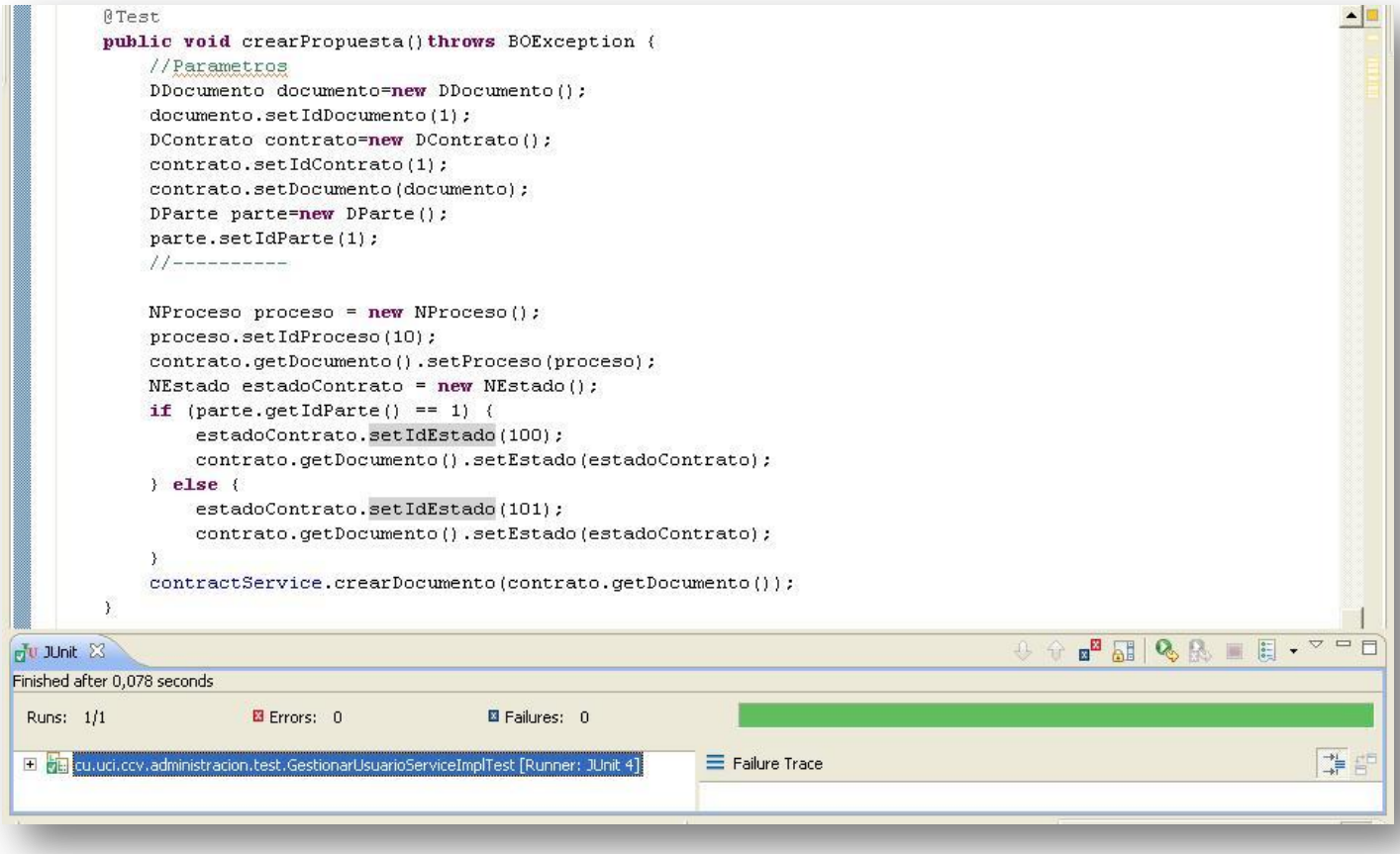


Figura 3.1: Ejecución exitosa del método crearPropuesta validándose con JUnit.

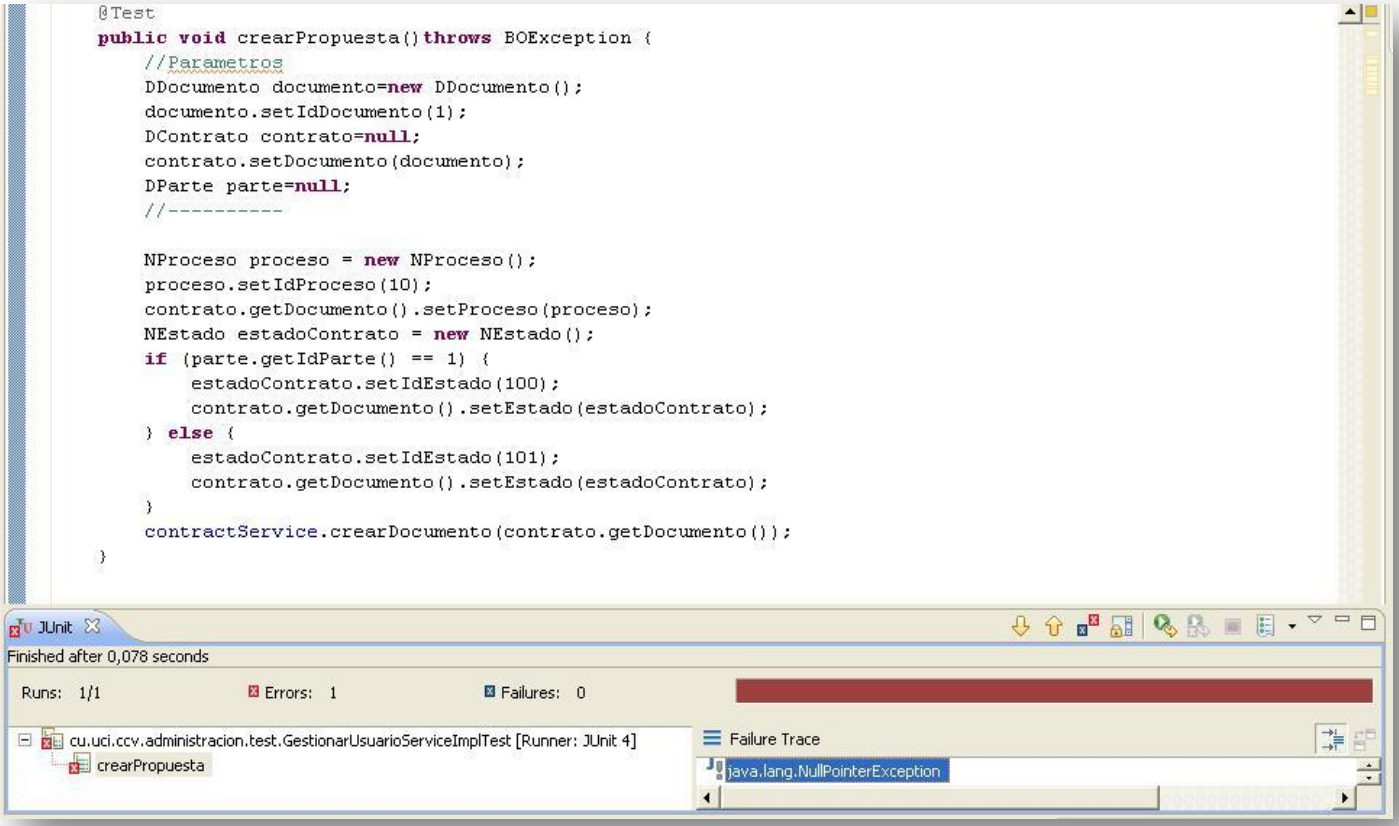


Figura 3.2: Ejecución fallida del método crearPropuesta validándose con JUnit.

También se efectuaron pruebas para cuando se ejecutan métodos del negocio, que dependen de un método de la capa de acceso a datos y este último no funciona. Seguidamente se muestra en la Figura 3.3 un ejemplo de esta prueba al ejecutar el método cargarCronogramasPorProcesos() el cual llama al método del dao sin este último estar implementado, no obstante, también se obtuvo el resultado esperado al lanzarse una excepción:

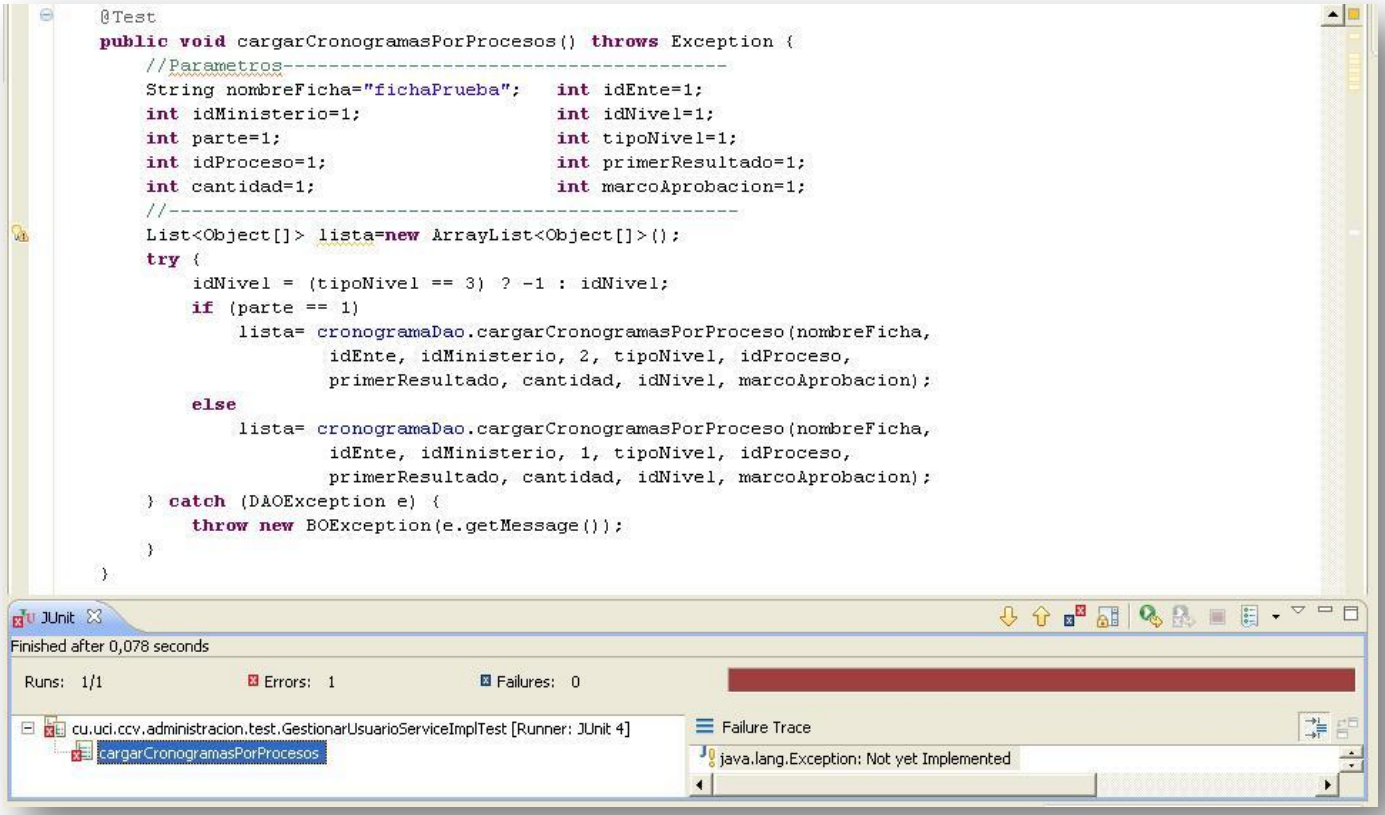


Figura 3.3: Ejecución fallida del método cargarCronogramasPorProcesos validándose con JUnit.

3.3.2 Módulo Administración-Configuración

La Figura 3.4 muestra el ejemplo del método generarPassword, el cual al no recibir parámetros esta operación generalmente va a resultar exitosa:

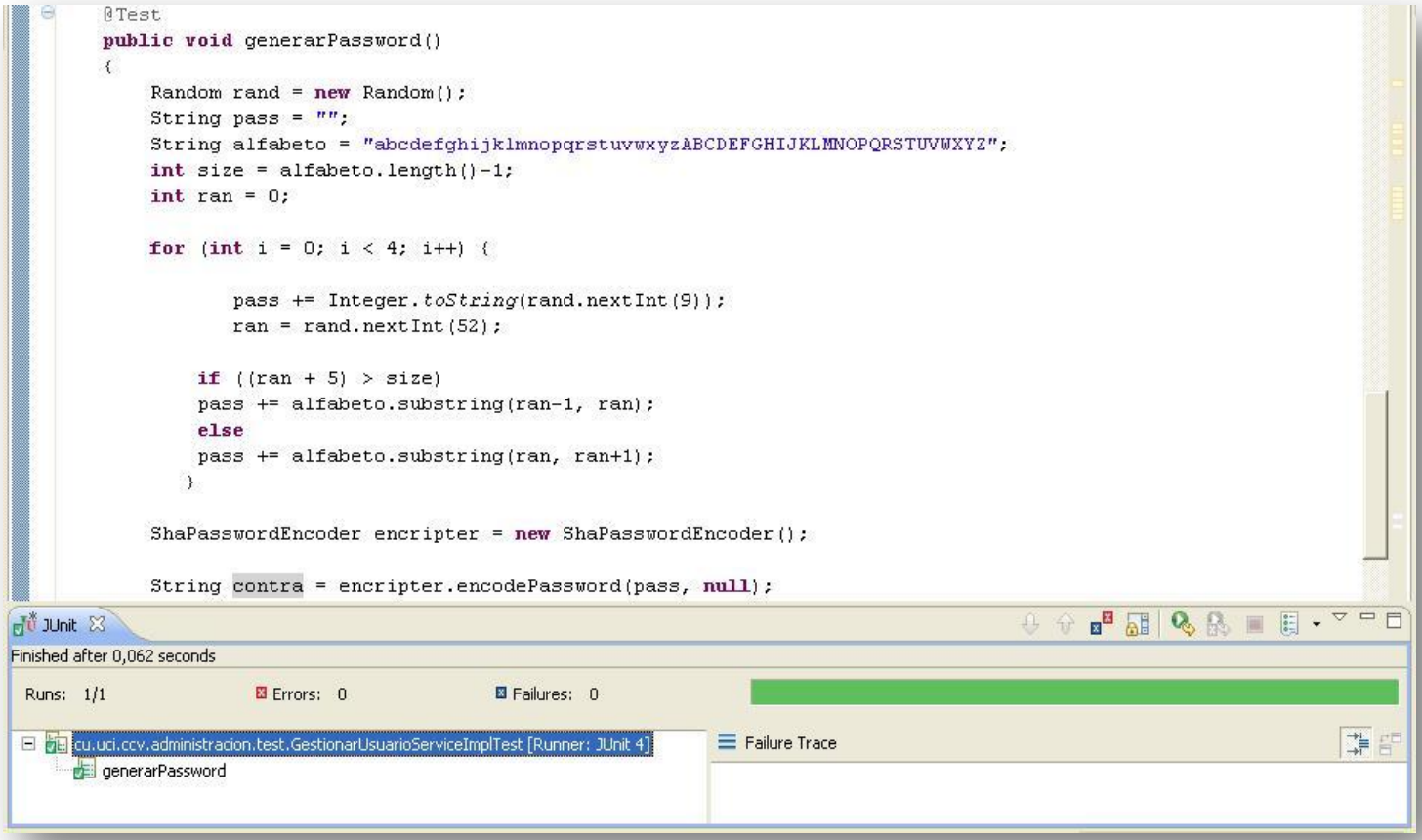


Figura 3.4: Ejecución exitosa del método generarPassword validándose con JUnit.

Existen otros casos como el que se muestra en la Figura 3.5, en el que el método resulta fallido, pues al no estar configurada la base de datos correctamente, cuando se invocan las operaciones del acceso a datos no se ejecuta correctamente:

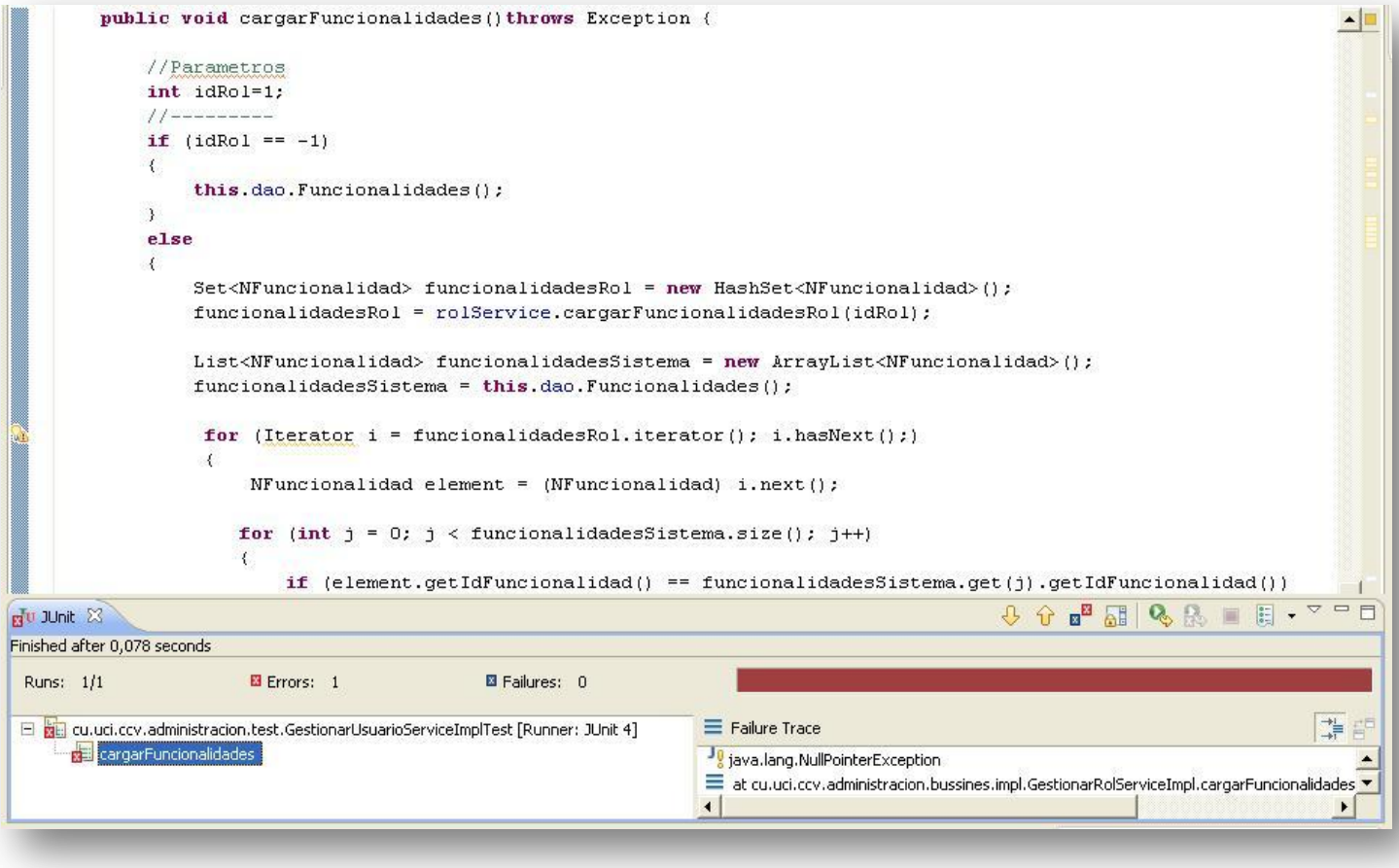


Figura 3.5: Ejecución fallida del método cargarFuncionalidades validándose con JUnit.

Seguidamente se muestra el ejemplo de un método, a los cuales primero se les pasan parámetros de clases válidas (Figura 3.6), como las definidas en los casos de prueba, y después en una segunda prueba se le pasan los parámetros de clases inválidas (Figura 3.7), en este caso **null**, obteniendo en ambos casos los resultados esperados:

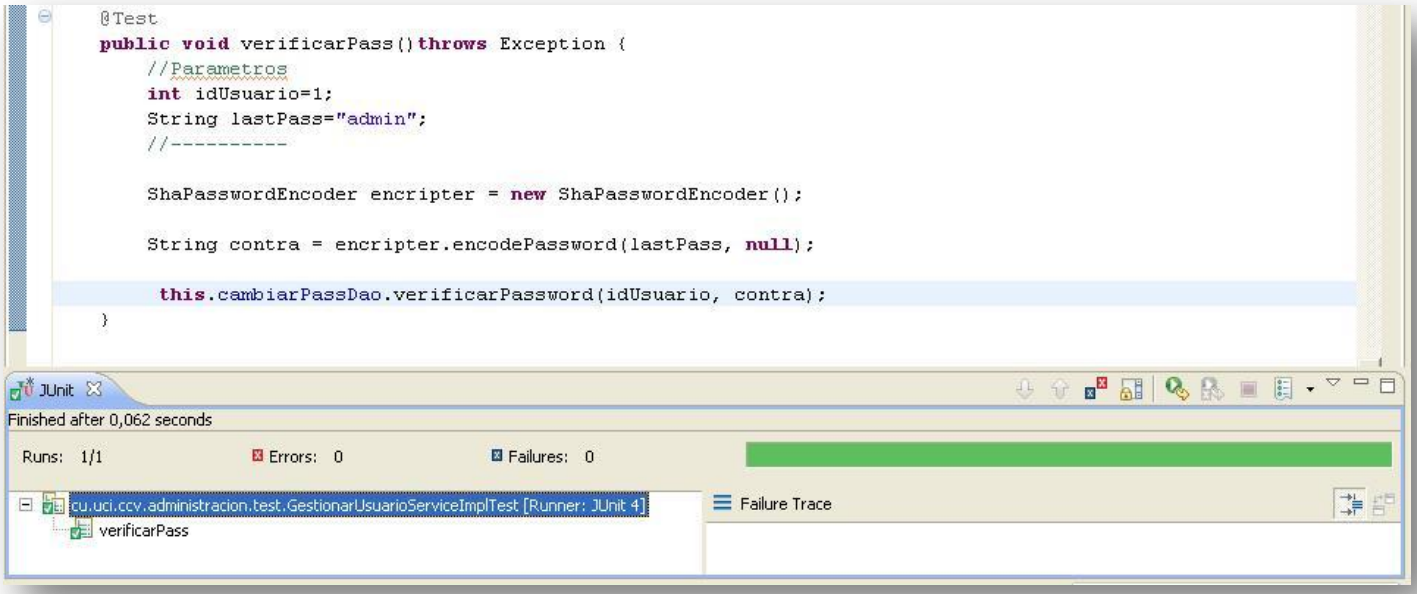


Figura 3.6: Ejecución exitosa del método verificarPass validándose con JUnit.

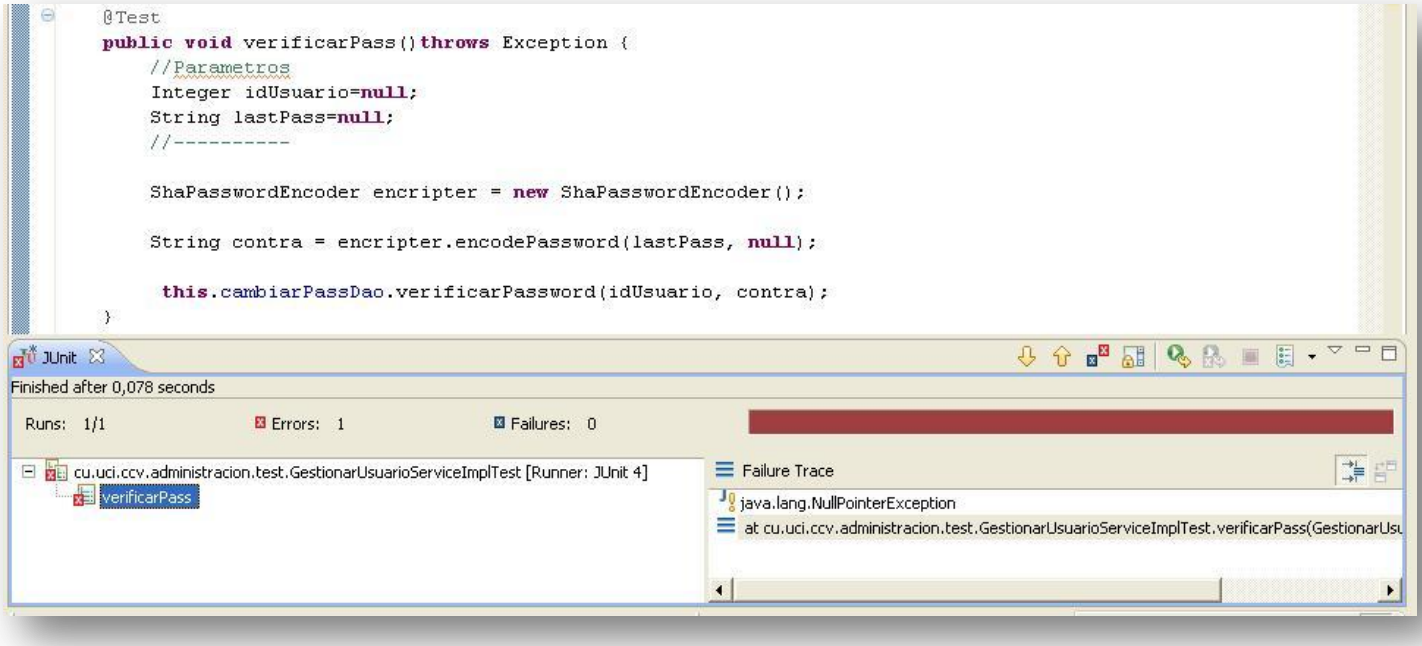


Figura 3.7: Ejecución fallida del método verificarPass validándose con JUnit.

3. 4 Conclusiones del capítulo

En este capítulo se realizó una evaluación de la solución propuesta en el capítulo anterior, para esto se llevaron a cabo pruebas unitarias mediante el framework JUnit y guiadas por casos de pruebas, con miras a validar la implementación de los casos de uso de la capa lógica de negocio en los módulos de Contratación y Administración-Configuración.

La realización de estas validaciones vislumbró la robustez y solidez de la implementación de la capa lógica de negocio en estos módulos, permitiendo que se integre a las restantes capas de presentación y acceso a datos con un alto grado de confiabilidad.

Conclusiones Generales

Al culminar el presente trabajo se ha dado cumplimiento a los objetivos y tareas que se propusieron:

- Se llevó a cabo un estudio del estado del arte de los Sistemas de Gestión de Proyectos, profundizando en la lógica de negocio, brindando esta investigación una visión más clara de los procesos de negocio que se llevan a cabo en los SGP.

- Se adoptó un estándar de codificación para la elaboración del sistema CCV.

- Se utilizaron patrones de diseño enfocados a J2EE garantizando que la solución cumpliera con los estándares de Java y de la plataforma J2EE.

- Se implementó un conjunto de clases reutilizables que automatizaron los procesos del negocio requeridos, facilitaron el mantenimiento del sistema y proporcionaron la adecuada integración con las restantes capas de presentación y acceso a datos.

- Se validó la solución propuesta través de las pruebas de unidad obteniéndose resultados satisfactorios, la liberación por parte de Calidad UCI y la aceptación por parte del cliente.

Recomendaciones

A partir del trabajo realizado se hacen las siguientes recomendaciones:

- Llevar a cabo una programación más genérica con el fin, precisamente, de lograr un producto genérico que se pueda utilizar en otros escenarios semejantes.

- Programar las mismas funcionalidades de enviar, aceptar y rechazar los cronogramas para que sean revisados y evaluados por todos los niveles como sucede con los contratos. Es válido aclarar que en cierta ocasión esto estaba programado pero se eliminó a pedido de los clientes.

- Hacer un uso más amplio de la Programación Orientada a Aspectos (POA).

- Considerar las especificaciones tecnológicas puestas en práctica en este trabajo como plataforma o modelo para investigaciones afines.

Referencias Bibliográficas

[1] Rolando Alfredo Hernandez León, Sayda Coello González. (2002) "El Paradigma Cuantitativo de la Investigación Científica" [Citado el lunes 23 de febrero de 2009].

[2] Sescoi Iberia. "Los Sistemas de Gestión de Proyectos".
URL: http://www.myworkplan.com/fileadmin/pdf/myworkplan/WhitePaper_ERP-ShopManagment_SP1.pdf [Citado el martes 24 de febrero de 2009]

[3] Rafael Menéndez-Barzanallana Asensio "Metodologías de desarrollo de software"
URL: http://www.wikilearning.com/curso_gratis/metodologias_de_desarrollo_de_software-metodologia_de_desarrollo_de_software/3617-1 [Citado el miércoles 25 de febrero de 2009]

[4] "Programación extrema" URL: http://www.chuidiang.com/ood/metodologia/extrema.php [Citado el viernes 27 de febrero de 2009]

[5] Hista Internacional S.A. (2006) **"**Consultoría en Metodologías de Desarrollo de Software - RUP y las mejores prácticas para el desarrollo de software**"**
URL: http://www.histaintl.com/servicios/consulting/rup.php [Citado el jueves 26 de febrero de 2009]

[6] Rational Unified Process (2003) [Citado el jueves 26 de febrero de 2009]

[7] "Metodologías usadas en ingeniería del software - Paradigmas de programación y sus tipos"
URL:http://www.wikilearning.com/curso_gratis/metodologias_usadas_en_ingenieria_del_software-paradigmas_de_programacion_y_sus_tipos/3618-3 [Citado el viernes 27 de febrero de 2009]

[8] Miguel Ángel Álvarez. "Qué es la programación orientada a objetos"
URL: http://www.desarrolloweb.com/articulos/499.php [Citado el viernes 27 de febrero de 2009]

[9] "Programación orientada a objetos. Sistemas de Procesamiento de Datos"
URL: http://www.monografias.com/trabajos/objetos/objetos.shtml [Citado el viernes 27 de febrero de 2009]

[10] Fernando Asteasuain, Bernardo Ezequiel Contreras. "Programación Orientada a Aspectos"
URL: http://www.angelfire.com/ri2/aspectos/ [Citado el viernes 27 de febrero de 2009]

[11] "Lenguaje de programación. Artículo de la Enciclopedia"
URL: http://enciclopedia.us.es/index.php/Lenguaje_de_programacion [Citado el lunes 16 de febrero del 2009]

[12] “Lenguajes de programación” URL: http://www.lenguajes-de-programacion.com/lenguajes-de-programacion.shtml [Citado el lunes 16 de febrero del 2009]

[13] “Clasificación de lenguajes de programación” URL: http://www.larevistainformatica.com/clasificacion-de-los-lenguajes-de-programacion.html [Citado el lunes 16 de febrero del 2009]

[14] URL: http://qbitacora.wordpress.com/2007/09/21/clasificacion-de-lenguajes-de-programacion/ [Citado el lunes 16 de febrero del 2009]

[15] Claudia R. Ledesma, Claudia A. Alí. “Trabajo de Investigación: J2EE” [Citado el martes 17 de febrero del 2009]

[16] Gonzalo Álvarez (1997). “Características del lenguaje Java”
URL: http://www.iec.csic.es/CRIPTONOMICON/java/quesjava.html [Citado el martes 17 de febrero del 2009]

[17] “Conozca más sobre la tecnología Java” URL: http://www.java.com/es/about/ [Citado el martes 17 de febrero del 2009]

[18] Leonardo Viacava, Daniel Perovich. “Gestión de Transacciones en la Plataforma J2EE” [Citado el martes 17 de febrero del 2009]

[19] “Java 2 Platform, Enterprise Edition (J2EE) FAQ” URL:
http://java.sun.com/javaee/overview/faq/j2ee.jsp [Citado el martes 17 de febrero del 2009]

[20] “Netbeans vs Eclipse en el mundo hispano: una encuesta con 3400 participantes - Asociación javaHispano” URL: http://www.javahispano.org/contenidos/es/netbeans_vs_eclipse_en_el_mundo_hispano_una_encuesta_con_3400_participantes/ [Citado el miércoles 18 de febrero del 2009]

[21] Juan Gutiérrez (2004). “Eclipse (2.1) y Java”
URL:http://74.125.47.132/search?q=cache:v61syUfORUAJ:www.uv.es/~jgutierr/MySQL_Java/TutorialEclipe.pdf [Citado el miércoles 18 de febrero del 2009]

[22] Laura Bermejo, Enrique Gomez.”Eclipse como IDE”. [Citado el miércoles 18 de febrero del 2009]

[23] “Subcplise” URL: http://subclipse.tigris.org/ [Citado el miércoles 18 de febrero del 2009]

[24] “Hibernate Tools for Eclipse and Ant” URL: http://www.hibernate.org/255.html [Citado el jueves 19 de febrero del 2009]

[25] “Bienvenido a NetBeans y www.netbeans.org, Portal del IDE Java de Código Abierto”
URL: http://www.netbeans.org/index_es.html [Citado el jueves 19 de febrero del 2009]

[26] Craig Walls, Ryan Breidenbach. (2005). “Spring in Action”. [Citado el jueves 26 de febrero del 2009]

[27] Rod Johnson. (2008) “Spring, java/j2ee Application Framework” Version 2.5.5 [Citado el viernes 27 de febrero del 2009]

[28] “Java en castellano. Primeros Pasos con JUnit.”
URL: http://www.programacion.com/java/articulo/jap_junit/ [Citado el lunes 2 de marzo del 2009]

[29] Scott Hommel; Sun Microsystems Inc. (1999) “Convenciones de Código para el lenguaje de programación JAVA™” [Citado el miércoles 4 de marzo de 2009]

[30] “Patrones de Diseño en aplicaciones Web con Java J2EE.”
URL: http://java.ciberaula.com/articulo/diseno_patrones_j2ee/ [Citado el martes 3 de marzo del 2009]

[31] Mahmoud Qusay H., Kladko Konstantin. “Compresión y descompresión de datos utilizando Java”
URL:http://developer.java.sun.com/developer/technicalArticles/Programming/compression/ [Citado el martes 17 de marzo de 2009].

[32] Jorge Riquelme Santana. “Comprimir/Descomprimir en Java”
URL: http://www.totex.cl/news/sw-dev [Citado el martes 17 de marzo de 2009].

Anexos

Anexo 1. Acta de Finiquito

ACTA DE FINIQUITO

Entre la República Bolivariana de Venezuela, actuando por órgano del Ministerio del Poder Popular para la Energía y Petróleo de la República Bolivariana de Venezuela, representado en este acto por el ciudadano **Ammar Jabour**, venezolano, mayor de edad, de este domicilio, titular de la Cédula de Identidad No. 9962729, quien actúa en su condición de Coordinador General del Convenio Integral de Cooperación Cuba-Venezuela, suficientemente facultado para este acto y debidamente designado para ejercer tal condición conforme a lo establecido en **CONTRATO E08-001-000 SOLUCIÓN TECNOLÓGICA INTEGRAL PARA EL DESARROLLO DEL SISTEMA DE GESTIÓN PARA SEGUIMIENTO DE LOS PROYECTOS DEL CONVENIO INTEGRAL DE COOPERACIÓN CUBA VENEZUELA,** que en lo sucesivo se denominará la "**Parte Venezolana**", por una parte; y por la otra, la sociedad mercantil **ALBET, Ingeniería y Sistemas, S.A.**, sociedad mercantil cubana constituida mediante Escritura 271 de fecha 7 de Noviembre de 2005, autorizada por la Notario Lic. Isabel Cristina Martínez Alfonso con sede en Notaría Especial del Ministerio de Justicia de Ciudad de la Habana, inscrita en el Registro Mercantil de esta ciudad con fecha 14 de Noviembre del año 2005, al Tomo XVIII, Folio 120, Hoja 11, Sección SM, con Nº de inscripción 1 con domicilio social en Centro de Negocios de Miramar, Edificio Barcelona, Oficina 322, Avenida 5ta entre 76 y 78, Miramar, Municipio Playa, Ciudad de La Habana, República de Cuba, representada en este acto por el ciudadano cubano **Ibrahím Nápoles Albanés**, mayor de edad, portador de carné de identidad Nº 62032504808 en su condición de Coordinador General, suficientemente facultado para este acto según lo dispuesto en **CONTRATO E08-001-000 SOLUCIÓN TECNOLÓGICA INTEGRAL PARA EL DESARROLLO DEL SISTEMA DE GESTIÓN PARA SEGUIMIENTO DE LOS PROYECTOS DEL CONVENIO INTEGRAL DE COOPERACIÓN CUBA VENEZUELA**, que en lo sucesivo se denominará **"Parte Cubana",** al tenor de las siguientes declaraciones y cláusulas:

CONSIDERANDO

Primero: Consta de documento privado suscrito en fecha 18 de marzo de 2008, que ambas partes celebraron un **CONTRATO DE SOLUCIÓN TECNOLÓGICA INTEGRAL PARA EL DESARROLLO DEL SISTEMA DE GESTIÓN PARA SEGUIMIENTO DE LOS PROYECTOS DEL CONVENIO INTEGRAL DE COOPERACIÓN CUBA VENEZUELA,** el cual fue celebrado al Amparo del Convenio Integral de Cooperación, suscrito el 30 de octubre del 2000 y su Addendum; de la Declaración Conjunta y el Acuerdo suscrito entre ambas Repúblicas, para la aplicación de la Alternativa Bolivariana para las Américas, firmados en diciembre del 2004 y de los Acuerdos y las Condiciones Generales firmados en el Acta de la Reunión de la Comisión Mixta Cuba-Venezuela, celebrada en La Habana, Cuba, cuyos contenidos se dan aquí enteramente por reproducidos.

Segundo: Consta que la **Parte Cubana** ha cumplido a entera satisfacción de la **Parte Venezolana** con el objeto, alcance y actividades previstas en el **Contrato** ya mencionado, así como con sus Anexos y Suplementos, cumpliendo por tanto con todos sus deberes y obligaciones por lo que se considera que la Solución ha sido implementada en las condiciones previstas y bajos los requisitos y especificaciones técnicas pactadas entres **Las Partes**.

Tercero: Consta que la **Parte Cubana** ha hecho entrega del **Informe Técnico Final**, de conjunto a toda la documentación que avala y soporta lo descrito en el Segundo de los **CONSIDERANDOS**, debidamente firmada y aceptada por los especialistas de la **Parte Venezolana**.

LAS PARTES CONVIENEN:

Primero: Dar por terminada la relación contractual derivada del **CONTRATO DE SOLUCIÓN TECNOLÓGICA INTEGRAL PARA EL DESARROLLO DEL SISTEMA DE GESTIÓN PARA SEGUIMIENTO DE LOS PROYECTOS DEL CONVENIO INTEGRAL DE COOPERACIÓN CUBA VENEZUELA** de fecha 16 de marzo de 2008.

Segundo: Las Partes se otorgan el finiquito más amplio que en derecho proceda, no reservándose acción o derecho que ejercitar con posterioridad.

Leída que les fue la presenta Acta, las partes se ratificaron en su contenido, para constancia firman en cuatro (4) ejemplares de igual tenor en la Ciudad de Caracas el día 17 del mes de diciembre del 2008.

Por la PARTE VENEZOLANA	Por la PARTE CUBANA
Ammar Jabour Coordinador General	**Ibrahim Nápoles Albanés** Coordinador General

Acta.1 Acta de Finiquito

Anexo 2. Acta de Aceptación

Acta de Aceptación

albet INGENIERÍA Y SISTEMAS

Producto: Sistema de Gestión para el seguimiento de los Proyectos del Convenio Integral de Cooperación Cuba Venezuela (CCV)

Categoría de las pruebas: Revisión de Módulos Presentación, Administración y Configuración del CCV versión 1.0.

Fecha de conciliación: 20 de agosto de 2008

Involucrados en el proceso:

- **Por la parte del Cliente (CCV):** Sandra Cortés
- **Por la parte desarrolladora (ALBET):** Ing. Nahuel Massón Padilla
- **Observador independiente (CALISOFT):** Ing. Dayami Rodríguez Brito
- **Observador independiente (CALISOFT):** Ing. Roig Calzadilla Diaz

Observaciones del proceso:

Durante el proceso de pruebas de aceptación se identificaron un conjunto de cambios y mejoras necesarias que quedaron registrados adecuadamente en el correspondiente documento de Respuestas a las No Conformidades detectadas, con sus respectivas observaciones. Teniendo en cuenta que las No Conformidades han sido debidamente respondidas, ejecutadas por el Equipo de Desarrollo y validada la eficacia de los mismos por los clientes, se ha tomado el acuerdo de aceptar el **"Sistema de Gestión para el seguimiento de los Proyectos del Convenio Integral de Cooperación Cuba Venezuela"** para la versión 1.0, con fecha 20 de agosto de 2008, el cual se anexa a esta Acta, estableciendo de esta forma la condición necesaria y suficiente para su despliegue.

Para que conste la aceptación de los resultados de las pruebas y por tanto la aceptación **del "Sistema de Gestión para el seguimiento de los Proyectos del Convenio Integral de Cooperación Cuba Venezuela",** dando fe del acuerdo firman la presente los principales representantes de las partes.

Sandra Cortés
(Por la parte del Cliente)

Ing. Nahuel Massón Padilla
(Por la parte Cubana - ALBET)

Ing. Roig Calzadilla Diaz
(Observador IndependienteCALISOFT)

Acta.2 Acta de Aceptación

Anexo 3. Acta de Aceptación. CU Módulo de Contratación

albet INGENIERÍA Y SISTEMAS | PROYECTOS

ACTA DE ACEPTACIÓN

Producto: Sistema de Gestión para Seguimiento de los Proyectos del Convenio Integral de Cooperación Cuba-Venezuela.

Categoría: Aceptación CU Módulo de Contratación.

Fecha de la conciliación: 3/07/2008.

Involucrados en el proceso:

Por la parte del Cliente (MENPET): Sandra Cortés
Por la parte del Suministrador (ALBET): Ing. Nahuel Massón

Observaciones del proceso:

Por acuerdo entre las partes involucradas en el proceso se Acepta el Documento CU del Módulo de Contratación de la versión v.2.0 del Sistema de Gestión para Seguimiento de los Proyectos del Convenio Integral de Cooperación Cuba-Venezuela, con fecha 3 de julio de 2008.

Para que conste la aceptación de la descripción de los CU y requerimientos del Módulo de Contratación, dando fé al acuerdo, firman la presente los principales representantes de las Partes.

Sandra Cortés
Representante MENPET

Ing. Nahuel Massón
Representante ALBET

Referencia: CV-SW-OC-011

ALBET, S.A.
Centro de Negocios Miramar, Edificio Barcelona, Oficina 302. Avenida 5ta e/ 76 y 78. Miramar. Playa, Ciudad Habana, Cuba
Tel/Fax: +53 (7) 837 2407
E-mail: albet@albet.cu

Acta.3 Acta de Aceptación. CU Módulo de Contratación.

Anexo 4. Acta de Aceptación. CU Módulo de Administración

albet INGENIERÍA Y SISTEMAS | PROYECTOS

ACTA DE ACEPTACIÓN

Producto: Sistema de Gestión para Seguimiento de los Proyectos del Convenio Integral de Cooperación Cuba-Venezuela.

Categoría: Aceptación CU Módulo de Administración.

Fecha de la conciliación: 28/07/2008

Involucrados en el proceso:

Por la parte del Cliente (MENPET): Sandra Cortés
Por la parte del Suministrador (ALBET): Ing. Nahuel Massón

Observaciones del proceso:

Por acuerdo entre las partes involucradas en el proceso se Acepta el Documento CU del Módulo de Administración de la versión v.1.0 del Sistema de Gestión para Seguimiento de los Proyectos del Convenio Integral de Cooperación Cuba-Venezuela, con fecha 28 de julio de 2008.

Para que conste la aceptación de la descripción de los CU y requerimientos del Módulo de Administración, dando fe al acuerdo, firman la presente los principales representantes de las Partes.

Sandra Cortés
Representante MENPET

Ing. Nahuel Massón
Representante ALBET

Referencia: CV-SW-CC-006

ALBET, S.A.
Centro de Negocios Miramar, Edificio Barcelona, Oficina 322, Avenida 5ta e/ 76 y 78, Miramar, Playa, Ciudad Habana, Cuba
Tel/Fax: +53 (7) 837 2407
E-mail: albet@albet.cu

Acta.4 Acta de Aceptación. CU Módulo de Administración.

Anexo 5. Acta de Aceptación. CU Módulo de Configuración

albet INGENIERÍA Y SISTEMAS | PROYECTOS

ACTA DE ACEPTACIÓN

Producto: Sistema de Gestión para Seguimiento de los Proyectos del Convenio Integral de Cooperación Cuba-Venezuela.

Categoría: Aceptación CU Módulo de Configuración.

Fecha de la conciliación: 28/07/2008

Involucrados en el proceso:

Por la parte del Cliente (MENPET): Sandra Cortés
Por la parte del Suministrador (ALBET): Ing. Nahuel Massón

Observaciones del proceso:

Por acuerdo entre las partes involucradas en el proceso se Acepta el Documento CU del Módulo de Configuración de la versión v.1.0 del Sistema de Gestión para Seguimiento de los Proyectos del Convenio Integral de Cooperación Cuba-Venezuela, con fecha 28 de julio de 2008.

Para que conste la aceptación de la descripción de los CU y requerimientos del Módulo de Configuración, dando fe al acuerdo, firman la presente los principales representantes de las Partes.

Sandra Cortés
Representante MENPET

Ing. Nahuel Massón
Representante ALBET

Referencia: CV-SW-CC-010

ALBET, S.A.
Centro de Negocios Miramar, Edificio Barcelona, Oficina 322, Avenida 5ta e/ 76 y 78, Miramar, Playa, Ciudad Habana, Cuba
Tel/Fax: +53 (7) 837 2407
E-mail: albet@albet.cu

Acta.5 Acta de Aceptación. CU Módulo de Configuración.

Anexo 6. Descripción de las clases de negocio

Las siguientes tablas describen el resto de las clases dentro del paquete **bussines/impl** que implementan a las interfaces referentes a los módulos de Contratación y Administración-Configuración.

Módulo Contratación

CU Aprobar Contrato a Nivel de Ministerio

Nombre de la Clase: AprobarContratoMServiceImpl	
Atributos	
Nombre:	**Tipo:**
contractService	ContractService
Responsabilidades	
Nombre:	**Descripción:**
aceptarContrato(DContrato contrato, NTipo_Nivel tipoNivel, DParte parte): void	Este método llama al método procesarOperación() pasándole 1 con el objetivo de aceptar el envío del contrato.
rechazarContrato(DContrato contrato, NTipo_Nivel tipoNivel, DRechazo rechazo, DParte parte): void	Llama al método procesarOperación() pasándole 2 con el objetivo de rechazar el envío del contrato al mismo tiempo que crea un rechazo para dar su explicación.
enviarContrato(DContrato contrato, NTipo_Nivel tipoNivel, DParte parte): void	Llama al método procesarOperación() pasándole 7 con el objetivo de enviar el contrato hacia el nivel superior .
aceptarEnvio(DContrato contrato, NTipo_Nivel tipoNivel, DParte parte): void	Llama al método procesarOperación() pasándole 1 con el objetivo de aceptar el envío del contrato.
rechazarEnvio(DContrato contrato, NTipo_Nivel tipoNivel, DRechazo rechazo, DParte parte): void	Llama al método procesarOperación() pasándole 2 con el objetivo de rechazar el envio del contrato al mismo tiempo que crea un rechazo con la

	explicación pertinente.
modificarContrato(DContrato contrato, NTipo_Nivel tipoNivel, DParte parte): void	Modifica el contrato llamando al procesarOperación() creando un nuevo contrato con los datos ya modificados y el contrato del que se partió se pone inactivo y de anterior del nuevo.
aceptarModificacion(DContrato contrato, NTipo_Nivel tipoNivel, DParte parte): void	Acepta la modificación del contrato borrando las fichas y el contrato anterior, además de poner en nulo el atributo referenciaId de las fichas contenidas por el nuevo contrato.
rechazarModificacion(DContrato contrato, NTipo_Nivel tipoNivel,DRechazo rechazo, DParte parte): void	Rechaza la modificación del contrato actualizándolo con los atributos que tenía el anterior y creando el rechazo con la explicación.
procesarOperacion(DContrato contrato, NTipo_Nivel tipoNivel, int operacion, DParte parte): void	Procesa las operaciones llamando al método Siguiente Flujo implementado en la clase padre ProcesoServiceImpl el cual le cambia el estado y el proceso a los documentos.

Tabla a.1: Descripción de la clase AprobarContratoMServiceImpl.

CU Aprobar Contrato a Nivel de Secretaría

Nombre de la Clase: AprobarContratoSTServiceImpl	
Atributos	
Nombre:	**Tipo:**
contractService	ContractService
Responsabilidades	
Nombre:	**Descripción:**
aceptarContrato(DContrato contrato, NTipo_Nivel tipoNivel, DParte parte): void	Este método llama al método procesarOperación() pasándole 1 con el objetivo de aceptar el envío del contrato.
rechazarContrato(DContrato contrato,	Llama al método procesarOperación() pasándole 2

NTipo_Nivel tipoNivel, DRechazo rechazo, DParte parte): void	con el objetivo de rechazar el envío del contrato al mismo tiempo que crea un rechazo para dar su explicación.
procesarOperacion(DContrato contrato, NTipo_Nivel tipoNivel, int operacion, DParte parte): void	Procesa las operaciones llamando al método Siguiente Flujo implementado en la clase padre ProcesoServiceImpl el cual le cambia el estado y el proceso a los documentos.

Tabla a.2: Descripción de la clase AprobarContratoSTServiceImpl.

CU Obtener Reportes

Nombre de la Clase: ReporteContratacionServiceImpl	
Atributos	
Nombre:	**Tipo:**
contratoDao	ContratoDao
nomenclador	NomencladorService
Responsabilidades	
Nombre:	**Descripción:**
reporteFichasContratadas(Integer idMinisterio, Integer idEnte, Integer idFicha, DParte parte): List<Object[]>	Este método devuelve una lista de objetos donde cada uno de ellos es un atributo de las fichas contratadas necesario para realizar los reportes a nivel de ente, ministerio y secretaría.
reporteFichasNoContratadas(Integer idMinisterio, Integer idEnte, Integer idFicha, DParte parte): List<Object[]>	Este método devuelve una lista de objetos donde cada uno de ellos es un atributo, en este caso de las fichas no contratadas, necesario para realizar los reportes a nivel de ente, ministerio y secretaría.

Tabla a.3: Descripción de la clase ReporteContratacionServiceImpl.

CU Firmar Contrato

Nombre de la Clase: FirmarContratoServiceImpl

Atributos	
Nombre:	**Tipo:**
contratoDao	ContratoDao
contractService	ContractService
cronogramService	CronogramService
cronogramaDao	CronogramaDao
fichaService	FichaService
Responsabilidades	
Nombre:	**Descripción:**
firmarContrato(DContrato contrato, NTipo_Nivel tipoNivel,DParte parte): void	Este método elimina todos lo rechazos anteriores del contrato, llama directamente al método siguiente flujo para que se modifique su estado y su proceso y por último se modifica o actualiza en la base de datos.
estadoCronogramaParaFirmarContrato (int idContrato): boolean	Devuelve verdadero o falso en caso de que los cronogramas que contiene un contrato estén o no en el estado indicado para que le contrato pueda ser firmado.
eliminarRechazosDocumentos(DContrato contrato): void	Elimina todos los rechazos pertenecientes a un contrato determinado.

Tabla a.4: Descripción de la clase FirmarContratoServiceImpl.

Módulo Administración-Configuración

CU Configurar Tipos de Recurso Humano

Nombre de la Clase: TipoRecursoHumanoServiceImpl	
Atributos	
Nombre:	**Tipo:**
tipoRecursoHumanoDao	ConfigurarTipoRecursoHumanoDao
honorarioDao	ConfigurarHonorariosDao
Responsabilidades	
Nombre:	**Descripción:**
buscarTiposRecursoHumano():List<NCategoria>	Devuelve una lista de categorías de recursos humanos.
crearTipoRecursoHumano(String nombre, String descripcion, String desde, String hasta): void	Crea una nueva categoría de recurso humano con el nombre, descripción, y los valores desde y hasta de su tasa de honorario.
modificarTipoRecursoHumano(int TipoRecurso, int idTasaHonorario, String nombre,String descripcion, String desde, String hasta): void	Modifica una categoría de recurso humano filtrando por su identificador de tipo de recurso humano y por el de su tasa de honorario, modificando el nombre, descripcion y los valores desde y hasta de su tasa de honorario.
eliminarTipoRecursoHumano(int idTipoRecurso): void	Elimina una categoría de recurso humano filtrando por su identificador de tipo de recurso humano.

Tabla a.5: Descripción de la clase TipoRecursoHumanoServiceImpl.

CU Configurar Fuentes de Financiamiento

Nombre de la Clase: FuenteFinanciamientoServiceImpl

Atributos	
Nombre:	**Tipo:**
fuenteFinanciamientoDao	ConfigurarFuenteFinanciamientoDao
Responsabilidades	
Nombre:	**Descripción:**
buscarFuenteFinanciamiento(): List<NFuente_Financiamiento>	Devuelve una lista de las fuentes de financiamiento.
crearFuenteFinanciamiento(String nombre, String descripcion): void	Crea una nueva fuente de financiamiento con el nombre y descripción.
modificarFuenteFinanciamiento(int idFuenteFinanciamiento,String nombre, String descripcion): void	Modifica una fuente de financiamiento filtrando por su identificador, modificando el nombre y la descripcion.
eliminarFuenteFinanciamiento(int idFuenteFinanciamiento): void	Elimina una fuente de financiamiento filtrando por su identificador.

Tabla a.6: Descripción de la clase FuenteFinanciamientoServiceImpl.

CU Configurar Viáticos

Nombre de la Clase: ViaticoServiceImpl	
Atributos	
Nombre:	**Tipo:**
viaticoDao	ConfigurarViaticoDao
Responsabilidades	
Nombre:	**Descripción:**
buscarViatico():List<NConfiguracion_Viaticos>	Devuelve una lista de los viáticos.
crearViatico(BigDecimal monto, int desde, int	Crea un nuevo viático con el monto, el día desde,

hasta, int pparte): void	el día hasta y a la Parte a la que le corresponde.
modificarViatico(int idViatico, BigDecimal monto, int desde, int hasta, int pparte): void	Modifica un viático filtrando por su identificador, modificando el monto, el día desde, el día hasta y a la Parte a la que le corresponde.
eliminarViatico(int idViatico): void	Elimina un viático filtrando por su identificador.

Tabla a.7: Descripción de la clase ViaticoServiceImpl.

CU Configurar Modalidad de Proyecto

Nombre de la Clase: ModalidadProyectoServiceImpl	
Atributos	
Nombre:	**Tipo:**
modalidadProyectoDao	ConfigurarModalidadProyectoDao
Responsabilidades	
Nombre:	**Descripción:**
buscarModalidadProyecto():List<NModalidad>	Devuelve una lista de las modalidades de proyecto.
crearModalidadProyecto(String nombre, String descripcion): void	Crea una nueva modalidad de proyecto con el nombre y la descripción.
modificarModalidadProyecto(int idModalidadProyecto, String nombre, String descripcion): void	Modifica una modalidad de proyecto filtrando por su identificador, modificando el nombre y la descripción.
eliminarModalidadProyecto(int idModalidadProyecto): void	Elimina una modalidad de proyecto filtrando por su identificador.

Tabla a.8: Descripción de la clase ModalidadProyectoServiceImpl.

CU Configurar Tipo de Recurso

Nombre de la Clase: TipoRecursoServiceImpl	
Atributos	
Nombre:	**Tipo:**
tipoRecursoDao	ConfigurarTipoRecursoDao
Responsabilidades	
Nombre:	**Descripción:**
buscarTiposRecurso():List<NTipo_Recurso>	Devuelve una lista de los tipos de recursos.
crearTipoRecurso(String nombre, String descripcion): void	Crea un nuevo tipo de recurso con el nombre y la descripción.
modificarTipoRecurso(int idTipoRecurso, String nombre, String descripcion): void	Modifica un tipo de recurso filtrando por su identificador, modificando el nombre y la descripción.
eliminarTipoRecurso(int idTipoRecurso): void	Elimina un tipo de recurso filtrando por su identificador.

Tabla a.9: Descripción de la clase TipoRecursoServiceImpl.

CU Gestionar Datos de Entes Ejecutores

Nombre de la Clase: GestionarEnteServiceImpl	
Atributos	
Nombre:	**Tipo:**
enteDao	GestionarEnteDao
Responsabilidades	
Nombre:	**Descripción:**

buscarMinisterios(int idParte): List<DNivel>	Devuelve una lista de ministerios dado el identificador de una Parte.
buscarEntes(int idParte, String siglas, String nombreEnte, int primero, int cantidad, int idMinisterio): List<DNivel>	Devuelve una lista de entes con paginado, filtrando por determinados parámetros como el identificador de la Parte, las siglas, el nombre del ente y el identificador del ministerio.
crearEnte(DNivel nivel): void	Crea un nuevo ente.
modificarEnte(DNivel nivel): void	Modifica un ente.
eliminarEnte(DNivel nivel): boolean	Elimina un ente, devuelve **true** si puede ser eliminado y **false** en caso contrario.
cantidadEntes(int idParte, String siglas, String nombreEnte, int idMinisterio): int	Devuelve la cantidad de entes que tiene un determinado ministerio de una determinada Parte.

Tabla a.10: Descripción de la clase GestionarEnteServiceImpl.

CU Modificar Contraseña

Nombre de la Clase: CambiarPasswordServiceImpl	
Atributos	
Nombre:	**Tipo:**
cambiarPassDao	CambiarPasswordDao
Responsabilidades	
Nombre:	**Descripción:**
cambiarPassword(int idUsuario, String pass): void	Cambia el password de un determinado usuario.
verificarPass(int idUsuario, String lastPass): boolean	Verifica si un determinado usuario tiene un determinado password, devuelve **true** en caso de que coincida con ese password y **false** en caso contrario.

encriptarPassword(String password): String	Encripta una cadena de caracteres pasada por parámetro y la devuelve.

Tabla a.11: Descripción de la clase CambiarPasswordServiceImpl.

CU Gestionar Datos de Ministerios

Nombre de la Clase: GestionarSectorServiceImpl	
Atributos	
Nombre:	**Tipo:**
gestionarSectorDao	GestionarSectorDao
Responsabilidades	
Nombre:	**Descripción:**
buscarSectores(int idparte): List<NSector>	Devuelve una lista de los sectores dado el identificador de una Parte.
crearSector(NSector sector): void	Crea un nuevo sector.
modificarSector(NSector sector): void	Modifica un sector.
eliminarSector(NSector sector): void	Elimina un sector.

Tabla a.12: Descripción de la clase GestionarSectorServiceImpl.

CU Configurar Tasa de Cambio

Nombre de la Clase: TazaCambioServiceImpl	
Atributos	
Nombre:	**Tipo:**
tazaCambioDao	ConfigurarTazaCambioDao

Responsabilidades	
Nombre:	**Descripción:**
buscarTazaCambio():double	Devuelve la tasa de cambio.
crearTazaCambio(String valor): void	Crea una taza de cambio con un determinado valor.

Tabla a.13: Descripción de la clase TazaCambioServiceImpl.

CU Configurar Honorarios

Nombre de la Clase: HonorarioServiceImpl	
Atributos	
Nombre:	**Tipo:**
honorarioDao	ConfigurarHonorariosDao
Responsabilidades	
Nombre:	**Descripción:**
buscarHonorarios():List<NCategoria>	Devuelve una lista de las categorías de los tipos de recursos humanos.
modificarHonorario(List<NCategoria> honorarios): void	Modifica los honorarios dada una lista de categorías.

Tabla a.14: Descripción de la clase HonorarioServiceImpl.

Trazas

Esta clase es útil para obtener las trazas de las operaciones que realizan los usuarios, no pertenece a ningún caso de uso en particular.

Nombre de la Clase: TrazasServiceImpl

Atributos	
Nombre:	**Tipo:**
trazasDao	TrazasDao
Responsabilidades	
Nombre:	**Descripción:**
buscarTrazas(String usuario, String persona, Date desde, Date hasta, boolean error, int primero, int cantidad): List<DTrazas>	Devuelve una lista de trazas con paginado filtrando por usuario, nombre de la persona, fecha desde, fecha hasta y si son trazas de error.
totalTrazas(String usuario, String persona, Date desde, Date hasta, boolean error): int	Devuelve un total de trazas que posee un usuario, filtrando por nombre de la persona, fecha desde, fecha hasta y si son trazas de error.

Tabla a.15: Descripción de la clase TrazasServiceImpl.

Glosario de Términos

Capa de Negocio o Capa Lógica de Negocio: La programación por capas es un estilo de programación en el que el objetivo primordial es la separación de la lógica de negocios de la lógica de diseño, un ejemplo básico de esto consiste en separar la capa de datos de la capa de presentación al usuario. La ventaja principal de este estilo es que el desarrollo se puede llevar a cabo en varios niveles y, en caso de que sobrevenga algún cambio, sólo se ataca al nivel requerido sin tener que revisar entre código mezclado, pues esto permite distribuir el trabajo de creación de una aplicación por niveles, de este modo, cada grupo de trabajo está totalmente abstraido del resto de los niveles. Específicamente, en esta capa es donde residen los programas que se ejecutan, se reciben las peticiones del usuario y se envían las respuestas tras el proceso. Se denomina Capa de Negocio (e incluso de Lógica del Negocio) porque es aquí donde se establecen todas las reglas que deben cumplirse. Esta capa se comunica con la Capa de Presentación, para recibir las solicitudes y presentar los resultados, y con la Capa de Acceso a Datos, para solicitar al gestor de base de datos con el objetivo de almacenar o recuperar datos de él. También se consideran aquí los programas de aplicación. El tener una arquitectura n-capas bien definida reducirá los hoyos de vulnerabilidad que evidentemente se dan cuando no desarrollamos con esta arquitectura. Dentro de esta arquitectura la capa intermedia o la capa lógica de negocios ocupa un lugar cimero en la construcción de una infraestructura de software que permitirá el crecimiento y la extensión de servicios para todas las aplicaciones existentes y futuras.

Framework: Es una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, un *Framework* puede incluir soporte de programas, bibliotecas y un lenguaje interpretado, entre otros, software para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

Framework Spring: Es un Framework de código abierto de desarrollo de aplicaciones para la plataforma Java. A pesar de que el Framework Spring no obliga a usar un modelo de programación en particular, se ha popularizado en la comunidad de programadores en Java al considerársele una alternativa y sustituto del modelo de Enterprise JavaBean. Por su diseño ofrece mucha libertad a los desarrolladores en Java y soluciones muy bien documentadas y fáciles de usar para las prácticas comunes en la industria.

Open Source: Significa código abierto (en español) y es el término con el que se conoce al software distribuido y desarrollado libremente.

Estándar de Codificación: (también llamado *estándares de código* o *convención de código*) es un término que describe convenciones para escribir código fuente en ciertos lenguajes de programación. Es frecuentemente dependiente del lenguaje de programación que se haya elegido para escribir. Por ejemplo: el estilo del lenguaje de programación C variará con respecto al del lenguaje BASIC.

Patrón de diseño: Son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces. Es una solución a un problema de diseño. Para que una solución sea considerada un patrón debe poseer ciertas características:

1. Debe haber comprobado su efectividad resolviendo problemas similares en ocasiones anteriores.
2. Debe ser reusable, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias.

Pruebas de unidad: En informática, es una forma de probar el correcto funcionamiento de un módulo de código. Sirve para asegurar que cada uno de los módulos funcione correctamente por separado.

Action: Acción, evento lanzado en el Framework Spring y en la Programación Orientada a Objetos, en general, para que un sistema cumpla con determinadas funcionalidades.

Servlet: Es un objeto que corre dentro del contexto de un contenedor de servlets (ej: Tomcat) y extiende su funcionalidad. También podría correr dentro de un servidor de aplicaciones (ej: OC4J Oracle), que, además de contenedor para servlet, tendrá contenedor para objetos más avanzados, como son los EJB. El uso más común de los servlets es generar páginas web de forma dinámica a partir de los parámetros de la petición que envíe el navegador web.

Herramientas CASE: Según sus siglas, Computer Aided Software Engineering, en español, Ingeniería de Software Asistida por Ordenador; son diversas aplicaciones informáticas destinadas a aumentar la

productividad en el desarrollo de software reduciendo el coste de las mismas en términos de tiempo y de dinero. Estas herramientas pueden ayudar en todos los aspectos del ciclo de vida de desarrollo del software en tareas como el proceso de realizar un diseño del proyecto, calculo de costes, implementación de parte del código automáticamente con el diseño dado, compilación automática, documentación o detección de errores, entre otras.