Universidad de las Ciencias Informáticas

Facultad 3

Título: Implementación e integración de las capas de presentación y lógica de negocio del módulo Seguimiento del proyecto ICICV

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autores: Yunier Ricardo Sánchez

Adrián Marín García

Tutora: Ing. Danaysa Macías Hernández

Ciudad de la Habana, mayo de 2009.


La virtud, como el arte,
se consagra constantemente
a lo que es difícil de hacer,
y cuanto más dura es la tarea,
más brillante es el éxito.
Aristóteles

DECLARACIÓN DE AUTORÍA

Declaramos ser autores del presente trabajo de diploma y reconocemos 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 ____.

Yunier Ricardo Sánchez

_____
Firma del Autor

Adrián Marín García

_____
Firma del Autor

Ing. Danaysa Macías Hernández

_____
Firma del Tutor

AGRADECIMIENTOS

A nuestros familiares por su cariño, dedicación, apoyarnos en cada momento y depositar su confianza en nosotros.

A nuestro Comandante en Jefe Fidel por tener la idea de crear esta gran universidad y darnos la oportunidad de formarnos en ella como profesionales.

A la Revolución, ese proceso gigante que ha engendrado tantos sueños y ha hecho el nuestro realidad.

A los amigos que hemos tenido en las diferentes etapas por las que hemos transitado y nos han apoyado incondicionalmente.

A nuestra tutora por apoyarnos, atendernos cuando la necesitamos y guiarnos para poder realizar el presente trabajo.

A todos los profesores que hemos tenido a lo largo de nuestra vida como estudiante, sin ellos no hubiese sido posible adquirir los conocimientos que hoy poseemos.

A todo aquel que de una forma u otra ha contribuido con la realización de este sueño.

Muchas gracias.

DEDICATORIA

A mi padre que aunque no está entre nosotros fue la persona que me guió, y me aconsejó en cada momento que lo necesité, quien me inspiró para hacer realidad este sueño.
A mi madre por su preocupación, cariño, comprensión y por el apoyo que siempre me ha brindado.
A mi hermano por apoyarme y ayudarme siempre que lo necesité.
A mis abuelos por preocuparse tanto por mí, especialmente Manuel.
A todos mis familiares por su apoyo incondicional.
A mis amistades.
A los profesores que contribuyeron a mi formación profesional.

Yunier Ricardo

A mi abuelo Miño que no pudo ver su más preciado sueño.
A mis padres que han hecho de mi lo lo que soy hoy.
A mi hermana por ser mi cómplice en todo momento.
A mis abuelos Ñico y Deysi por enseñarme lo principal de la Vida.
A mis tíos Manuel y Mirna por ser mis segundos padres y darme todo su amor como a su hijo.
A Yoan y a Mema por ser tan buenos conmigo.
A toda mi familia del monte que día a día trabajan de sol a sol y me enseñaron amar el trabajo.
A mi abuela Cachita por tenerme siempre presente en sus oraciones.
A Yaque por soportarme tanto tiempo y darme todo su amor.
A mis tíos José, Belkis, Santa y Silvia.
A mis primas.
A mis amigos.
A todos mis maestros.
A Fidel y Raúl.
A la Revolución.

Adrian Marín

RESUMEN

La República de Cuba y la República Bolivariana de Venezuela con el objetivo de fortalecer los tradicionales lazos de amistad entre los dos países, con interés común por progresar en sus respectivas economías y adquirir las ventajas que resultan de una cooperación que tenga resultados efectivos en el avance económico y social acordaron el cumplimiento del Convenio Integral de Cooperación entre la República de Cuba y la República Bolivariana de Venezuela. Para cumplir con lo acordado se realizan anualmente Mixtas de Cooperación en las cuales se firman importantes proyectos de colaboración que benefician a ambas naciones.

La ausencia de un software capaz de gestionar el control de los proyectos firmados trae como consecuencia la falta de organización, calidad y control de las actividades necesarias para llevar a cabo el desarrollo de los proyectos firmados dejando de realizarse obras que benefician a ambas repúblicas.

Para solucionar esos problemas se le asignó a la Universidad de las Ciencias Informáticas desarrollar un software para informatizar el Convenio Integral de Cooperación Cuba-Venezuela, el mismo está formado por siete módulos, entre ellos se encuentra el de Seguimiento.

El presente trabajo de diploma contiene la implementación e integración de las capas de presentación y lógica de negocio para el módulo Seguimiento, el cual contendrá las funcionalidades que permiten controlar el estado de ejecución física y la ejecución financiera de las actividades enmarcadas en la realización de los proyectos.

PALABRAS CLAVES

Capa de presentación, Capa de lógica de negocio, Implementación, Integración.

TABLA DE CONTENIDOS

INTRODUCCIÓN

En octubre de 2000 el Comandante en Jefe Fidel Castro y Hugo Chávez, Presidente de la República Bolivariana de Venezuela, firmaron en el Palacio de Miraflores el Convenio Integral de Cooperación Cuba-Venezuela (CICCV). Ambas naciones conscientes de su interés común por promover el progreso de sus economías y las ventajas recíprocas que resultan de una cooperación con miras a fortalecer el ámbito socio-cultural acordaron elaborar programas y proyectos de cooperación.

Para la ejecución de los diferentes proyectos se pactó la participación de organismos públicos, privados, organizaciones no gubernamentales y universidades. Cuba dispuso la prestación de servicios y tecnologías que estuvieran a su alcance para apoyar el programa de desarrollo económico-social en Venezuela. Estos programas son definidos cada año a través de las Comisiones Mixtas donde se acuerdan los términos financieros para los proyectos a nivel de gobierno, se precisan las especificaciones, regulaciones y modalidades en las que serán ejecutados.

En la actualidad el seguimiento de las actividades de los proyectos firmados en el Convenio se realiza de forma manual, lo que dificulta el trabajo de ambos países. Esto trae como consecuencia que no se tenga un control estricto y centralizado de los proyectos contratados, ni del monto facturado en cada uno de ellos. A ello se adiciona la difícil comunicación en la negociación por tratarse de naciones distantes geográficamente.

Ante tal situación se puede apreciar la necesidad de emplear herramientas de gestión de proyectos para efectuar las negociaciones enmarcadas en el Convenio de Colaboración, que amplíen la capacidad y rapidez de respuesta de las organizaciones e instituciones representadas por las partes contractuales, que tienen bajo su responsabilidad la ejecución de los proyectos. Además que permitan compartir información, 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. Estos son muy aceptados a nivel mundial pues proporcionan nuevos métodos de colaboración e integración.

Los Sistemas de Gestión de Proyectos especializados en las relaciones colaborativas se denominan Groupware, estos programas integran el trabajo de usuarios concurrentes que se encuentran en diversas estaciones de trabajo conectados a través de una red en un solo proyecto [22]. Existen varias herramientas de gestión que aportan ventajas y facilidades (definir hitos, tareas, notificaciones) bajo un entorno amigable y sencillo de administrar, las cuales permiten controlar la publicación de información y el avance del proyecto.

Como resultado de un análisis de las características específicas de distintas herramientas colaborativas, se llegó a la conclusión de que éstas no brindan las funcionalidades que le dan solución a los requerimientos exigidos por el CICCV, 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. Analizando la situación anterior se decidió desarrollar un Sistema de Gestión de Proyecto propio para el CICCV que garantice que no se pierdan sumas de dinero, y que permita controlar el estado de ejecución de los proyectos. En consecuencia, se le ha asignado a la Universidad de las Ciencias Informáticas durante la VIII Mixta la tarea de crear el proyecto Informatización del Convenio Integral Cuba-Venezuela (ICICV) para automatizar las actividades que se efectúan antes, durante y después de realizada la Comisión Mixta de cada año.

Para el desarrollo del sistema se dividió el proyecto en siete módulos estructurados en dependencia de las diferentes funcionalidades identificadas, entre ellos se encuentra el de Seguimiento. Éste se enmarca en la planificación, seguimiento y control de las actividades de cada uno de los proyectos contratados. Permite señalar el inicio de un proyecto en la fecha prevista en su concepción o en otra superior, modificar el contrato de los proyectos a través de documentos anexados, modificar o reformular los objetivos y metas trazadas en el inicio del proyecto, así como el cambio de los responsables de la ejecución del mismo o las instituciones a las que pertenecen.

Debido a las características políticas de cada país y a que no todos los organismos cubanos tienen acceso a Internet, se exigía la existencia de una aplicación en cada parte que fuera capaz de brindar servicios a los clientes de ambas naciones, así como toda la información relacionada con los proyectos, para en caso de que alguna información presente un determinado problema tener un

respaldo. Por tal exigencia se necesita contar con porciones de la aplicación que se puedan intercambiar sin tener que modificar el resto de la aplicación, lo que impulsa a la utilización de una Arquitectura basada en capas, donde las mismas se comuniquen entre sí y se distribuyan las responsabilidades de la aplicación para que el futuro sistema sea más versátil, ágil y poco complejo para las modificaciones posteriores y su mantenimiento.

La atención de este trabajo estará enfocada en las capas de presentación y lógica de negocio, dando respuesta a la necesidad que existe de implementar e integrar ambas capas para el módulo Seguimiento donde se cumpla con los requisitos funcionales y no funcionales asociados a dicho módulo.

Problema científico

¿Cómo contribuir al cumplimiento de los requisitos funcionales y no funcionales asociados al módulo Seguimiento del proyecto ICICV implementando e integrando las capas de presentación y lógica de negocio?

Objeto de estudio

Arquitectura en capas.

Campo de acción

Capas de presentación y lógica de negocio del módulo Seguimiento del proyecto ICICV.

Objetivo general

Implementar e integrar las capas de presentación y lógica de negocio para contribuir al cumplimiento de los requisitos funcionales y no funcionales del módulo Seguimiento del proyecto ICICV.

Idea a defender

La implementación e integración de las capas de presentación y lógica de negocio contribuyen al cumplimiento de los requisitos funcionales y no funcionales asociados al módulo Seguimiento.

Tareas trazadas

- Estudio e investigación de metodologías, herramientas y frameworks (marcos de trabajo) que pueden ser utilizados para desarrollar el módulo Seguimiento.
- Estudio del modelo de casos de usos del sistema correspondiente al módulo Seguimiento del proyecto ICICV.
- Implementar las capas de presentación y lógica de negocio del módulo Seguimiento.

El tipo de estrategia de investigación utilizada en este trabajo fue la siguiente:

Investigación exploratoria

Esta tiene como principal objetivo familiarizar al investigador con el tema objeto de estudio.

Métodos Científicos de investigación utilizados:

Los métodos científicos de investigación son la forma de estudiar la realidad natural, el pensamiento, los distintos fenómenos sociales con el objetivo de descubrir su esencia y sus relaciones. Para realizar la investigación se utilizaron los siguientes métodos teóricos:

Analítico-Sintético

Permitió desglosar las características y funcionalidades que tendría la aplicación a partir de un estudio de la descripción de los casos de uso y las reglas del negocio, que permitirán conocer los elementos más importantes que se deberán implementar en las capas de presentación y lógica de negocio, así como la integración de ambas.

Histórico-Lógico

Ayudó a realizar un estudio histórico sobre otras aplicaciones que realizaban trabajos similares, las herramientas utilizadas, para así poder tener un mayor conocimiento y saber aplicarlo en la solución del problema presentado.

Hipotético-Deductivo

Brindó la posibilidad de determinar los principales componentes que formarían parte del nuevo sistema que permitirían dar cumplimiento al objetivo propuesto.

Estructura del trabajo de diploma

Para el desarrollo del presente trabajo se proponen tres capítulos, los cuales están estructurados de la siguiente forma:

Capítulo 1: Fundamentación Teórica: En este capítulo se realiza un estudio del estado del arte acerca de las diferentes metodologías y herramientas que pueden ser empleadas para implementar el módulo Seguimiento. Se realiza un estudio acerca de los tipos de pruebas que pueden utilizarse para garantizar la calidad del producto realizado.

Capítulo 2: Descripción y análisis de la propuesta de solución. En este capítulo se analizan los subsistemas del módulo Seguimiento del proyecto ICICV, así como diferentes diagramas que muestran la estructura interna de cada uno de ellos. Se describe el estándar de codificación utilizado, así como las clases de las capas de presentación y lógica de negocio utilizadas para darle solución al problema existente. Se describe la política de integración entre ambas capas que permiten que el sistema cumpla con el pedido de los clientes.

Capítulo 3: Análisis de los resultados: En este capítulo se muestran las diferentes pruebas a las que fue sometido el sistema. Se realiza un breve análisis de los resultados obtenidos de las pruebas de caja negra realizadas al sistema.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1 INTRODUCCIÓN

En el presente capítulo se muestran las principales características y ventajas que posee una arquitectura por capas, así como el estudio de algunos frameworks que existen para facilitar la implementación de las mismas. Se ofrece un análisis de las diferentes metodologías de desarrollo de software más conocidas, además de hacer un estudio detallado de las tecnologías y herramientas que se utilizarán. En cada caso se asume una posición como cimiento a la realización del trabajo.

1.2 ARQUITECTURA EN CAPAS

Después de realizado un análisis por la dirección del proyecto de la necesidad que existía de separar partes de la aplicación en varios ordenadores, decidieron utilizar la Arquitectura en capas para que facilitara su separación en varios niveles, los cuales corresponden a la forma en que las capas lógicas estarán distribuidas físicamente. La Arquitectura Orientada a Servicios permite esta separación, pero no se utilizó debido a que no existe en la Universidad personas (arquitectos) capacitadas que sean capaces de desarrollar un proyecto de gran complejidad con esta arquitectura. Además, que es compleja y difícil de darle mantenimiento, un cambio en el código puede traer un gran impacto en el funcionamiento de la aplicación. Por otro lado, la Arquitectura en capas es la más conocida y utilizada en la Universidad, siendo definida desde el punto de vista arquitectónico para el desarrollo de aplicaciones de gran envergadura.

La arquitectura de un software es la vista conceptual de la estructura de éste. Toda aplicación contiene código de presentación, de procesamiento de datos y de almacenamiento de datos. La arquitectura de las aplicaciones difiere según como está distribuido el código. [1]

El principal objetivo de la programación por capas es la separación de la lógica de negocio de la lógica de diseño; un ejemplo básico de esto consiste en separar la capa de datos de la capa de presentación al usuario. [2]

La ventaja principal de esta arquitectura es que el desarrollo se puede llevar a cabo en varias capas y, en caso de que sobrevenga algún cambio, sólo se modifica la capa requerida sin tener que revisar entre código mezclado. Además, permite distribuir el trabajo de creación por capas, de esta forma cada desarrollador sólo debe centrarse y conocer lo referente a su capa. [3]

En el diseño de sistemas informáticos actual se suele usar la Arquitectura por capas o Programación por capas donde a cada una se le confía una misión simple, lo que permite el diseño de arquitecturas escalables (que pueden ampliarse con facilidad en caso de que las necesidades aumenten).

El desarrollo de aplicaciones bajo J2EE (Java 2 Enterprise Edition), al igual que otras plataformas como .NET, se basa en la separación de capas. Esta separación de capas permite una delimitación de responsabilidades a la vez que satisface los requisitos no funcionales de este tipo de aplicaciones (escalabilidad, extensibilidad, etc.) y disminuye el acoplamiento entre las diferentes partes de las mismas.

El número de capas de una aplicación J2EE variará según su complejidad y/o necesidades. Aún así, la estructura sugerida como modelo para el desarrollo de aplicaciones J2EE es la que se aprecia en la Figura 1.1.

Figura 1.1: Diseño en capas de una aplicación distribuida

Esta estructura en capas se ha constituido en un estándar a la hora de desarrollar aplicaciones distribuidas para sistemas empresariales dejando obsoleto el clásico modelo cliente-servidor. Las dos plataformas empresariales más importantes de la actualidad, J2EE y .NET, proponen este esquema de

desarrollo. Esta estandarización es bastante importante ya que si la arquitectura es clara y está diseñada en términos de alto nivel sin basarse en código explícito de cada plataforma será mucho más fácil una posible migración.

El rol de cada una de estas capas es similar en ambas plataformas variando únicamente la tecnología sobre la que se sustentan cada una de dichas capas:

- **Capa de cliente:** Es la capa donde se localizan los diferentes clientes del software. Estos normalmente serán de tipos y funcionalidades muy diversas.

- **Capa de presentación:** La capa de presentación contiene toda la lógica de interacción entre la aplicación y el usuario. Además, es la capa encargada de controlar la interacción entre éste y la lógica de negocio generando las vistas necesarias para mostrar información en la forma y formatos más adecuados.

- **Capa de lógica de negocio:** Es la capa donde se localiza el código y las reglas que sirven como núcleo de las aplicaciones. Como tal es importante que cumpla una serie de características fundamentales como la fácil extensibilidad y mantenibilidad, alta reutilización, alta flexibilidad y fácil adopción de tecnologías, etc.

- **Capa de integración:** Es la capa donde se realizan diferentes tareas de integración con otros sistemas como son el acceso a datos, el acceso a sistemas legales, la aplicación de motores de reglas o de workflow (Flujo de trabajo), etc. Es importante que estos sistemas sean especialmente extensibles de modo que sea fácil añadir nuevas fuentes sin que afecte a la capa de lógica de negocio.

- **Capa de recursos:** La capa de recursos contiene todos los sistemas de información a los que se accederá desde la capa de integración. Estos sistemas pueden ser muy diversos, desde servidores de bases de datos relacionales o de bases de datos orientadas a objetos, hasta sistemas de ficheros COBOL, pasando por diferentes ERPs (Enterprise Resource Planning,

Planificación de Recursos Empresariales), sistemas SAP o de otros fabricantes como Siebel o Navision, etc.

A continuación se analizarán las capas de presentación y lógica de negocio, así como los patrones de desarrollo más utilizados.

Capa de presentación

Esta capa puede localizarse en una aplicación de escritorio o dentro de un contenedor web. Independientemente de su localización, es la encargada de controlar la generación del contenido que se mostrará al usuario final. Esta generación comprende la comunicación con la capa de lógica de negocio para obtener los datos necesarios, el control del flujo de pantallas, el control de la interacción entre diferentes componentes, ensamblar las diferentes vistas que puedan formar una pantalla y limitar la información en base a perfiles de usuario y reglas de autorización, entre otras tareas.

Es fácil comprender que es una de las más complejas ya que engloba muchas tareas de diferente índole, es por ello que se debe de ser cuidadoso en su creación para no penalizar en exceso factores como la extensibilidad. Esta capa ha de ser extensible y fácil de mantener ya que es la más sensible al cambio (la lógica de negocio de la empresa no es algo que cambie tan frecuentemente) y por lo tanto es muy importante que pequeñas modificaciones y añadidos no interfieran en el funcionamiento de las aplicaciones existentes.

Ante una aplicación de escritorio o una aplicación web, el patrón más común a la hora de implementar la capa de presentación es el Modelo Vista Controlador (MVC, Model View Controller), que permite una separación entre lo que se conoce como modelo (en este caso será la lógica de negocio), el controlador y la vista.

MVC define tres roles diferentes: el modelo, la vista y el controlador. El modelo es un objeto que se encarga de almacenar información sobre los datos y comportamiento de estos en el dominio de la aplicación. La vista es una representación visual de los datos del modelo o de una parte de los mismos. El controlador se encarga de gestionar la interacción entre el modelo y las diferentes vistas,

de modo que cuando el modelo cambie se actualicen automáticamente todas las vistas reflejando dichos cambios.

Las ventajas de este patrón son múltiples, entre algunas de las más tratadas se destacan:

- Separación de responsabilidades. Cada una de las partes de este patrón se encarga de tareas muy diferentes que se encuentran aisladas y no interfieren entre ellas.

- Múltiples vistas. El escaso acoplamiento entre las partes de este patrón permite que se puedan generar fácilmente múltiples vistas para un mismo modelo. Cada vez que el modelo se ve modificado todas las vistas se actualizan automáticamente.

- Test de funcionalidad. La separación entre vista y modelo permite implementar de manera muy sencilla conjuntos de tests para probar el funcionamiento del modelo y del controlador utilizando una vista mucho más sencilla que la que tendrá la aplicación final. Una vez que se comprueba el correcto funcionamiento del modelo y el controlador se puede desarrollar una vista mucho más rica.

Capa de lógica de negocio

La capa de lógica de negocio es sin duda la más importante dentro de una aplicación ya que es la que va a contener todo el conjunto de entidades, relaciones y reglas que se encargan de la implementación de los procesos de negocio de la empresa. Los datos, por si mismos, rara vez carecen de significado y es necesario tratarlos utilizando reglas y procesos propios de cada empresa. Estas reglas y procesos se conocen como lógica de negocio.

Esta capa se comunica con la capa de presentación, para recibir las solicitudes y presentar los resultados, y con la capa de datos, para solicitar al gestor de base de datos el almacenamiento o recuperación de datos de él.

Los blueprints (escritos azules) de Java para J2EE recomiendan la implementación de los procesos de negocio como beans (componente software que tiene la particularidad de ser reutilizable, es un objeto

que es creado y manejado por el contenedor Spring) de sesión sin estado y los datos como beans de entidad. A continuación se analizarán otras alternativas para representar la lógica de negocio.

Beans de sesión

Los beans de sesión son la opción recomendada por los blueprints (escritos azules) de SUN Microsystems para la implementación de la lógica de negocio de las aplicaciones basadas en J2EE 1.3. Se trata de un tipo especial de EJBs (Enterprise JavaBeans) que representan procesos o conjuntos de tareas del software. Como tal, se trata de una opción ideal para la representación de una arquitectura orientada al servicio (SOA, Service Oriented Architecture), donde cada bean de sesión ofrece un servicio al exterior. El extremo contrario sería una arquitectura orientada al dominio (DOA o Domain Oriented Architecture) donde la lógica de negocio en lugar de estar contenida en beans de sesión, se encontrará dentro de los objetos de dominio, es decir, dentro de los beans de entidad.

Fachada de sesión

La Fachada de Sesión o Session Facade, es uno de los patrones básicos en el diseño de proyectos que utilicen EJBs (Enterprise JavaBeans).

Los desarrolladores se lanzaron al uso de la tecnología de EJBs, sin tener en cuenta algunos factores que disminuían considerablemente el rendimiento. Uno de los factores más importantes era el conocido como round trip o tiempo de acceso al servidor web. Cada vez que se realiza una llamada a un EJB, es necesario transmitir y recibir información a través de la red desde el cliente al servidor. Si el número de llamadas es excesivo, como comúnmente sucedía, la aplicación veía como decrecía alarmantemente su rendimiento. Los desarrolladores utilizaban los beans de sesión y de entidad como si se tratase de objetos locales y no se daban cuenta de las consecuencias que esto tenía para el rendimiento. Obviamente, en el momento del despliegue de las aplicaciones las consecuencias eran dramáticas. [2]

La Fachada de Sesión es un patrón de diseño que evita este problema. Para solucionarlo, se intenta evitar a toda costa las llamadas finas a EJBs utilizando llamadas mucho más gruesas, es decir, llamadas que realicen procesos completos. Esa es justamente la mejor función para un bean de

sesión, por lo que estos beans son los candidatos ideales para la implementación de este patrón. El cliente realizará una única llamada a través de la red y el bean de sesión se encargará de realizar múltiples llamadas locales a los diferentes componentes de la capa de integración hasta que complete todo el proceso de negocio que quería realizar el cliente.

Beans de entidad

Los beans de entidad representan datos y pueden contener en su interior lógica de negocio, siendo esta aproximación mucho más cercada a un diseño orientado al dominio que a un diseño orientado al servicio. Si los beans de entidad únicamente van a ser accedidos de modo local, entonces la decisión de colocar lógica de negocio dentro de los beans de entidad puede ser apropiada si el diseño concuerda con esta decisión.

1.3 METODOLOGÍAS DE DESARROLLO DE SOFTWARE.

Una metodología es un conjunto de procedimientos que permiten producir y mantener un producto software, definiendo una serie de pasos a seguir para obtener un software de calidad. Un proceso define quién está haciendo qué, cuándo, y cómo alcanzar un determinado objetivo [9]. Las metodologías se desarrollan con el objetivo de dar solución a los problemas existentes en la producción de software, que cada vez son más complejos. Estas abarcan procedimientos, técnicas, documentación y herramientas que se utilizan en la creación de un producto de software.

Ejemplos de metodologías Orientadas a Objetos:

- Proceso Unificado de Desarrollo de Software (RUP)
- Programación Extrema (XP)
- Marcos de Soluciones Microsoft (MSF)

A continuación un breve análisis de ellas.

1.3.1 PROCESO UNIFICADO DE DESARROLLO DE SOFTWARE (RUP).

RUP es el resultado de varios años de desarrollo y uso práctico en el que se han unificado técnicas de desarrollo y trabajo de muchas metodologías. Es un proceso de software genérico que puede ser

utilizado para una gran cantidad de tipos de sistemas de software, para diferentes áreas de aplicación, diferentes tipos de organizaciones y diferentes tamaños de proyectos. Proporciona un enfoque disciplinado en la asignación de tareas y responsabilidades dentro de una organización de desarrollo. Su objetivo es asegurar la producción de software de muy alta calidad que satisfaga las necesidades de los usuarios finales, dentro de un calendario y presupuesto predecible.

Como RUP es un proceso, en su modelación define como sus principales elementos:

Trabajadores (“quién”)	Define el comportamiento y responsabilidades (rol) de un individuo, grupo de individuos, sistema automatizado o máquina, que trabajan en conjunto como un equipo. Ellos realizan las actividades y son propietarios de elementos.
Actividades (“cómo”)	Es una tarea que tiene un propósito claro, es realizada por un trabajador y manipula elementos.
Artefactos (”qué”)	Productos tangibles del proyecto que son producidos, modificados y usados por las actividades. Pueden ser modelos, elementos dentro del modelo, código fuente y ejecutables.
Flujo de actividades (“Cuándo”)	Secuencia de actividades realizadas por trabajadores y que produce un resultado de valor observable.

RUP define nueve flujos de trabajo principales, donde seis son flujos de ingeniería y tres de apoyo. En la Figura 1.2 se refleja a RUP en dos dimensiones donde se grafican los flujos de trabajo y las fases.

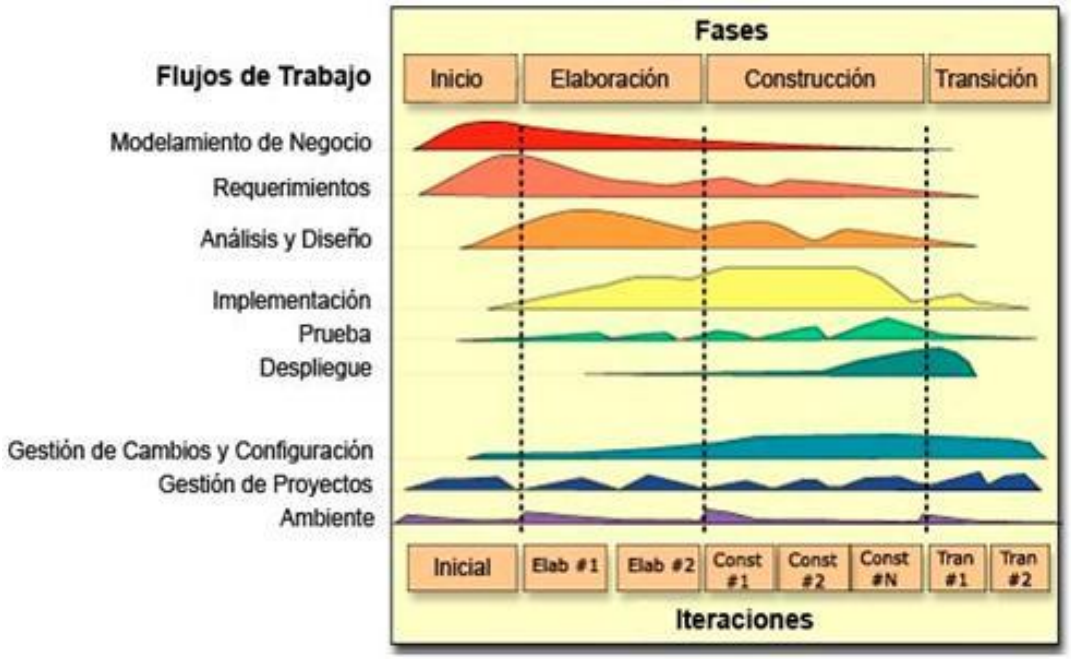

Figura 1.2: Representación del proceso de desarrollo de software según RUP [9]

El ciclo de vida de RUP está caracterizado por ser:

- **Dirigido por casos de uso**

Los casos de uso reflejan lo que los usuarios futuros necesitan y desean, lo cual se capta cuando se modela el negocio y se representa a través de los requerimientos. A partir de aquí los casos de uso guían el proceso de desarrollo ya que los modelos que se obtienen, como resultado de los diferentes flujos de trabajo, representan la realización de los casos de uso.

- **Centrado en la arquitectura**

La arquitectura muestra la visión común del sistema completo en la que el equipo de proyecto y los usuarios deben estar de acuerdo, por lo que describe los elementos del modelo que son más importantes para su construcción, los cimientos del sistema que son necesarios como base para

comprenderlo, desarrollarlo y producirlo económicamente. RUP se desarrolla mediante iteraciones, comenzando por los CU relevantes desde el punto de vista de la arquitectura.

- **Iterativo e Incremental**

RUP propone que cada fase se desarrolle en iteraciones o sea dividir el proyecto en mini-proyectos. Cada mini-proyecto resulta una iteración, cada iteración involucra actividades de todos los flujos de trabajo, aunque desarrolla fundamentalmente algunos más que otros. Cada iteración se realiza de forma planificada.

1.3.2 PROGRAMACIÓN EXTREMA (XP)

Es una de las metodologías de desarrollo de software más exitosas utilizada en la actualidad para proyectos de corto plazo y equipos pequeños. La metodología consiste en una programación rápida o extrema, cuya particularidad es tener como parte del equipo, al usuario final, pues es uno de los requisitos para llegar al éxito del proyecto.

Esta metodología posee cinco principios fundamentales, que son: Simplicidad, Comunicación, Retroalimentación, Disposición para enfrentar el cambio en cualquier momento y Respeto.

- **Simplicidad:**

La simplicidad es la base de la programación extrema. Se simplifica el diseño para agilizar el desarrollo y facilitar el mantenimiento. También se aplica la simplicidad en la documentación, de esta manera el código debe comentarse en su justa medida. 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 este. 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 ya que el cliente forma parte del equipo de desarrollo. El cliente decide qué características tienen prioridad y siempre debe estar disponible para solucionar dudas.

- **Retroalimentación:**

Al estar el cliente integrado en el proyecto, su opinión sobre el estado del proyecto 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 tirarse 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.

- **Disposición para enfrentar el cambio:**

Se debe estar dispuesto 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.

- **Respeto:**

El respeto se manifiesta de varias formas. Los miembros del equipo se respetan los unos a otros, porque los programadores no pueden realizar cambios que hacen que las pruebas existentes fallen o que demore el trabajo de sus compañeros. Los miembros se respetan sus trabajos porque siempre

están luchando por la alta calidad en el producto y buscando el diseño más optimo para la solución a través de la refactorización del código. [11]

Las características fundamentales de esta metodología son:

- Desarrollo iterativo e incremental: pequeñas mejoras, unas tras otras.
- Pruebas unitarias continuas, frecuentemente repetidas y automatizadas, incluyendo pruebas de regresión.
- Programación en parejas: se recomienda que las tareas de desarrollo se lleven a cabo por dos personas en un mismo puesto.
- Frecuente integración del equipo de programación con el cliente o usuario. Se recomienda que un representante del cliente trabaje junto al equipo de desarrollo.
- Corrección de todos los errores antes de añadir una nueva funcionalidad. Hacer entregas frecuentes.
- Refactorización del código, es decir, reescribir ciertas partes del código para aumentar su legibilidad sin modificar su comportamiento.
- Propiedad del código compartida: en vez de dividir la responsabilidad en el desarrollo de cada módulo en grupos de trabajo distintos, este método promueve que todo el personal pueda corregir y extender cualquier parte del proyecto.
- Simplicidad en el código: es la mejor manera de que las cosas funcionen. Cuando todo funcione se podrá añadir otra funcionalidad si es necesario.
- La simplicidad y la comunicación son extraordinariamente complementarias. Con más comunicación resulta más fácil identificar qué se debe y qué no se debe hacer.

1.3.3 MARCO DE SOLUCIONES MICROSOFT (MSF)

MSF es una flexible e interrelacionada serie de conceptos, modelos y prácticas de uso que controlan la planificación, el desarrollo y la gestión de proyectos tecnológicos. MSF se centra en los modelos de proceso y de equipo dejando en un segundo plano las elecciones tecnológicas. Originalmente creado en 1994 para conseguir resolver los problemas a los que se enfrentaban las empresas en sus

respectivos proyectos, se ha convertido posteriormente en un modelo práctico que facilita el éxito de los proyectos tecnológicos. Concretamente MSF se compone de principios, modelos y disciplinas. [14]

MSF posee varios principios como: Promover comunicaciones abiertas, trabajar para una visión compartida, fortalecer los miembros del equipo, establecer responsabilidades claras y compartidas, focalizarse en agregar valor al negocio, permanecer ágil y esperar los cambios, invertir en la calidad y aprender de todas las experiencias.

También se compone de varios modelos encargados de planificar las diferentes partes implicadas en el desarrollo de un proyecto como: Modelo de Arquitectura del Proyecto, Modelo de Equipo, Modelo de Proceso, Modelo de Gestión del Riesgo, Modelo de Diseño de Proceso y el Modelo de Aplicación.

Entre las disciplinas se pueden encontrar:
Disciplina de Gestión de Riesgos: Para ayudar al equipo a identificar las prioridades, tomar las decisiones estratégicas correctas y controlar las emergencias que puedan surgir. Esta disciplina proporciona un entorno estructurado para la toma de decisiones y acciones valorando los riesgos que puedan provocar.

Disciplina de Gestión de Proyectos: Describe el rol de la gestión del proyecto dentro del modelo de equipo de MSF, y permite mayor escalabilidad desde proyectos pequeños a proyectos largos y complejos.

Disciplina de Gestión de la Preparación: Esta disciplina describe aquellos conocimientos, aptitudes y habilidades que son necesarias para planificar, desarrollar y gestionar soluciones satisfactorias.

Tras evaluar las anteriores metodologías se adoptará RUP como proceso rector del desarrollo. Por ser apropiado para proyectos complejos y de larga duración. Por tener el equipo de desarrollo conocimiento en su aplicación y por contar con clientes que no tendrán una relación directa con el equipo del proyecto. También define claramente actividades realizadas por roles generando a su paso artefactos que sustenten el proceso de construcción de software. Constituye una metodología,

adaptable al proyecto, utilizada para el análisis, implementación y documentación de sistemas a través del UML (Unified Modeling Language), que implementa el Paradigma Orientado a Objetos.

1.4 PLATAFORMAS PARA EL DESARROLLO

En la actualidad, el desarrollo de software se ha incrementado considerablemente, para lograr este incremento se han desarrollado plataformas que son muy utilizadas a nivel mundial como por ejemplo: J2EE de Sun Microsystems y .Net de Microsoft.

1.4.1 PLATAFORMA J2EE

J2EE (Java 2 Plataform Enterprise Edition, Edición Empresarial para la Plataforma Java 2) es, según la definición de Sun Microsystems, un conjunto de estándares y especificaciones para el desarrollo de aplicaciones empresariales basado en la tecnología Java. Esquemáticamente se resumiría en la siguiente fórmula= Java + Componentes adicionales orientados a empresas (EJBs, JSPs).

La plataforma J2EE ha sido creada con la participación de cientos de empresas de diversa índole y es, sin lugar a dudas una plataforma conjunta, no exclusiva de Sun o de ninguna otra compañía. Actualmente se han desarrollado una serie de herramientas (quizás las más conocidas sean JBuilder, de Borland, y ForteTM de la propia Sun) comerciales para implementar esta plataforma.

El lenguaje en el que se basa J2EE es Java, un lenguaje orientado a objetos que alcanzó su madurez con la popularización de Internet y que es en cierta manera el heredero legítimo de C++. La expansión de este lenguaje entre la comunidad de programadores ha sido vertiginosa y se ha impuesto como el paradigma de los lenguajes de programación orientados a objetos. [4]

Java incorpora un recolector automático de memoria (garbage collector), con lo que elimina una fuente tradicional de problemas en C/C++. Es un lenguaje fuertemente "tipado", en el que se comprueban los tipos declarados en tiempo de compilación, a diferencia de lo que sucede en Smalltalk, donde ésto se comprueba en tiempo de ejecución (lo cual suele retrasar el proceso de depuración). Java admite la herencia múltiple (es decir, una clase puede derivar de varias clases distintas) de un modo distinto a lo

que lo hace C++, usando interfaces. Una interfaz es una colección de nombres de métodos sin definiciones reales (o lo que es lo mismo: sin implementación) que indican que una clase tiene un conjunto de comportamientos, además de los que la clase hereda de sus superclases.

Una de las novedades revolucionarias de Java fue la portabilidad: Sun abordó el problema introduciendo el modelo de bytecodes: cuando un programa Java se compila no se transforma en un conjunto de instrucciones código máquina nativas de la plataforma utilizada (lo cual impediría su completa portabilidad), sino que se transforma en un conjunto de bytecodes independientes de la plataforma utilizada que son leídos e interpretados por la máquina virtual Java (JVM) para ejecutar el programa. Por ejemplo, cuando se compila un programa Java en una plataforma Windows/Intel, se obtiene la misma salida compilada (o los mismos bytecodes) que en un sistema Macinstosh o Unix. [4]

Esta máquina virtual Java (recibe este nombre porque es una máquina imaginaria que se implementa emulando por software una máquina real) es una aplicación que debe ser instalada en el ordenador para que los programas Java puedan ser ejecutados y será diferente para cada plataforma. A la hora de hacer funcionar una aplicación Java en una plataforma distinta a aquella en la que se escribió el programa, bastará con llevar los archivos de bytecodes generados por la primera plataforma a la otra plataforma (donde deberá haberse instalado la máquina virtual Java correspondiente a ésta, que será distinta a la máquina virtual Java de la primera plataforma).

Java, como lenguaje de programación es multipropósito, robusto, estable, independiente de la plataforma, puede utilizarse para realizar aplicaciones en múltiples plataformas hardware y sistemas operativos (Unix, Linux, OS/390, Windows 2000, ó HP-UX, entre otros sistemas operativos para ordenadores personales o estaciones de trabajo), reúne todas las características de un ambiente orientado a objetos: es fácil de aprender para programadores que hayan trabajado previamente con lenguajes orientados a objetos, cuenta con capacidad de generación de aplicaciones distribuidas, segura, de arquitectura neutral, portable, multihilo, dinámico y de alto rendimiento. Pero ésto no lo es todo, la API (Application Programming Interface, interfaz de programación de aplicación) de Java es muy versátil, ya que está formada por un conjunto de paquetes de clases que le proporcionan una extensa funcionalidad. El núcleo de la API cuenta con cada una de las implementaciones de la

máquina virtual: tipos de datos, clases y objetos, manejo de red, seguridad, componentes, etc. Estos componentes son llamados Java Beans, los cuales son código reusable que se puede desarrollar fácilmente para crear aplicaciones sofisticadas. Se puede decir que con Java, Sun Microsystems introdujo en el mercado la primera plataforma de software universal diseñada desde y para el crecimiento de Internet y de las intranets corporativas. Esta tecnología permite escribir aplicaciones una sola vez y ejecutarlas en cualquier computadora, lo que, desde entonces ha revolucionado el mundo del desarrollo de software por representar un cambio de paradigma. [8]

Java fue pensado originalmente para utilizarse en cualquier tipo de electrodoméstico pero la idea fracasó. Uno de los fundadores de Sun Microsystems rescató la idea para utilizarla en el ámbito de Internet y convirtieron a Java en un lenguaje potente, seguro y universal gracias a que lo puede utilizar todo el mundo y es gratuito. Uno de los primeros triunfos de Java fue que se integró en el navegador Netscape y permitía ejecutar programas dentro de una página web, hasta entonces impensable con el HTML. [9]

Actualmente Java se utiliza en un amplio abanico de posibilidades y casi cualquier cosa que se puede hacer en cualquier lenguaje se puede hacer también en Java y muchas veces con grandes ventajas. Con Java se pueden programar páginas web dinámicas, con accesos a bases de datos, utilizando XML, con cualquier tipo de conexión de red entre cualquier sistema. En general, cualquier aplicación que se desee hacer con acceso a través de la web se puede desarrollar utilizando Java.

J2EE incluye varias especificaciones de API, tales como JDBC, RMI, e-mail, JMS, Servicios Web, XML, etcétera, y define cómo coordinarlos. Además configura algunas especificaciones únicas para componentes. Estas incluyen Enterprise JavaBeans (EJBs), servlets, portlets (siguiendo la especificación de Portlets Java), JavaServer Pages (JSP) y varias tecnologías de servicios web, lo que permite al desarrollador crear una aplicación de empresa portable entre plataformas y escalable, a la vez que integrable con tecnologías anteriores. Otros beneficios añadidos son, por ejemplo, que el servidor de aplicaciones puede manejar transacciones, la seguridad, escalabilidad, concurrencia y gestión de los componentes desplegados, significando que los desarrolladores pueden concentrarse más en la lógica de negocio de los componentes en lugar de tareas de mantenimiento de bajo nivel.

La plataforma J2EE define un modelo de programación encaminado a la creación de aplicaciones basadas en n-capas. La lógica de la aplicación se divide en componentes de diferentes funciones que componen una aplicación J2EE y que están distribuidos en dependencia de la capa en el ambiente multicapas J2EE al cual la aplicación pertenece. El modelo de desarrollo con J2EE está basado en componentes reutilizables, con el objetivo de aumentar la reusabilidad de las aplicaciones. Estos componentes gracias a las especificaciones, son intercambiables entre servidores de aplicaciones, por lo que la portabilidad de las aplicaciones es máxima. [5]

Como se puede apreciar, J2EE no es sólo una plataforma o una tecnología, sino un estándar de desarrollo, construcción y despliegue de aplicaciones. J2EE ofrece muy buenas perspectivas para la implementación de software empresarial para aquellos sistemas informáticos que requieran basar su arquitectura en productos basados en software libre.

1.4.2 PLATAFORMA .NET

Microsoft .Net es, de acuerdo con la definición de Microsoft, una plataforma que comprende servidores, clientes y servicios. Consiste en un conjunto de aplicaciones como Visual Studio .Net, los servicios .Net, etc. Esta plataforma es una implementación basada en estándares abiertos como SOAP, WSDL, C#, y el CLI (Command Line Interface). Desde el punto de vista del programador, el entorno .NET ofrece un sólo entorno de desarrollo para todos los lenguajes que soporta (actualmente [Abril 2002] unos 30: Visual Basic, C++, C#, Visual J#, Fortran, Cobol...).

La estrategia .Net es innovadora en el sentido de que no compila aplicaciones en código nativo, es decir, no compila aplicaciones en código específico para Intel o Mac. La compilación, al igual que sucede con Java, se realiza en dos pasos sucesivos. El código escrito por el programador se compila en el lenguaje intermedio de Microsoft (MSIL: Microsoft Intermediate Language), del mismo modo que las instrucciones en Java se convierten en bytecodes. Es entonces cuando el CLR (CLR: Common Language Runtime ó entorno común de ejecución) de Microsoft compila en tiempo de ejecución las aplicaciones en código nativo de la plataforma Intel ó Mac. El CLR también revisa el código, verificando la seguridad del mismo y recolectando los objetos para los cuales no existe ya ninguna

referencia (recolección de basura), además de gestionar las excepciones. Es inevitable comparar esta manera de trabajar con los bytecodes de Java. [4]

Para generar el código MSIL, los compiladores de .NET utilizan la información recogida en el CTS (Common Type System), un sistema de tipos comunes en el que se incluyen todos los tipos de datos, estructuras y operaciones de muchos lenguajes de alto nivel distintos. Para que el código pueda utilizarse en otras aplicaciones escritas en otro/otros lenguajes es necesario que los lenguajes usados cumplan la CLS (Common Language Specification), una especificación de los tipos de datos, estructuras y operaciones comunes a todos los lenguajes de programación (y, por tanto, un subconjunto del CTS).

El código escrito en un lenguaje que cumpla la CLS puede ser utilizado en el entorno .NET en aplicaciones escritas en otros lenguajes que también cumplan la CLS (Nota: Es precisa la exigencia de que cumplan la CLS, pues el CTS es demasiado general y podría suceder que un lenguaje A usara características del CTS de las que otro lenguaje B carece, por lo que no sería posible utilizar código escrito en A en aplicaciones escritas en B y viceversa). Por ejemplo, es posible definir una clase en C# y derivar una subclase de ella usando Visual Basic .Net, pues ambos lenguajes se ajustan a la CLS aunque su sintaxis sea muy diferente. Del mismo modo, y por poner otro ejemplo, un bucle que imprima diez veces el tradicional mensaje "Hola Mundo" se escribirá de distinta manera en C# y Visual Basic .Net (o en lenguajes de terceras partes como Fortran, Cobol, etc.), pero los compiladores de estos lenguajes, si cumplen la CLS, generarán el mismo código MSIL.

C# (C Sharp) es un nuevo lenguaje de programación incluido en la plataforma .Net por vez primera. Es un lenguaje orientado a objetos fuertemente "tipado", diseñado por Microsoft para obtener un elevado rendimiento con una relativa simplicidad del lenguaje. Como ya se ha apuntado antes, pero vale la pena remarcar este punto, la plataforma .NET está centrada en torno al Common Language Runtime (similar a la Java Virtual Machine de Sun) y a un conjunto de bibliotecas que pueden ser usadas por una amplia variedad de lenguajes capaces de trabajar conjuntamente al ser compilados todos en el lenguaje intermedio MSIL, ya citado anteriormente. C# juega un importante papel en .NET porque ha sido diseñado para trabajar de forma óptima con .NET y ciertas características de .NET se

implementaron pensando en que su rendimiento fuera óptimo con C# (de hecho, algunas bibliotecas de .NET como Collection, XML, ADO+, ASP+, GDI+ y otras fueron escritas en C#). [4]

La plataforma .Net permite utilizar un lenguaje llamado Visual J# (comentado de forma oficial por Microsoft), que implementa casi de forma completa el JDK 1.1.4 (Java Development Kit) de Sun dentro de .Net. Se ha diseñado teniendo muy presente los servicios Web, una nueva forma de enfocar el negocio del software.

1.4.3 COMPARACIÓN ENTRE J2EE y .NET

El propósito tanto de J2EE como de la plataforma .NET es facilitar y simplificar el desarrollo de aplicaciones corporativas. Los servidores de aplicaciones J2EE y .Net proporcionan un modelo de acceso de componentes a datos y de lógica del negocio, separados por una capa intermedia de presentación implementada mediante ASP .Net (.Net) ó Servlets (J2EE).

Visual Basic .Net y C# son lenguajes orientados a objetos, al igual que Java, y en su diseño ha tenido mucha importancia la existencia de Internet. [4]

Desde la perspectiva de los desarrolladores, J2EE y .Net proporcionan las herramientas para crear Servicios Web.

Tanto .NET como J2EE tienen un fin común a la hora de abordar problemas como la seguridad, portabilidad, etc. En el aspecto funcional ambas plataformas guardan varias similitudes; se programa en un lenguaje que luego se compila a un código intermedio (“Intermediate Language” en el caso de Microsoft .NET y “Bytecodes” en el caso de Java). Este código se ejecutará en un “entorno de ejecución” que transformará el lenguaje intermedio a código propio de la máquina en la que se corre la aplicación, Common Language Runtime (CLR, Entorno Común del Lenguaje) en Microsoft .NET y Java Runtime Environment (JRE, Entorno de Ejecución de Java) en J2EE. [5]

A la vista del lenguaje, algunas (no todas) de las similitudes entre Java y C# son las siguientes:

- Ambos lenguajes compilan un código independiente de la máquina y el sistema operativo que se ejecuta mediante sus correspondientes entornos (JVM o entorno .NET).
- No necesitan punteros (Java no los permite y C# permite un uso restringido dentro del código señalado como unsafe (inseguro).
- No se utilizan ficheros de cabecera (como en C y C++), todo el código se empaqueta en packages (Java) o assemblies (C#).
- Admiten herencia múltiple mediante interfaces.
- Permiten clases internas (clases en el interior de otras clases).
- No admiten programación estructurada, cualquier "cosa" pertenece a una clase.

A pesar de las similitudes de ambas plataformas tienen diferencias que las hacen únicas:

- Las implementaciones de J2EE pueden adquirirse a distintas compañías, mientras que .Net solo puede comprarse a Microsoft. El hecho de que hayan distintas organizaciones implementando J2EE ofrece mayor variedad para los usuarios finales y permite la existencia de una cierta competencia entre ellas para obtener mejores productos que no existe en el caso de Microsoft y su .Net.
- Las aplicaciones Java pueden correr en una amplia gama de sistemas operativos (desde sistemas como Windows 2000, OS/390, Solaris, HP-UX, IRIX u otras versiones de Unix) y de arquitectura hardware. Hasta la fecha, .Net corre solamente sobre sistemas operativos de Microsoft (aunque esta situación podría cambiar en el futuro), siendo J2EE el único entorno de desarrollo que ofrece una independencia real de la plataforma.
- Aunque Java fue creado originalmente por una compañía: Sun MicroSystems, lo cierto es que J2EE es ahora el producto de la colaboración de más de 400 empresas y organizaciones de todo tipo (públicos, privados sin ánimo de lucro, privados con ánimo de lucro, y de normalización en ámbitos nacionales e internaciones). La plataforma .Net es, y será, el producto de una sola compañía, que aunque haya implementado algunos

estándares en .Net y esté intentando conseguir que ciertas tecnologías se conviertan en estándares "oficiales", no puede tener el mismo consenso que .Net (sobre todo teniendo en cuenta que la mayor parte de su código no es público). [4]

- Una de las principales características de la plataforma .NET es que soporta una gran cantidad de lenguajes de programación como C# (su lenguaje principal), Visual Basic, C++, entre otros. J2EE, por su parte, el único lenguaje que soporta es Java y es el que se tendrá que utilizar para el desarrollo de todos los componentes. Existen sólo dos formas oficiales para acceder a la plataforma J2EE con otros lenguajes, la primera es a través de JNI (Java Native Interface) y la segunda es a través de la interoperabilidad que ofrece CORBA. [5]

- .NET cuenta con un ambiente de desarrollo llamado Visual Studio .NET, mientras que J2EE no es un producto de ninguna empresa, se trata en cambio de un estándar, por lo que no cuenta con un entorno de desarrollo tipo “Visual Studio”. Existen como alternativa en el mercado múltiples productos que ofrecen entornos de desarrollo adecuados, tales como NetBeans de Sun Microsystems, Visual Café de WebGain, Visual Age for Java de IBM, Eclipse, entre otros. La mayoría de estos entornos de desarrollo no ofrecen las prestaciones y las posibilidades que brinda Visual Studio .NET.

- Las herramientas de desarrollo incluidas por Microsoft en su Visual Studio .Net son mucho más simples, intuitivas y sencillas de manejar que las herramientas de desarrollo equivalentes en J2EE suministradas por otras empresas (entre ellas la propia Sun). Cualquier programador medio/avanzado se manejará rápidamente con la programación del interface de usuario en Visual Studio .Net, al igual que sucedía con versiones anteriores de Visual Studio.

Después de realizado el estudio de las plataformas J2EE y .Net, se puede apreciar que ambas tienen tanto similitudes como ventajas una sobre la otra, pero como para desarrollar el módulo de Seguimiento se requiere de una aplicación que sea multiplataforma, que sea una tecnología que opere bajo licencia abierta y que pueda implementarse sobre cualquier sistema operativo, se escogió la plataforma J2EE. Además, la plataforma J2EE resulta una propuesta de vanguardia que responde, de

manera natural, a la demanda actual para el desarrollo de software bajo el concepto de arquitectura en capas.

1.5 ENTORNO INTEGRADO DE DESARROLLO (IDE)

Un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica GUI. Los IDEs pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes, proveen un marco de trabajo amigable para la mayoría de los lenguajes de programación. Es posible que un mismo IDE pueda funcionar con varios lenguajes de programación. Este es el caso de Eclipse o NetBeans, que mediante pluggins se le puede añadir soporte de lenguajes adicionales.

1.5.1 ECLIPSE

Eclipse es un entorno de desarrollo integrado de código abierto multiplataforma para desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores. Esta plataforma, típicamente ha sido usada para desarrollar entornos de desarrollo integrados, como el IDE de Java llamado Java Development Toolkit (JDT) y el compilador (ECJ) que se entrega como parte de Eclipse (y que son usados también para desarrollar el mismo Eclipse). El IDE de Eclipse emplea módulos (en inglés plug-in) para proporcionar toda su funcionalidad, a diferencia de otros entornos monolíticos donde las funcionalidades están todas incluidas, las necesite el usuario o no. El mecanismo de módulos permite que el entorno de desarrollo soporte otros lenguajes además de Java. Por ejemplo, existe un módulo para dar soporte a C/C++. Existen módulos para añadir un poco de todo, desde Telnet hasta soporte a bases de datos.

Eclipse fue desarrollado originalmente por IBM como el sucesor de su familia de herramientas para VisualAge. Eclipse es ahora 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.

La versión actual de Eclipse dispone de las siguientes características:

- Editor de texto.
- Resaltado de sintaxis.
- Compilación en tiempo real.
- Pruebas unitarias con JUnit.
- Control de versiones con CVS.
- Integración con Ant.
- Asistentes (wizards): para creación de proyectos, clases, pruebas, etc.
- Refactorización.

Asimismo, a través de "plugins" libremente disponibles es posible añadir:

- Control de versiones con Subversion.
- Integración con Hibernate.

Eclipse fue liberado originalmente bajo la Licencia Pública Común pero después fue re-licenciado bajo la Licencia Pública del Eclipse. La Fundación del Software Libre ha dicho que ambas licencias son de software libre, pero son incompatibles con la Licencia Pública General de GNU (GNU GPL).[12]

1.5.2 NETBEANS

NetBeans se refiere a una plataforma para el desarrollo de aplicaciones de escritorio usando Java y a un entorno de desarrollo integrado (IDE) desarrollado usando la Plataforma NetBeans. Es un proyecto de código abierto de gran éxito con una gran base de usuarios, una comunidad en constante crecimiento, y con cerca de 100 socios en todo el mundo. Sun MicroSystems fundó el proyecto de código abierto NetBeans en junio de 2000 y continúa siendo el patrocinador principal de los proyectos.

El IDE NetBeans es una herramienta para programadores pensada para 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 este IDE. Es un producto libre y gratuito sin restricciones de uso [13].

NetBeans es un IDE escrito completamente en Java que soporta el desarrollo de todos los tipos de aplicaciónes Java (J2SE, web, EJB y aplicaciones móviles). Además, es un sistema de proyectos basado en herramientas de código abierto utilizadas en la compilación y creación de programas Java que permite el cambio de código sin afectar la aplicación.

La plataforma NetBeans permite que las aplicaciones sean desarrolladas a partir de un conjunto de componentes de software llamados módulos. Un módulo es un archivo Java que contiene clases de java escritas para interactuar con las APIs de NetBeans y un archivo especial (manifest file) que lo identifica como módulo. Las aplicaciones construidas a partir de módulos pueden ser extendidas agregándole nuevos módulos. Debido a que los módulos pueden ser desarrollados independientemente, las aplicaciones basadas en la plataforma NetBeans pueden ser extendidas fácilmente por otros desarrolladores de software.

La versión actual es NetBeans IDE 6.5, la cual fue lanzada el 19 de Noviembre de 2008. NetBeans IDE 6.5 extiende las características existentes del Java EE (incluyendo Soporte a Persistencia, EJB 3 y JAX-WS). 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 servicies, y modelado UML. El NetBeans C/C++ Pack soporta proyectos de C/C++ [13].

Todas las funciones del IDE son provistas por módulos. Cada módulo provee una función bien definida, tales como el soporte de Java, edición, o soporte para el sistema de control de versiones. NetBeans contiene todos los módulos necesarios para el desarrollo de aplicaciones Java en una sola descarga, permitiéndole al usuario comenzar a trabajar inmediatamente. Desde Julio de 2006, es

licenciado bajo la Licencia Común de Desarrollo y Distribución (CDDL), una licencia basada en la Licencia Pública de Mozilla (MPL).

Se decide utilizar el IDE Eclipse fundamentalmente porque en el momento de inicio del proyecto este era el IDE más estable, se encontraba más establecido en el mercado, contaba con mayor comunidad de usuarios, y era uno de los más empleados en la universidad. Se puede además alegar los siguientes beneficios para este IDE. Soporta la construcción de una variedad de herramientas para el desarrollo de aplicaciones. Soporta el desarrollo de aplicaciones basadas en GUI y non-GUI. Se ejecuta en una gran cantidad de sistemas operativos incluyendo Windows y Linux. Es fácilmente integrable con la herramienta CASE Visual Paradigm y soporta perfectamente la plataforma de desarrollo J2EE.

1.6 PRINCIPALES FRAMEWORKS

Actualmente existen varios Frameworks que son de gran utilidad para el desarrollo de software que facilitan el trabajo en la capa de presentación y lógica de negocio, entre ellos están: Spring y Struts, los cuales poseen características que los hacen únicos para el desarrollo de estas capas.

1.6.1 SPRING

Spring es un framework de aplicación desarrollado para aplicaciones escritas en el lenguaje de programación Java. Se considera a Spring un framework lightweight (liviano o ligero), ya que no es una aplicación que requiere de muchos recursos para su ejecución, además el framework completo puede ser distribuido en un archivo .jar de aproximadamente 1 MB, lo cual representa muy poco espacio, y para la cantidad de servicios que ofrece es relativamente insignificante su tamaño.

Este framework está adquiriendo gran auge y una gran popularidad. Una de las características que ayuda a este éxito, es ser una aplicación open source (código abierto), lo cual implica que no tiene ningún costo, ni se necesita una licencia para utilizarlo, por lo tanto da la libertad a muchas empresas y desarrolladores a incursionar en la utilización de esta aplicación. Spring intenta integrar las diferentes tecnologías existentes, en un sólo framework para el desarrollo más sencillo y eficaz de aplicaciones

J2EE portables entre servidores de aplicación. Otro de los principales enfoques de Spring y por el cual está ganando dicha popularidad es que simplifica el desarrollo de aplicaciones J2EE, al intentar evitar el uso de EJB, ya que como menciona Craig Walls en su libro Spring in Action, “En su estado actual, EJB es complicado porque EJB fue creado para resolver cosas complicadas, como objetos distribuidos y transacciones remotas. Y muchas veces aunque el proyecto no es lo suficientemente complejo, se utiliza EJB, contenedores de alto peso y otras herramientas que soportan un grado mayor de complejidad, como una solución a un proyecto. Con Spring, la complejidad de tu aplicación es proporcional a la complejidad del problema que se está resolviendo”. [6]

Spring fue creado basado en los siguientes principios:

- El buen diseño es más importante que la tecnología subyacente.
- Los JavaBeans ligados de una manera más libre entre interfaces es un buen modelo.
- El código debe ser fácil de probar.

Spring es un Framework para el desarrollo de aplicaciones para la plataforma Java, que interviene en todas las capas arquitectónicas de una aplicación J2EE, brinda soporte a varios frameworks de presentación, entre ellos Java Server Faces (JSF), brinda soporte a Hibernate integrándose con ellos para formar una poderosa herramienta para el desarrollo de software.

A pesar de que Spring no obliga a usar un modelo de programación en particular, se ha popularizado en la comunidad de programadores en Java. Por su diseño el Framework ofrece mucha libertad a los desarrolladores en Java y soluciones muy bien documentadas y fáciles de usar, prácticas comunes en la industria. La simplificación del desarrollo de aplicaciones y de sus respectivas pruebas es una de las claves de su éxito. Spring puede organizar de forma efectiva los objetos de la capa central y manejar sus conexiones. Además puede eliminar la proliferación de solitarios y facilita unas buenas prácticas de programación orientada a objetos, por ejemplo utilizando interfaces. [7]

Arquitectura de Spring

Spring es un framework modular que cuenta con una arquitectura dividida en siete capas o módulos, lo cual permite tomar y ocupar únicamente las partes que interesen para el proyecto y juntarlas con gran libertad.

Principales módulos:

Spring Core: Esta parte es la que provee la funcionalidad esencial del framework, está compuesta por el BeanFactory, uno de los componentes principales del núcleo de Spring que utiliza el patrón de Inversión de Control (IoC, Inversion of Control) y configura los objetos a través de Inyección de Dependencia. Los paquetes org.springframework.beans y org.springframework.context proporcionan la base para el contenedor IoC (Spring Framework's IoC container). En el primer paquete está la interfaz BeanFactory, que proporciona la capacidad de gestionar cualquier tipo de objeto. En el segundo paquete se encuentra la subinterfaz ApplicationContext, construido sobre la base del anterior. La implementación más utilizada de BeanFactory es la clase XmlBeanFactory, la cual toma de un archivo XML la definición de instancias o beans, así como sus dependencias. Existen diferentes tipos de IoC como:

- Elevación de dependencia, en el que el objeto cliente para acceder al servicio primero lo identifica y después lo localiza por medio de una referencia y a continuación hace la llamada al procedimiento. El inconveniente es el acoplamiento entre la capa cliente, la capa de acceso al servicio y la implementación del servicio.
- Inyección de dependencia, en la que el servicio se identifica y localiza por medio de mecanismos no programáticos, externos al código, como por ejemplo un archivo XML. Las dependencias con respecto a los servicios son explicitas y no están en el código. Con ello se gana en facilidad de test y mantenimiento. Algunas formas de inyección de dependencia son: Inyección por medio del método setter que utiliza el framework para inyectar las propiedades de las clases y por medio de constructor creando un bean con los argumentos del constructor.

Spring Context: es un archivo de configuración que provee de información contextual al framework general. Además provee servicios enterprise como JNDI, EJB, email, validación y funcionalidad de

agenda. Brinda funcionalidades de localización y reconocimiento automático de las definiciones de los Beans, búsqueda de mensajes para encontrar su origen, cargar múltiples contextos y acceso a recursos.

Spring AOP: Aspect-oriented programming, o AOP, es una técnica que permite a los programadores modularizar ya sea las preocupaciones, o el comportamiento que corta a través de las divisiones de responsabilidad, como logging, y manejo de transacciones. El núcleo de construcción es el aspect, que encapsula comportamiento que afecta a diferentes clases, en módulos que pueden ser reutilizados. AOP se puede utilizar para persistencia, manejo de transacciones, seguridad, logging y debugging.

Spring ORM: En lugar de que Spring proponga su propio módulo ORM (Object-Relational Mapping), para los usuarios que no se sientan confiados en utilizar simplemente JDBC, propone un módulo que soporta los frameworks ORM más populares del mercado, entre ellos: Hibernate, iBATIS, Apache OJB, entre otros. Spring al combinarse con alguna herramienta ORM brinda ventajas como el manejo de sesión, de recursos, de transacciones integrado y facilidad de prueba.

Spring DAO: El patrón DAO (Data Access Object) es uno de los patrones más importantes y usados en aplicaciones J2EE, y la arquitectura de acceso a los datos de Spring provee un buen soporte para este patrón.

Spring Web: El módulo web de Spring se encuentra en la parte superior del módulo de contexto, y provee el contexto para las aplicaciones web. Este módulo también se encarga de diversas operaciones web como por ejemplo: las peticiones multi-parte que puedan ocurrir al realizar cargas de archivos y la relación de los parámetros de las peticiones con los objetos correspondientes (domain objetcs o business objects).

Spring MVC: Spring brinda un MVC (Model View Controller, Modelo Vista Controlador) para web bastante flexible y altamente configurable, pero esta flexibilidad no le quita sencillez, ya que se pueden desarrollar aplicaciones sencillas sin tener que configurar muchas opciones. MVC está basado en

interfaces, provee interceptores al igual que controladores, estos son configurados de la misma manera que los demás objetos en Spring, a través de IoC.

Spring cuenta con una gran cantidad de controladores de los cuales se puede elegir dependiendo de la tarea, entre los más populares se encuentra: Controller y AbstractController para tareas sencillas; el SimpleFormController ayuda a controlar formularios y el envió de los mismos, MultiActionController ayuda a tener varios métodos dentro un solo controlador a través del cual se podrán mapear las diferentes peticiones a cada uno de los métodos correspondientes. [6]

Spring provee del mecanismo HTTP invoker para la implementación de aplicaciones remotas de forma transparente para el código de la aplicación. El protocolo HTTP invoker usa la serialización Java de la misma forma en la que lo hace RMI (Remote Method Invocation, Invocación de Método Remoto), pero su configuración y despliegue es tan sencilla como en los protocolos Hessian y Burlap del framework Caucho. El protocolo HTTP invoker es más potente y flexible que Hessian y Burlap pero manteniendo la sencillez de configuración de estos, lo que le confiere una ventaja respecto a RMI.

Creación de un servicio mediante HTTP invoker

La creación de un servicio mediante HTTP invoker del framework Spring consta de los siguientes pasos:

- Creación del interfaz del servicio y de la implementación del mismo. Este paso se realiza del mismo modo que para los protocolos Hessian y Burlap.
- Exportación del servicio mediante la definición de un bean de Spring que permitirá acceder al mismo.
- Despliegue del servicio en un contenedor de servlets para publicar los métodos como servicios Http.
- Creación del bean de Spring para el cliente, referenciando la dirección donde se encuentra el servicio publicado que se desea consumir.
- Invocación del servicio. [19]

Como ventajas destaca el uso del mecanismo de serialización de Java, lo que soluciona el problema que presentan Hessian y Burlap en este aspecto. Asimismo, es destacable la sencillez del protocolo para la configuración y despliegue de servicios. HTTP invoker resulta apropiado para aplicaciones Java desarrolladas con el framework Spring en las que se desea evitar los inconvenientes de los protocolos Hessian y Burlap.

1.6.2 STRUTS

Es una herramienta de soporte para el desarrollo de aplicaciones Web bajo el patrón MVC bajo la plataforma J2EE (Java 2 Enterprise Edition). Struts se desarrollaba como parte del proyecto Jakarta de la Apache Software Foundation, pero actualmente es un proyecto independiente conocido como Apache Struts. Permite reducir el tiempo de desarrollo. Su carácter de "software libre" y su compatibilidad con todas las plataformas en que Java Entreprise esté disponible, lo convierte en una herramienta altamente disponible. Se encarga de normalizar el desarrollo de la capa vista dentro de la arquitectura MVC. También proporciona mecanismos para trabajar con la capa controller, pero nunca se mezcla con la capa de modelo. Esta característica principal de Struts permite separar la lógica de presentación de la lógica de negocio. [12]

Struts es un framework open source (código abierto) creado por Craig R. McClanahan y donado al Apache Software Fundation en el año 2002 quién lo acogió como parte de su proyecto Jakarta. Tiene como objetivo ayudar a los desarrolladores a construir aplicaciones web de una forma rápida y fácil. Struts se apoya en un grupo de tecnologías estándar de JEE como los JavaBeans, Java Servlets, y Java Server Pages (JSP) [8]. Este framework posee un Servlet controlador que se encarga de manejar el flujo entre la JSP (Java Server Page) y otros dispositivos de la capa de presentación, apoyándose en el uso de los ActionForwards y ActionMappings para el control del flujo de decisiones fuera de la capa de presentación.

Algunos de los elementos que brinda Struts para la implementación de su arquitectura son:

- Action: es una parte del controlador que gestiona los cambios de estados, invoca e interactúa con la lógica del negocio. Forma parte del controlador en el modelo MVC. Implementa el patrón de diseño Front Controller (Controlador Frontal).

- ActionForward: se encarga de la selección de las interfaces del usuario y el redireccionamiento. Maneja el flujo de trabajo de una aplicación.
- ActionForm: gestiona el intercambio de datos entre las Java Server Pages y el controlador.
- ActionMapping: encapsula la información de mapeo entre las vistas y el controlador, permite el direccionamiento de cada solicitud hasta la acción indicada como su respectiva respuesta. Se encuentra ubicado dentro del controlador en el modelo MVC.
- ActionServlet: recibe las solicitudes, las empaqueta y las enruta hacia el manejador adecuado.
- Tiles: permiten utilizar las plantillas de Struts como mecanismo de construcción de vistas compuestas. Forma parte de la vista en el modelo MVC. Implementa el patrón de diseño Composite View (Vista Compuesta).

Además de todos estos elementos que conforman el centro de la arquitectura, Struts brinda más opciones, algunas de ellas son:

- El framework validator (validador) permite configurar todas las validaciones de los datos que se obtienen en los formularios (ActionForms) tomando los mensajes de los Resources Bundles (Fichero que contiene el diccionario de los textos de la aplicación) y con la opción de tratarlas en el cliente.
- El fichero struts-config.xml permite configurar el flujo de la aplicación, apoya el redireccionamiento del control hacia cualquier recurso.
- Sistema de internacionalización de recursos, utilizando ficheros .properties (Resources Bundles), que permiten manejar varios idiomas en la aplicación haciendo uso de ellos dinámicamente.
- Sistema de manejo de excepciones que se registran en el fichero de configuración donde se direccionan las respuestas a los recursos correspondientes. [8,12]

Las debilidades de Struts se hacen sentir en grandes aplicaciones donde el desarrollo de las páginas puede tomarse lento, puesto que este framework carece de herramientas que permitan un rápido desarrollo de las interfaces web, además las librerías de etiquetas que trae pueden ser insuficiente para el desarrollo de complejas interfaces de usuario. Otra de las desventajas de Struts es que las clases que manejan los formularios y ejecutan las acciones están acopladas al framework ya que heredan de las clases ActionForm y Action.

Son muchas las ventajas y funciones que ofrecen ambos frameworks de desarrollo pero se escogió Spring debido a la facilidad que brinda para trabajar en las capas de presentación y lógica de negocio, y por la experiencia en el trabajo con este framework. Además tiene como objetivo facilitar la construcción de aplicaciones java brindando las siguientes ventajas:

- Permite la integración entre varios frameworks, constituyendo un eslabón central en la arquitectura de las aplicaciones.
- Trae integrado el framework Acegi, que provee un mecanismo de seguridad robusto y fácil de configurar permitiendo implementar las políticas de seguridad de manera transparente al código de la aplicación.
- Maneja los objetos de negocio de la aplicación con la técnica de inyección de instancias desde un fichero de configuración, manteniendo el código limpio y disminuyendo el acoplamiento.
- Contiene un módulo que permite programar orientado a aspectos que permite implementar funciones complejas que de realizarlas orientada a objetos serían extensas e intrusivas en el código de la aplicación.
- Funciona como un framework integrador que permite la comunicación e integración de las capas de presentación y lógica de negocio.

1.7 PRUEBAS DEL SOFTWARE

Esta etapa requiere de gran seriedad y tiempo por parte del equipo que realizará las pruebas, su objetivo fundamental es probar la aplicación con el fin de encontrar errores. La etapa de pruebas se considera un proceso técnico de investigación que requiere de profesionales altamente calificados en lenguajes de desarrollo, métodos, técnicas de pruebas y herramientas especializadas. El conocimiento que debe manejar un ingeniero de prueba es muchas veces superior al del desarrollador de software.

La prueba ideal que se le podría realizar al sistema, sería exponerlo en todas las situaciones posibles, así se encontraría hasta el último fallo e indirectamente, se garantiza su respuesta ante cualquier caso que se le presente en la ejecución real, pero esto es imposible desde el punto de vista humano y técnico. Para probar el sistema se sometió a una serie de pruebas con elementos válidos e inválidos, teniendo en cuenta los casos de prueba.

Tanto para la realización de verificaciones como de validaciones se pueden utilizar distintos tipos de técnicas. En general, estas técnicas se agrupan en dos categorías:

Técnicas de Evaluación Estáticas: Buscan faltas sobre el sistema en reposo. Esto es, estudian los distintos modelos que componen el sistema software buscando posibles faltas en los mismos. Así, estas técnicas se pueden aplicar, tanto a requisitos como a modelos de análisis, diseño y código.

Técnicas de Evaluación Dinámicas: Generan entradas al sistema con el objetivo de detectar fallos, cuando el sistema ejecuta dichas entradas. Los fallos se observan cuando se detectan incongruencias entre la salida esperada y la salida real. La aplicación de técnicas dinámicas es también conocida como pruebas de software o testing y se aplican generalmente sobre código puesto que es, hoy por hoy, el único producto ejecutable del desarrollo.

Para realizar las pruebas al producto es necesario diseñar primeramente los casos de prueba, el diseño de un caso de prueba para comprobar la calidad del software requiere un esfuerzo significante cerca del 40 % del tiempo desarrollo, un caso de prueba es bueno cuando su ejecución conlleva una

alta probabilidad de encontrar un error. El éxito de la prueba se mide en función de la capacidad de detectar un error que estaba oculto [21].

Durante el desarrollo del software se realizarán una serie de técnicas de verificación y validación como las pruebas de caja blanca y pruebas de caja negra

1.7.1 PRUEBAS DE CAJA BLANCA

Técnicas de caja blanca o estructural, que se basan en un minucioso examen de los detalles de los procedimientos del código a evaluar, por lo que es necesario conocer la lógica del programa.

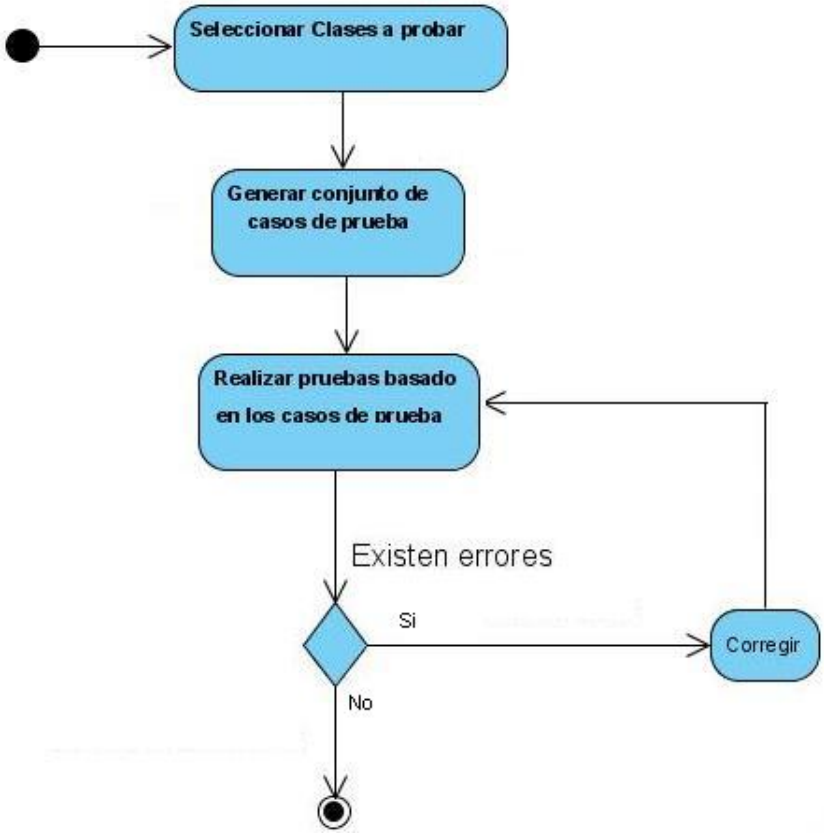

Figura 1.3: Ciclo para las pruebas de caja blanca.

Las pruebas de caja blanca se realizarán con la ayuda del framework JUnit.

1.7.1.1 HERRAMIENTA PARA REALIZAR PRUEBAS DE CAJA BLANCA

JUnit es framework creado por Erich Gamma y Kent Beck, que permite realizar 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. Es decir, en función de algún valor de entrada se evalúa el valor de retorno esperado; si la clase cumple con la especificación, entonces JUnit devolverá que el método de la clase pasó exitosamente la prueba; en caso de que el valor esperado sea diferente al que regresó el método durante la ejecución, JUnit devolverá un fallo en el método correspondiente.

JUnit es también un medio de controlar las pruebas de regresión, necesarias cuando una parte del código ha sido modificado y se desea ver que el nuevo código cumple con los requerimientos anteriores y que no se ha alterado su funcionalidad después de la nueva modificación.

El propio framework incluye formas de ver los resultados que pueden ser en modo texto, gráfico (AWT o Swing) o como tarea en Ant (herramienta para la realización de tareas).

En la actualidad las herramientas de desarrollo como NetBeans y Eclipse cuentan con plugins que permiten que la generación de las plantillas necesarias para la creación de las pruebas de una clase Java se realice de manera automática, 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.

Actualmente este framework se encuentra en la versión 4.5, que posee grandes mejoras a sus versiones anteriores como:

- Inclusión de anotaciones (Java 5 annotations) en lugar de utilizar herencia.
- Se utiliza @Test para sustituir a la herencia de TestCase.
- Se utiliza @Before y @After como sustitutos de setUp y tearDown.
- Añade @Ignore para deshabilitar tests.
- Permite timeouts en los tests.

- Configuración de excepciones esperadas.
- Ordenación, priorización, categorización y filtrado de tests.
- Permite el avance (Forward) y retroceso (backward) compatibilidad.
- Permite la autenticación (Logging).
- Se elimina la distinción entre errores y fallos. [17]

Spring se integra perfectamente con JUnit. JUnit por sí mismo no soporta la inyección de dependencia, Spring soluciona este problema de forma muy fácil y sencilla, en vez de heredar TestCase se deberá heredar de AbstractDependencyInjectionSpringContextTests o del AbstractTransactionalDataSourceSpringContextTests. La primera simplemente soporta inyección de dependencias mientras que el segundo crea una transacción por cada uno de los métodos que accedan a la BD y hace un retroceso (Rollback) al final de cada test, pero obliga a declarar una cadena de conexión (Data Source) en el contexto de la aplicación, además da acceso a la plantilla de JDBC (template de JDBC) para poder lanzar consultas y compararlas con los resultados de los métodos de la lógica de negocio [18].

1.7.2 PRUEBAS DE CAJA NEGRA

Las pruebas de Caja Negra o pruebas Funcionales se realizan sobre la interfaz del software, entendiendo por interfaz las entradas y salidas del mismo. No es necesario conocer su lógica de funcionamiento, únicamente la funcionalidad que debe realizar.

También conocidas como Pruebas de Comportamiento, estas pruebas se basan en la especificación del programa o componente a ser probado para elaborar los casos de prueba. El componente se ve como una “Caja Negra” cuyo comportamiento sólo puede ser determinado estudiando sus entradas y las salidas obtenidas a partir de ellas. No obstante, como el estudio de todas las posibles entradas y salidas de un programa sería impracticable se selecciona un conjunto de ellas sobre las que se realizan las pruebas. Para seleccionar el conjunto de entradas y salidas sobre las que trabajar, hay

que tener en cuenta que en todo programa existe un conjunto de entradas que causan un comportamiento erróneo en nuestro sistema, y como consecuencia producen una serie de salidas que revelan la presencia de defectos. Entonces, dado que la prueba exhaustiva es imposible, el objetivo final es pues, encontrar una serie de datos de entrada cuya probabilidad de pertenecer al conjunto de entradas que causan dicho comportamiento erróneo sea lo más alto posible [21].

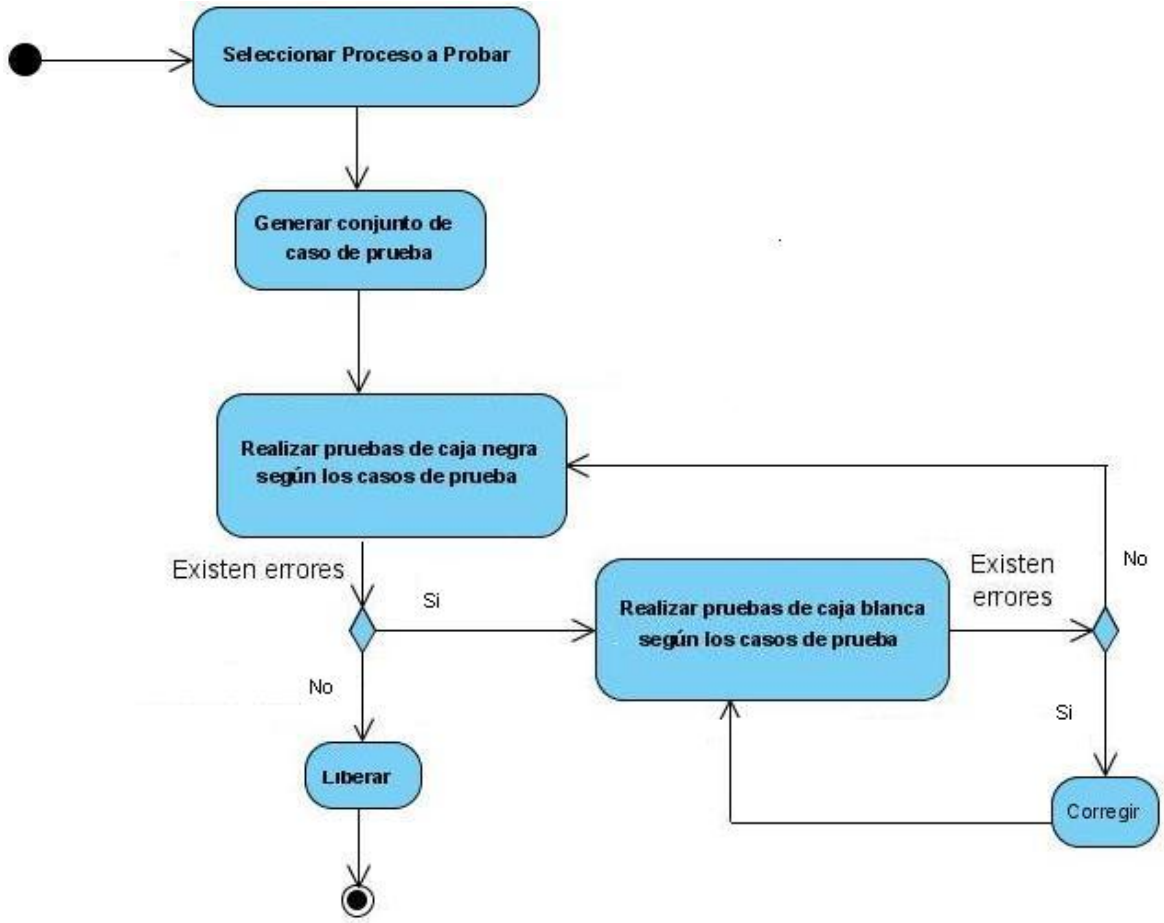

Figura1.4: Ciclo de prueba de caja negra.

1.8 CONCLUSIONES

Se puede concluir que:

- La Arquitectura en capas brinda grandes ventajas a la hora de elaborar un software ya que al tener las capas separadas existe poco acoplamiento entre las mismas y garantiza mejoras como la mantenibilidad, extensibilidad y reutilización de componentes.

- RUP es una metodología que define claramente las actividades realizadas por roles generando a su paso artefactos que sustentan el proceso de construcción de software.

- La plataforma J2EE provee un conjunto de librerías, interfaces y frameworks que brindan una infraestructura para el desarrollo, abarcando las transacciones, la mensajería, las conexiones de base de datos, interfaces de usuario, manejo de recursos y la seguridad de las aplicaciones.

- Eclipse es un entorno integrado de desarrollo multiplataforma que brinda la posibilidad de crear cualquier tipo de aplicaciones clientes.

- El framework Spring facilita el desarrollo de buenas prácticas de programación, posibilitando integrar todas las capas necesarias para desarrollar un software.

- JUnit es un framework que posibilita realizar pruebas de caja blanca al software comprobando si el código escrito es correcto sin tener que implementar la interfaz de la aplicación.

CAPÍTULO 2: DESCRIPCIÓN Y ANÁLISIS DE LA PROPUESTA DE SOLUCIÓN

2.1 INTRODUCCIÓN

En el presente capítulo se tratarán los aspectos fundamentales para darle solución al objetivo trazado, abordando los principales procesos del módulo Seguimiento de forma tal que faciliten la comprensión del sistema a desarrollar. Se representará el estilo arquitectónico utilizado en el diseño y se realizará el modelo de implementación. Se hará énfasis en la estructura del sistema para comprender cómo deben estar organizados los diferentes archivos y paquetes, definiendo la estrategia de seguridad del mismo y el estándar de codificación que se debe seguir para la implementación.

Se profundizará en el trabajo realizado en las capas de presentación y lógica de negocio para darle cumplimiento a los requisitos funcionales. Se describirán las clases implementadas en las capas así como las operaciones relacionadas con cada clase, y se definirá la estrategia utilizada para integrar ambas capas configurando los diferentes archivos que utiliza el framework Spring.

2.2 DESCRIPCIÓN DEL SISTEMA

Para darle cumplimiento al objetivo trazado fue necesario realizar un estudio detallado de las funcionalidades reflejadas en el modelo de casos de uso del sistema que se deseaba implementar. A continuación se muestra la vista global de los procesos del módulo Seguimiento que necesitan ser automatizados.

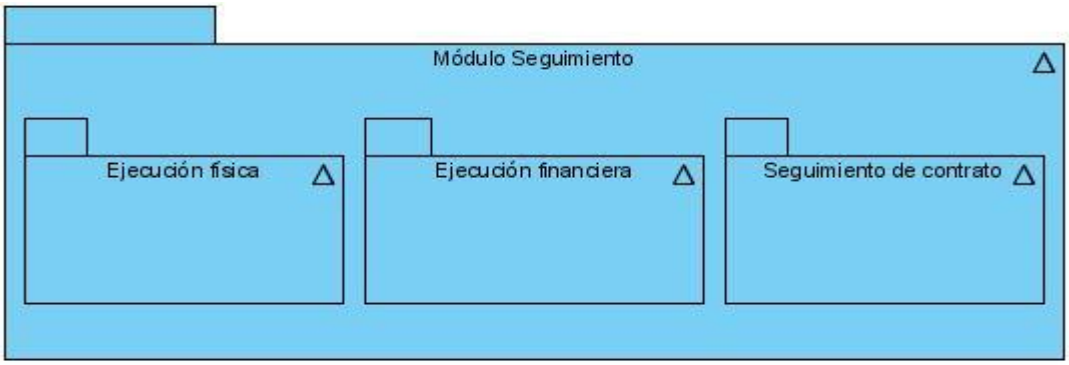

Figura 2.1: Vista global de procesos.

En el paquete Ejecución física se encuentran las funcionalidades que permiten darle seguimiento a la ejecución física de los proyectos permitiendo iniciar un proyecto en la fecha indicada en su concepción, terminar un proyecto, actualizar la ejecución física de cada una de las actividades del proyecto indicando una observación de la ejecución de las mismas, gestionar las observaciones de un proyecto, replanificar el plan operativo de un proyecto en el cual se pueden gestionar todas las actividades asociadas a este, así como los recursos de cada actividad, reformular una ficha de proyecto que incluye replanificar el plan operativo y modificar los datos básicos de la ficha. Además está la funcionalidad de revisar el inicio, terminación y actualización de la ejecución física de un determinado proyecto.

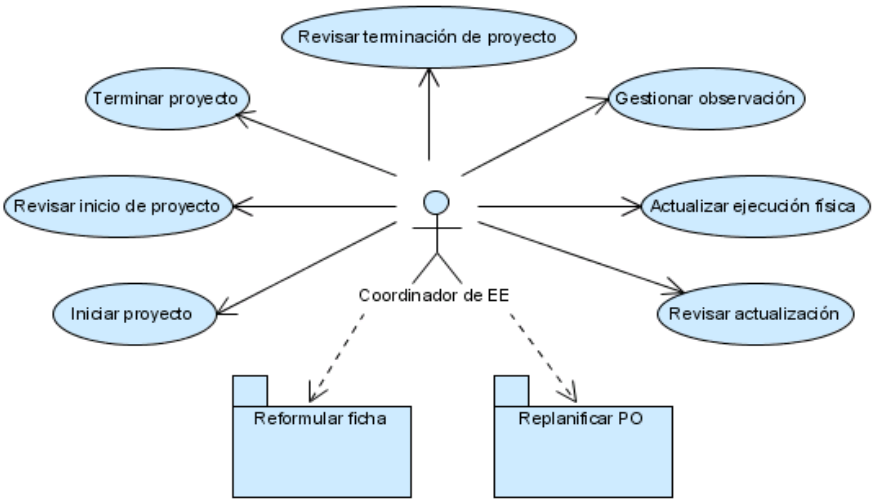

Figura 2.2: Vista de las funcionalidades del paquete Ejecución física.

En el paquete Ejecución financiera están las funcionalidades que permiten darle seguimiento a la ejecución financiera de los proyectos brindando la posibilidad de gestionar las solicitudes de pagos realizadas por los entes ejecutores tanto de los cubanos como de los venezolanos, gestionar las facturas de cada pago, confirmar un pago de una solicitud y revisar una solicitud de pago. En el paquete Replanificar Cronograma de ejecución financiera se encuentran las funcionalidades que permiten distribuir el monto de los proyectos a cobrar en cada uno de los meses que requiere un proyecto para su ejecución.

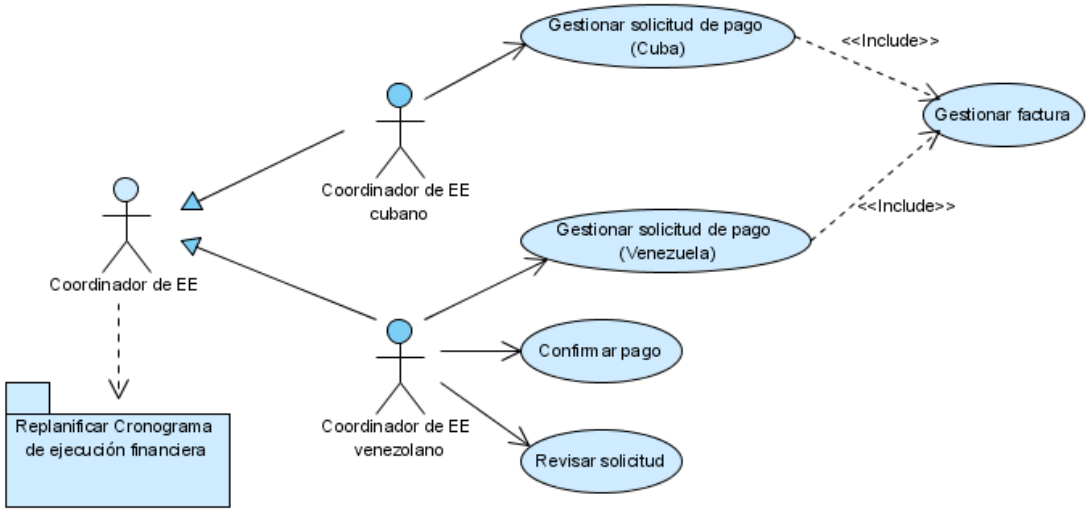

Figura 2.3: Vista de las funcionalidades del paquete Ejecución financiera.

En el paquete Seguimiento de contrato se encuentran las funcionalidades necesarias para darle seguimiento a los contratos firmados dando la posibilidad de modificar un contrato, descargar el documento de un contrato, revisar la modificación de un contrato por todos los involucrados (Entes, Ministerios y Secretarías Técnicas), gestionar el envío a un nivel superior (Ministerios o Secretarías Técnicas) y firmar la modificación de un contrato por parte de los Ministerios.

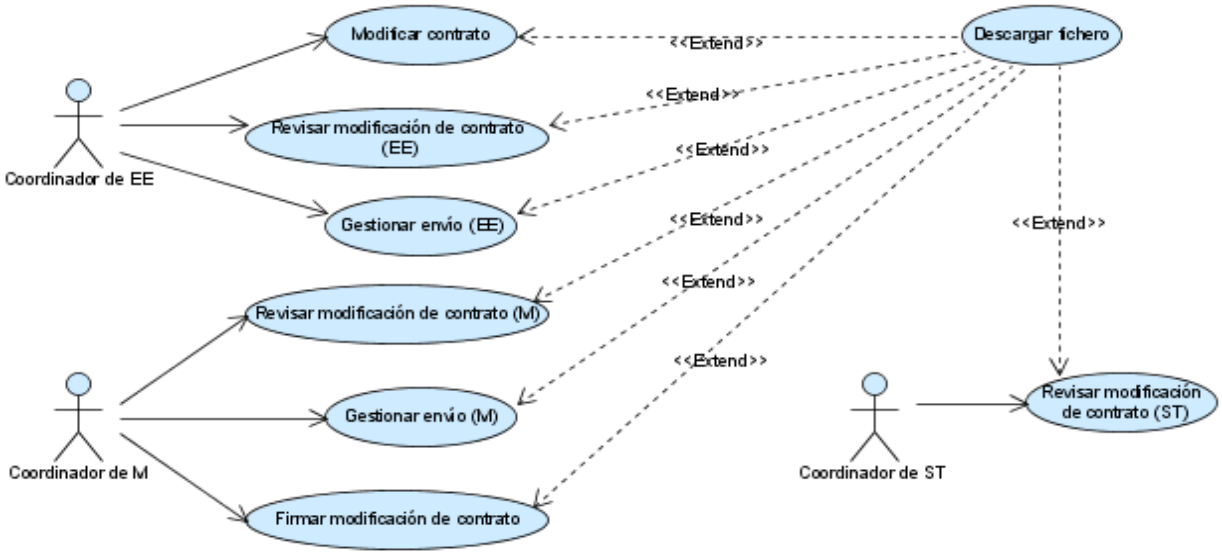

Figura 2.4: Vista de las funcionalidades del paquete Seguimiento de contrato.

Con la descripción anterior se pudo conseguir una comprensión más precisa de los requisitos que debía cumplir el sistema lo que contribuye al mejor entendimiento para el diseño de las funcionalidades que se necesitan implementar.

2.3 DESCRIPCIÓN DEL DISEÑO

Se diseñó la arquitectura teniendo en cuenta el estilo arquitectónico Arquitectura en capas, el cual permitió organizar la implementación del sistema en 4 capas que consumen los recursos de la capa de datos, las cuales se muestran a continuación:

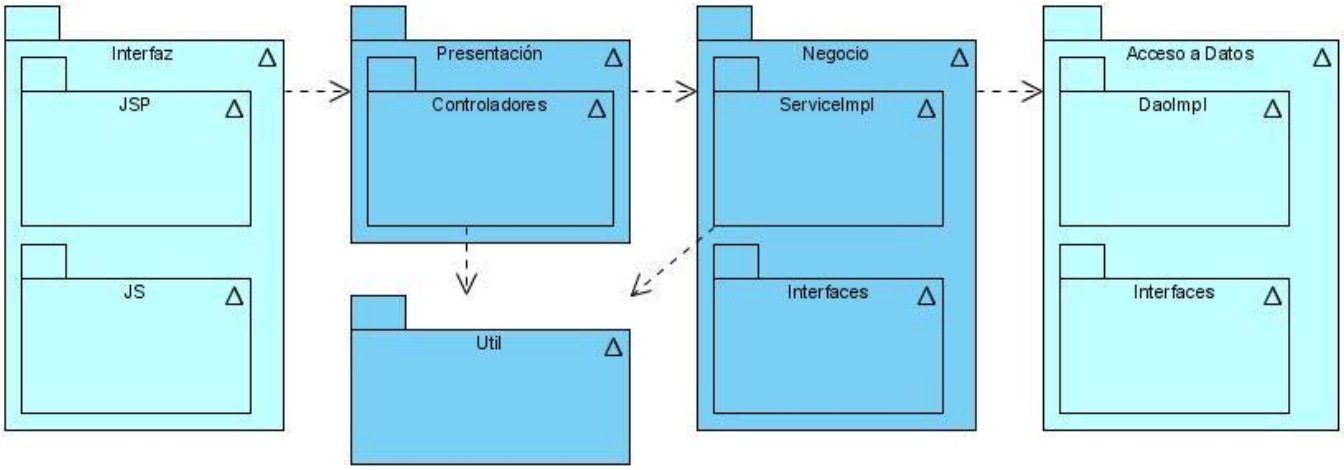

Figura 2.5: Organización de las capas en paquetes.

Este estilo posibilitó implementar las reglas del negocio en una capa aparte, para que estas reglas puedan ser usadas por otros sistemas o servicios que necesiten estas funcionalidades, evitando la duplicación de código y brindando una mejor organización de las funciones y clases de cada capa. Esto posibilita que se puedan modificar las reglas del negocio sin tener que modificar los servicios que las interfaces brindan a la capa superior.

A continuación se muestra el modelo de diseño dividido en tres subsistemas, Ejecución física, Ejecución financiera y Seguimiento de contrato, que muestran las relaciones, propiedades y responsabilidades de las clases necesarias para implementar las capas de presentación y lógica de negocio:

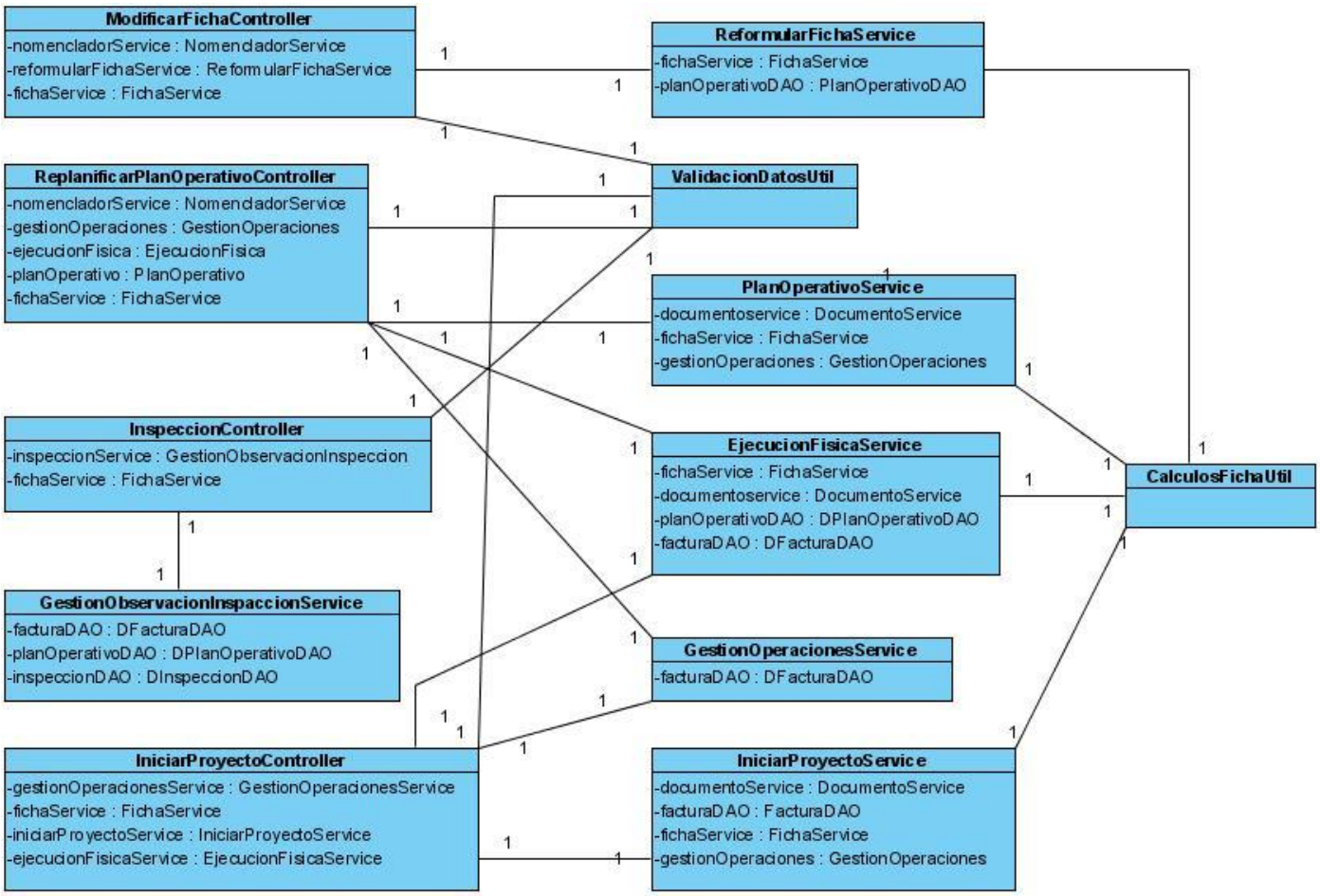

Figura 2.6: Clases de diseño del subsistema Ejecución física

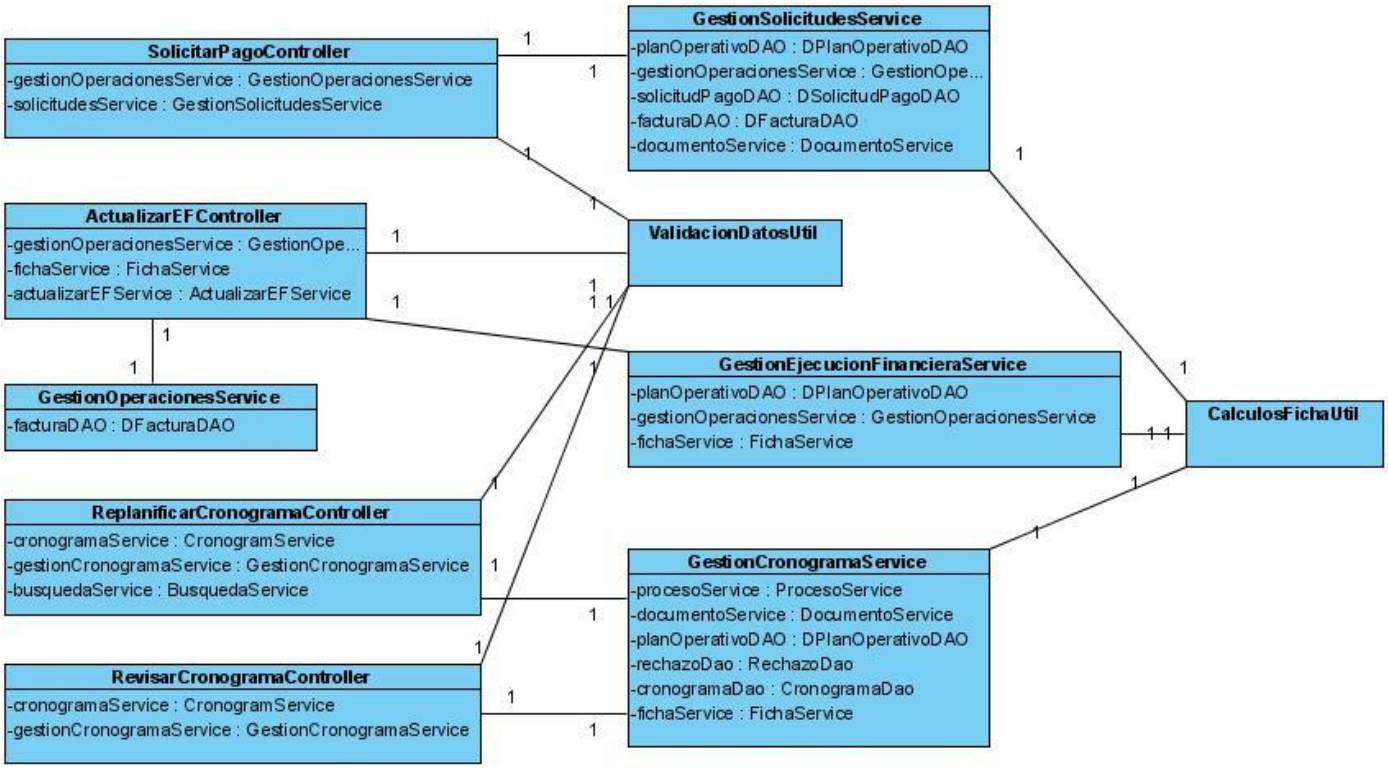

Figura 2.7: Clases de diseño del subsistema Ejecución financiera

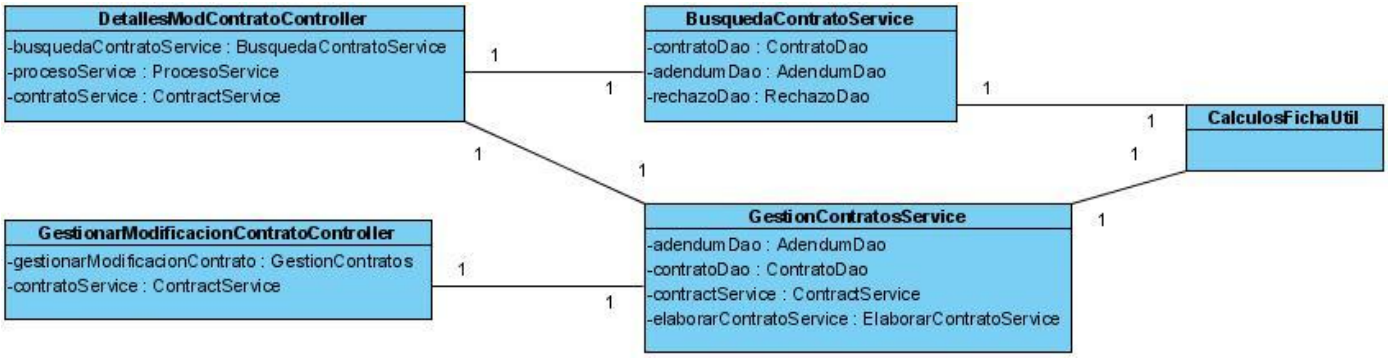

Figura 2.8: Clases de diseño del subsistema Seguimiento de contrato.

El diseño realizado describe como implementar ambas capas del módulo Seguimiento, obteniendo las clases principales que deben ser definidas para que el sistema funcione satisfactoriamente, así como los atributos y métodos que darán respuesta a las funciones que presenta el sistema y de esta forma tener una visión general de las responsabilidades que se deben implementar en cada clase.

El diseño fue elaborado siguiendo patrones basados en la experiencia, que de manera general constituyen soluciones simples y elegantes a problemas específicos y comunes del diseño orientado a objetos, permitiendo llevar a cabo la implementación clara del módulo bajo patrones como los GRASP (General Responsability Assignment Software Patterns, Patrones Generales de Software para Asignar Responsabilidades) y los GOF (Gang of Four, Pandilla de los Cuatro). Se utilizaron los patrones: experto, creador, alta cohesión, bajo acoplamiento y controlador. Estos patrones se les aplicaron a las clases definidas en el diseño, para distribuir las responsabilidades entre las mismas de forma tal que no existan muchas relaciones y que no se sobrecargue de métodos a una clase en específico pudiendo acomodarlos en otras.

Los patrones GOF generalmente se evidencian en clases que son creadas debido al uso de un patrón en específico. Existe un grupo de patrones de este tipo definidos para el diseño de clases, con el propósito de crear una arquitectura robusta para el sistema. Del gran número de patrones propuestos por la Pandilla de los Cuatro se utilizaron los siguientes:

- Prototipo: crea nuevos objetos clonándolos de una instancia ya existente, fue necesario para realizar una copia de un objeto en momentos determinados como a la hora de modificar un proyecto o replanificar el cronograma de un proyecto.

- Fabricación Pura: asigna un conjunto altamente cohesivo de responsabilidades a una clase artificial que no representa nada en el dominio del problema, una clase creada para dar soporte a una alta cohesión, un bajo acoplamiento y reutilización, ejemplo de esto son clases del negocio que se utilizaron para realizar operaciones y validaciones que se requerían en la capa de presentación.

- Facade (Fachada) como su nombre lo indica, es una interfaz la cual actúa de mediación entre dos capas en la aplicación, esta interfaz se empleó para exponer a la capa de presentación los métodos que esta necesita, sin tener que comunicarlas directamente.

2.4 MODELO DE COMPONENTES

A partir del diagrama de clases de diseño, se procede a realizar el diagrama de componentes; un componente es una parte física que se encuentra en la computadora como por ejemplo librerías, ejecutables, archivos de datos, entre otros. Con la realización del diagrama de componentes se tiene una aproximación de las relaciones que tendrán los componentes del sistema. A continuación se muestran los diagramas de componentes por cada uno de los subsistemas definidos para implementar las capas de presentación y lógica de negocio del módulo Seguimiento:

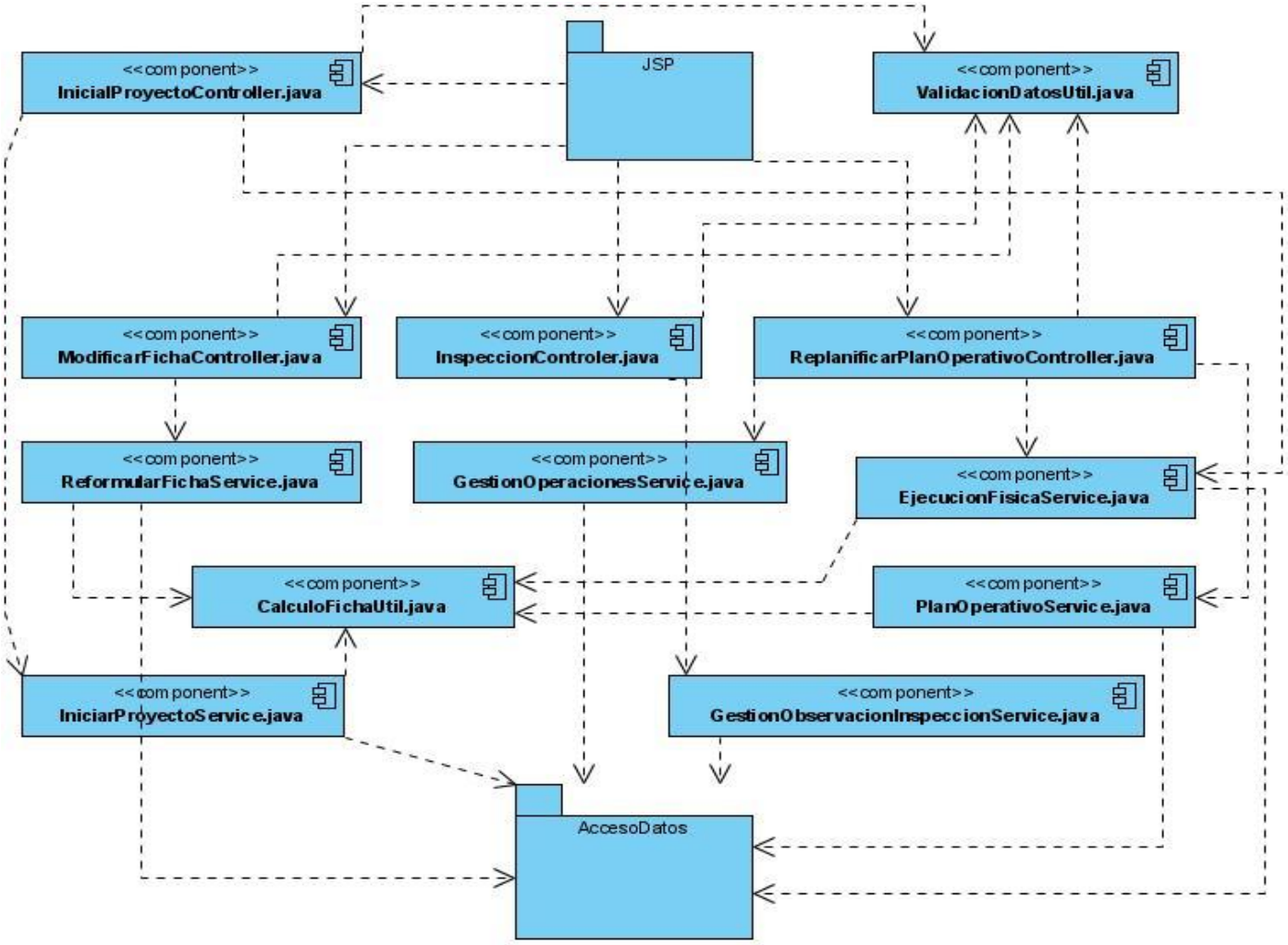

Figura 2.9: Diagrama de componentes del subsistema Ejecución física.

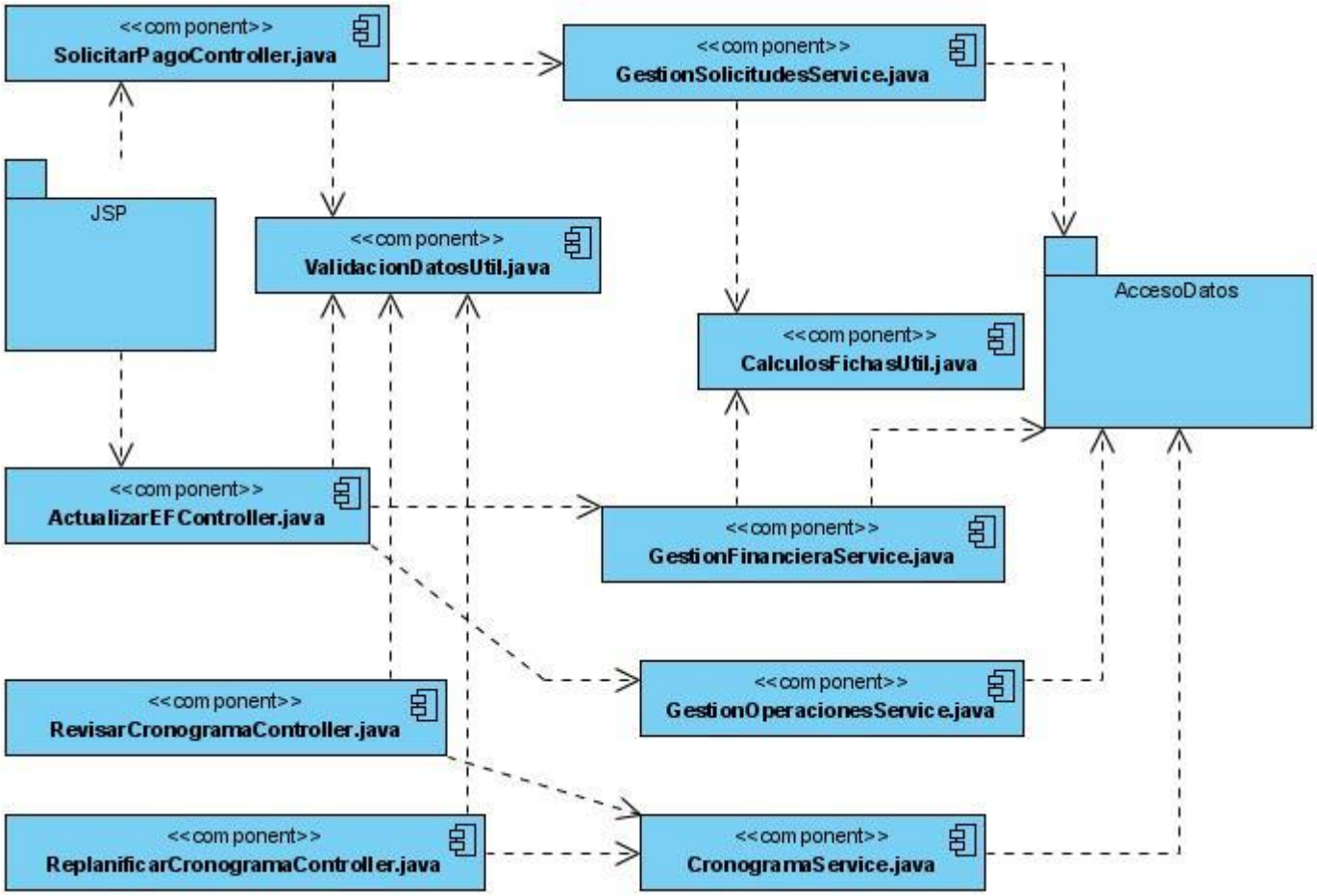

Figura 2.10: Diagrama de componentes del Subsistema Ejecución financiera.

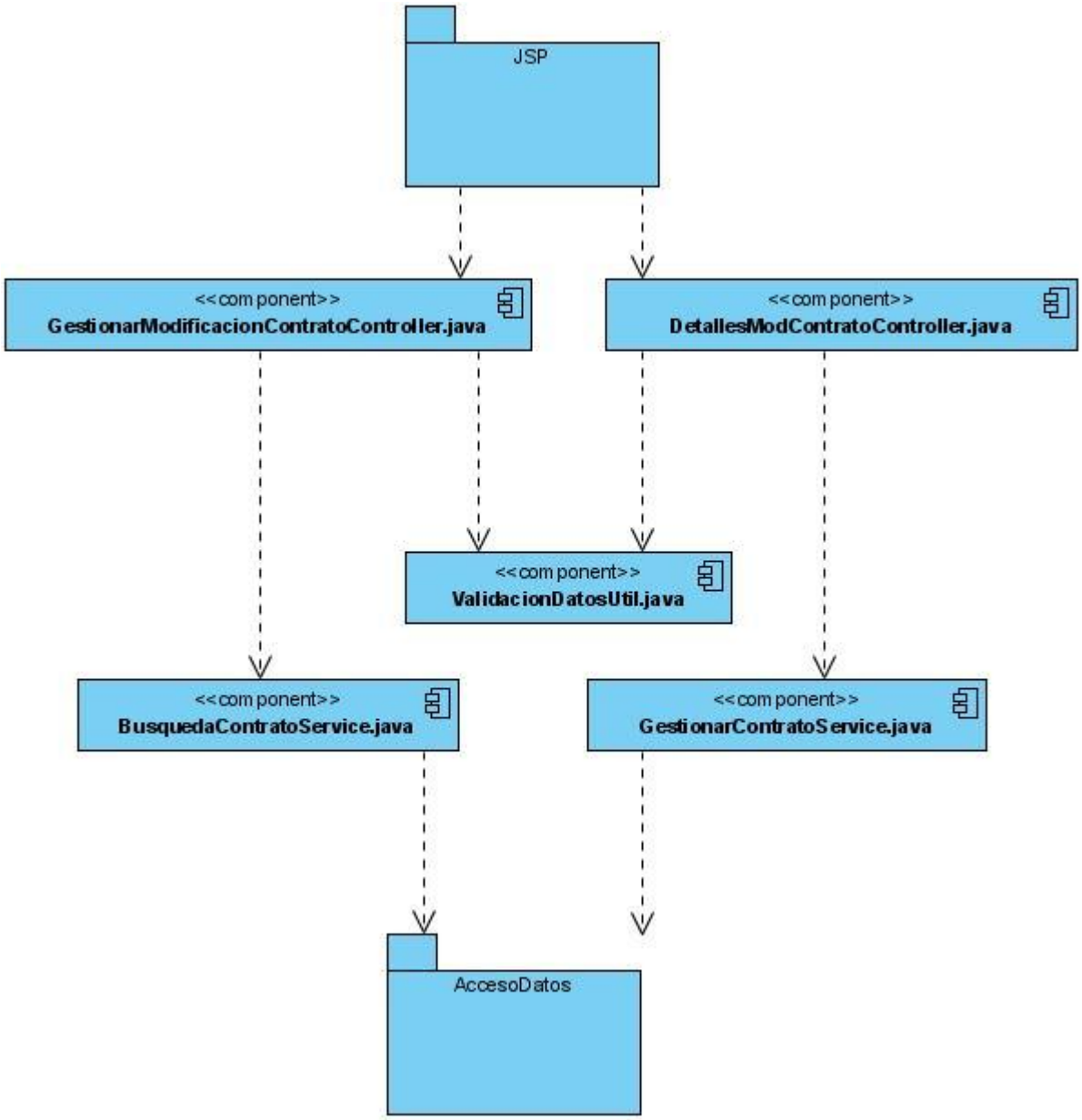

Figura 2.11: Diagrama de componentes del Subsistema Seguimiento de contrato.

2.5 ESTRUCTURA Y SEGURIDAD PARA EL DESARROLLO DEL SISTEMA

Para garantizar la organización y agilizar el desarrollo de la aplicación fue necesario separar el sistema en varios módulos en dependencia de las funcionalidades de cada uno, los cuales serían integrados a medida que se avanzara en la implementación. A continuación se muestra cómo estarán estructurados los paquetes del sistema y la estrategia a seguir para garantizar la seguridad de la aplicación.

2.5.1 CONVENCIONES DE ARCHIVOS Y PAQUETES

El proyecto Web que se creará en el Eclipse para el desarrollo del sistema será "CCV", al igual que la URL para acceder al mismo. Todas las clases, ficheros XML, páginas JSP y otros ficheros del sistema estarán agrupados en una estructura de carpetas y paquetes.

Las clases y ficheros de recursos se encontrarán dentro de la carpeta **src**, haciendo uso de las convenciones de nombrado de paquetes y adaptándolas al sistema quedaría como paquete principal: ***cu.uci.ccv.***

A partir de este nivel se agregaría la separación por subsistemas o módulos:
cu.uci.ccv.administracion (Subsistema Configuración y Administración)
cu.uci.ccv.common (Subsistema que contendría las clases y componentes comunes para el resto de los subsistemas)
cu.uci.ccv.contratacion (Subsistema Contratación de Proyectos)
cu.uci.ccv.financiamiento (Subsistema Financiamiento de Proyectos)
cu.uci.ccv.misiones (Subsistema Misiones)
cu.uci.ccv.presentacion (Subsistema Presentación de Proyectos)
cu.uci.ccv.seguimiento (Subsistema Seguimiento de Proyectos)

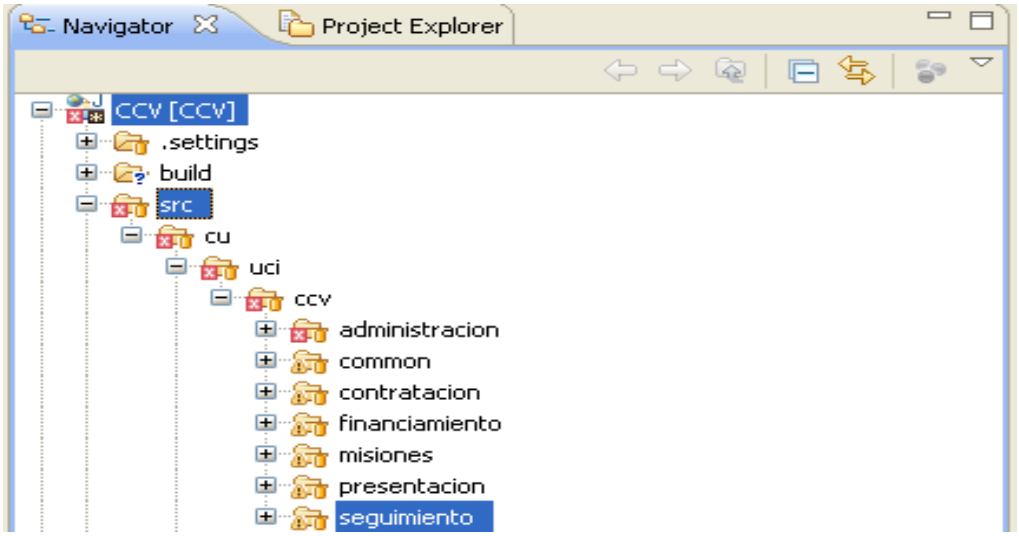

Figura 2.12 Organización de paquetes y subsistemas.

Dentro de cada subsistema se agregaría la separación de las clases por capas de cada uno de ellos, en este caso se muestra como se vería en el subsistema Seguimiento de Proyectos.

cu.uci.ccv.seguimiento.bussines (Interfaces de la capa de lógica de negocio, donde se exponen cada uno de los métodos de esta capa)

cu.uci.ccv.seguimiento.bussines.impl (Clases de la capa de lógica de negocio, las cuales implementan las interfaces)

cu.uci.ccv.seguimiento.dao (Interfaces de la capa de acceso a datos, donde se exponen cada uno de los métodos de esta capa)

cu.uci.ccv.seguimiento.dao.impl (Clases de la capa de acceso a datos, las cuales implementan las interfaces utilizando Spring-Hibernate)

cu.uci.ccv.seguimiento.reportes (Clases controladoras y entidades necesarias para los reportes)

cu.uci.ccv.seguimiento.vo (Entidades del modelo de datos, conjunto con los ficheros de mapeo objeto relacional de Hibernate)

cu.uci.ccv.seguimiento.web (Clases de la capa de presentación)

cu.uci.ccv.seguimiento.util (Clases útiles para el manejo de validaciones, datos y cálculos auxiliares)

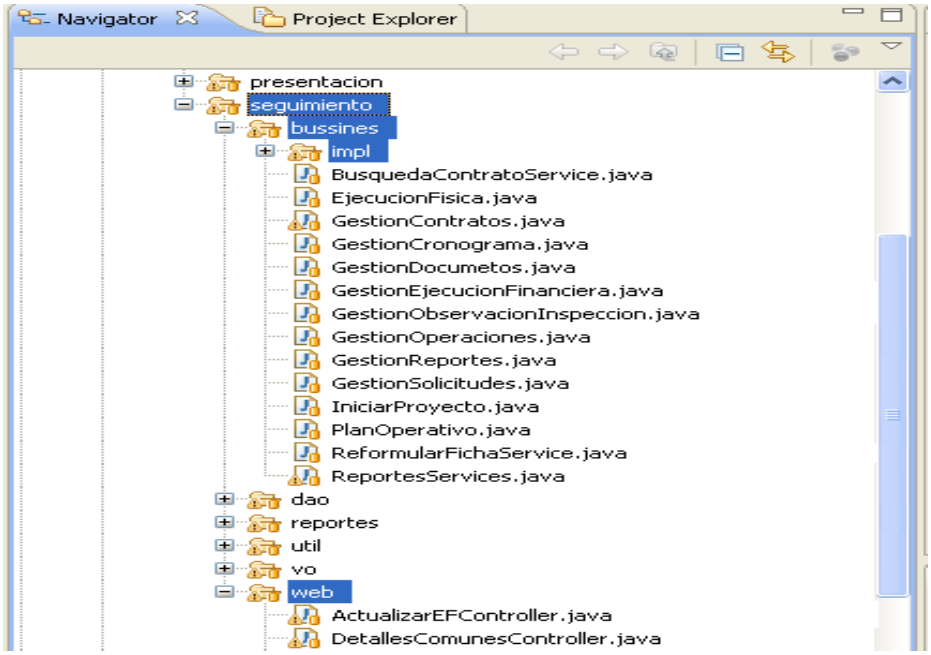

Figura 2.13 Separación de las clases del Subsistema Seguimiento.

Localización de ficheros de configuración del framework Spring

Los ficheros de configuración de Spring serán situados en la carpeta **WEB-INF** dentro de **WebContent**, en este nivel se organizarán por cada subsistema dentro de la carpeta **conf**, para este caso se muestran los ficheros para el subsistema Seguimiento de Proyectos.

seguimiento-aplicationContex-bussines.xml (Configuración de los beans de la capa de negocio)
seguimiento-aplicationContex-dao.xml (Configuración de los beans de la capa de acceso a datos)
seguimiento-aplicationContex-remoteClient.xml (Configuración de los beans del cliente remoto)
seguimiento-aplicationContex-remoteServer.xml (Configuración de los beans del servidor remoto)
seguimiento-aplicationContex-reportes.xml (Configuración de los beans de la capa de negocio)
seguimiento-aplicationContex-web.xml (Configuración de los beans de la capa de presentación)

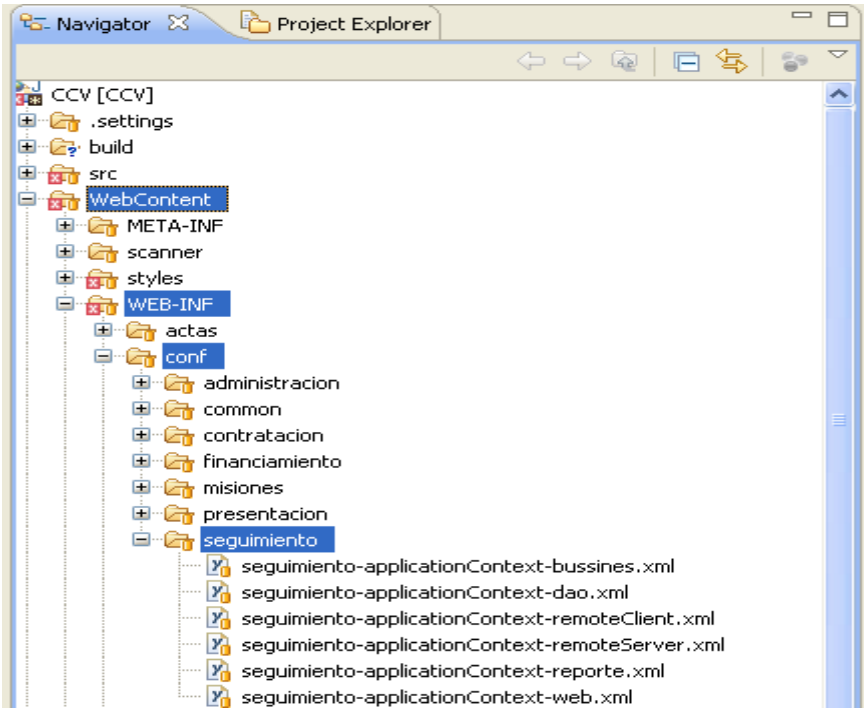

Figura 2.14 Ficheros de Configuración.

2.5.2 ESTRATEGIA DE SEGURIDAD

Encriptación de los datos sensibles que viajen por la red

- Utilizar HTTPs cuando se envíen solitudes con información sensible.
- Cuando se intercambie información de un país a otro se encriptará la misma utilizando algoritmos como md5 a 1024 bits.

Seguridad del canal de conexión

- Establecer en Cuba y Venezuela un canal de conexión seguro entre las máquinas clientes y el sistema.
- De igual forma para la comunicación entre ambos países (ya sea por réplica, invocación remota de métodos, etc.). Para el caso de réplica se empleará una red privada (VPN, Virtual Private Network) a la cual nadie tiene acceso, solamente los usuarios autorizados para entrar a dicha

red, toda la información estará encriptada. Para los métodos remotos se utilizará un canal seguro como SSL (Secure Sockets Layer, Capa de Conexión Segura) que permite el cifrado e integridad de los datos en las comunicaciones entre las dos partes a través de la red informática.

Para garantizar la seguridad del sistema se utilizará un fichero de configuración que se encuentra en el subsistema **Common**, en el cual se configuran los permisos de los usuarios y el acceso a las funcionalidades que puede realizar cada uno de ellos, así como, el acceso a los servicios que brindan los controladores para atender las peticiones de los usuarios.

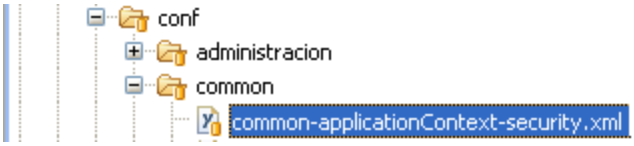

Figura 2.15 Fichero de configuración de seguridad.

EL sistema de permisos de accesos abarca tres niveles:

1. Permisos a nivel de funcionalidades. Ejemplo: adicionarFicha, eliminarFicha, aceptarPropuesta, rechazarModificacionFicha, etc (Lo cubre el Framework Acegi).

2. Acceso a los documentos de acuerdo al nivel que está accediendo (EE (Ente Ejecutor), M (Ministerio) o ST (Secretaría Técnica)). Ejemplo: un ente sólo ve los proyectos en los que está involucrado, al igual que un ministerio (Sólo para los documentos).

3. Permisos de acuerdo al estado del documento. Ejemplo: un proyecto que está rechazado por un ente venezolano, este no lo ve; un proyecto que está modificado esperando aprobación por un ente cubano, este lo ve pero no lo puede eliminar ni modificar (Sólo para los documentos).

Implementación de los niveles de permisos de acceso.

Para implementar los **permisos a nivel de funcionalidades** se configuran utilizando el framework Spring los manejadores de autenticación y de acceso, se definen utilizando Spring Security (Seguridad de Spring) y su sistema de filtros los patrones de URL que representan funcionalidades y sus

respectivos permisos de acceso, y se definen utilizando Spring Security y su sistema de intercepción de métodos (Basado en AOP) los permisos de acceso a métodos de la capa de lógica de negocio.

Acceso a los documentos de acuerdo al nivel que está accediendo

- Si el nivel que está logueado es un ente entonces:
    1. La consulta de listar fichas debe traer sólo los proyectos en los que el ente está involucrado.
    2. La consulta que elimina, modifica o trae los detalles de una ficha debe filtrarse por el id del documento y por los proyectos que estén relacionados con el ente.
- Si el nivel que está logueado es un ministerio entonces:
    1. La consulta de listar fichas debe traer sólo los proyectos en los que el ministerio está involucrado.
    2. La consulta que elimina, modifica o trae los detalles de una ficha debe filtrarse por el id del documento y por los proyectos en que el ministerio esté involucrado.

De esta manera se evita que un ente o un ministerio pueda de alguna manera acceder a un proyecto que no es de su competencia.

Permisos de acuerdo al estado del documento

En este caso para garantizar estas condiciones se creó una tabla de configuraciones (configuración_visibilidad) que abarca el color con que se va a ver un tipo de documento y si se puede eliminar y/o modificar, en un estado determinado, para un tipo de nivel (Ente, Ministerio, ST) y una parte (Cuba, Venezuela) dada.

Una vez realizada una modificación en el estado de un documento se debe disparar un trigger que escriba en otra tabla (documento_visibilidad), la cual va a tener registrado cómo se verá cada documento que exista en el sistema, para un tipo de nivel (EE, M, ST) y una parte (VNZ, CUB) específica. Cuando se vaya a cargar, modificar o eliminar algún documento se debe filtrar en primer

lugar por esta tabla pues en ella estarán los documentos que pueden ser vistos, para un tipo de nivel EE, M, ST) y una parte (VNZ, CUB) determinada, además si se puede eliminar o modificar dicho documento.

2.6 ESTÁNDAR DE CODIFICACIÓN

Un estándar de codificación comprende todos los aspectos de la generación de código. El estándar de codificación es muy importante para los programadores por muchas razones, pues el producto final no es mantenido por ellos toda la vida del software, permite mejorar la lectura del software y que el equipo de trabajo encargado del mantenimiento y soporte del producto pueda entender el código con mayor facilidad. El código fuente debe resultar un entorno familiar para cada miembro del equipo, 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.

Para el desarrollo se siguió el estándar de codificación definido para el lenguaje de programación de Java.

Nombres de ficheros

Los nombres de las clases deben terminar en **Service** para el caso de las interfaces de la capa del negocio, las clases que implementarán dichas interfaces deberán terminar en **ServiceImpl**, en el caso de las clases de la capa de presentación deberán terminar con la palabra **Controller** y las clases de apoyo deberán terminar con la palabra **Util**.

Organización de los ficheros

Los ficheros se agruparán por paquetes, cada paquete se corresponderá con cada una de las capas a implementar. La capa de presentación estará agrupada en la carpeta **web**, en el caso de la capa de lógica de negocio, las interfaces estarán agrupadas en la carpeta **bussines** y las clases que implementan estas interfaces estarán en la subcarpeta **impl**, las clases de apoyo serán ubicadas en la carpeta **util**.

Indentación

Para el salto de líneas de una condición se seguirá la regla de los ocho espacios, en los demás casos se seguirá la regla convencional de los cuatros espacios.

Comentarios

Se utilizarán los comentarios de implementación /*…*/ y //, el uso de estos comentarios son con el objetivo de dar más información acerca del código y evitar realizar comentarios triviales que pueden llevar a un mal entendimiento del mismo, así como realizar dibujos con el uso de los asteriscos.

Los comentarios en bloque deberán realizarse de la siguiente forma:

```
/*
* bloque de código
*/
```

Los comentarios de una línea se realizarán de la siguiente manera:

```
/* Línea de código */
```

Los comentarios de remolque se realizarán siguiendo la siguiente regla:

```
if (condición) {
    return a; /* caso especial */
}
else {
     return b; /* caso general */
    }
```

Declaraciones

Las funcionalidades deben comenzar con letra inicial minúscula, donde la primera palabra debe ser alusiva a la función que se va a realizar y la segunda palabra debe comenzar con mayúscula y debe ser alusiva al tipo de proceso que se va a realizar, ejemplo: eliminarPropuestaPlanOperativo(DFicha_Proyecto ficha_proyecto). No se deberá dejar ningún espacio en blanco entre la declaración de los métodos y los paréntesis, la llave de apertura debe encontrarse al

final de la línea de declaración del método, la llave de cierre comienza en una nueva línea indentada excepto en el caso que no exista sentencias entre ambas, donde deberá aparecer a continuación de la llave de apertura. Solamente se podrá realizar una sola declaración de los atributos por líneas.

Estas son las principales reglas a seguir para lograr un código de mayor calidad y que este pueda resultar de fácil entendimiento para las personas que sustituyan la labor de mantenimiento y soporte del software desplegado.

Para más información dirigirse al **Anexo 1**.

2.7 CAPAS

Para implementar el módulo Seguimiento se utilizaron 4 capas como se abordó anteriormente. A continuación se hará énfasis en las capas de presentación y lógica de negocio.

2.7.1 PRESENTACIÓN

Esta capa es la que interactúa directamente con el usuario para atender las peticiones, manejar los datos y visualizar la información solicitada. Es muy sensible al cambio por lo que se implementó lo más legible posible para posteriores modificaciones que puedan aparecer, y así otros desarrolladores puedan trabajar en ésta sin dificultad.

Para facilitar el desarrollo de esta capa se empleó el módulo Spring MVC, el cual se encarga de separar el modelo, la vista y el controlador, este último es el que recibe las peticiones de los usuarios y le envía los datos que serán mostrados en la vista especificada por el controlador, esto lo hace devolviendo un ModelAndView (Modelo y Vista). Se configuró el fichero seguimiento-aplicationContex-web.xml, en el cual se especifica a través de qué URL se va acceder al controlador y se referencian las propiedades que éste tiene para acceder a la lógica del negocio utilizando el patrón de Inyección de Dependencia.

El MVC utiliza varios tipos de controladores, cada uno especializado para realizar una tarea en particular, aunque algunos son de propósito general. Para poder crear un controlador basta con implementar la interfaz del que se desea emplear y sobrescribir los métodos que sean necesarios para procesar la petición. Esto ayuda a modularizar la aplicación con la combinación de diversos controladores que son enfocados a una tarea específica, teniendo una mejor estructura que ayudará a detectar las fallas y errores que puedan surgir. En el presente caso se utilizaron clases que implementan un Controller (Controlador) y otras que heredan de un MultiActionController (Controlador de muchas acciones).

Un Controller se utiliza generalmente para realizar tareas sencillas sobrescribiendo el método *handleRequest* (en él se manejan las peticiones del usuario), aunque también se puede implementar para atender varias acciones. En la Figura 2.19 se muestra el controlador DetallesComunesController, el cual se implementó para atender varias peticiones, dentro de ellas: obtener información de los ministerios y los entes contrapartes.

*DetallesComunesController.java | seguimiento-applicationContext-web.xml | ActualizarEFController.java

```
public class DetallesComunesController implements Controller
{
    private IniciarProyecto iniciarProyectoService;
    private CronogramService cronogramaService;
    private GestionSolicitudes solicitudesService;
    public void setSolicitudesService(GestionSolicitudes solicitudesService) {
        this.solicitudesService = solicitudesService;
    }
    public void setCronogramaService(CronogramService cronogramaService) {
    public void setNomencladorService(NomencladorService nomencladorService) {
    public void setIniciarProyectoService(IniciarProyecto iniciarProyectoService) {
    @Override
    public ModelAndView handleRequest(HttpServletRequest request,HttpServletResponse response)
    {
        int action = Integer.parseInt(request.getParameter("accion"));//accion a ejecutar
        switch (action)
        {
        case 0://Ministerios contrapartes
        {
          getMinisteriosContraparte(request, response);
        }break;
        case 1://entes contrapartes
        {
          getEntesContraparte(request, response);
        }break;
```

Figura 2.19: Implementación de un Controller.

El MultiActionController permite la implementación de varios métodos dentro de un sólo controlador, a través del cual se podrán mapear las diferentes peticiones a cada uno de los métodos correspondientes. En la Figura 2.20 se muestra el controlador ReplanificarCronogramaController, en el cual se implementaron varios métodos, algunos de ellos son los que permiten agregar y modificar el monto del cronograma de un proyecto.

*seguimiento-applicationContext-web.xml | ActualizarEFController.java | *ReplanificarCronogramaController.java

```
package cu.uci.ccv.seguimiento.web;

import java.math.BigDecimal;

public class ReplanificarCronogramaController extends MultiActionController
{
    private CronogramService cronogramaService;
    private GestionCronograma gestionCronogramaService;
    private BusquedaService busquedaService;
    public void setGestionCronogramaService(
            GestionCronograma gestionCronogramaService) {
        this.gestionCronogramaService = gestionCronogramaService;
    }
    public void setCronogramaService(CronogramService cronogramaService) {

    public void agregarMonto(HttpServletRequest request, HttpServletResponse response)throws Exception
    //
    public void modificarMonto(HttpServletRequest request, HttpServletResponse response)throws Exception
    {
        SecurityContext sc = (SecurityContext) request.getSession().getAttribute("ACEGI_SECURITY_CONTEXT");
        DUsuario usuario =  (DUsuario) sc.getAuthentication().getPrincipal();
        String datos = (String)request.getParameter("datos");
```

Figura 2.20: Implementación de un MultiActionController.

Descripción de las clases.

Para darle cumplimiento a los requisitos funcionales asociados al módulo Seguimiento y garantizar que el sistema funcione correctamente, fue necesario implementar 13 clases para controlar la interacción del usuario con la aplicación, 2 clases útiles para obtener datos específicos, realizar cálculos y validaciones necesarias para determinadas modificaciones, y 12 clases para gestionar la lógica de

negocio, cada una de éstas últimas implementan una interfaz donde se exponen las operaciones que necesitan los controladores para atender las diferentes peticiones de los usuarios. La descripción de cada una de estas clases con sus funcionalidades se encuentra en el **Anexo 2**.

2.7.2 LÓGICA DEL NEGOCIO

La capa de lógica de negocio tiene una gran importancia pues es la encargada de realizar toda la lógica del sistema a través de sus clases y operaciones, por tanto se debe tener un entendimiento claro del funcionamiento del sistema y del diseño de clases realizado por el diseñador, éstas deben cumplir con las particularidades de la plataforma de trabajo y con la arquitectura previamente definida por el arquitecto, se debe desglosar el sistema en funcionalidades las cuales corresponden a los requisitos funcionales que darán satisfacción al cliente, teniendo en cuenta además los patrones de diseño y un buen empleo del estándar de codificación definido para el lenguaje de programación Java.

La implementación de las clases de negocio se realizó teniendo en cuenta cada uno de los procesos del módulo Seguimiento del proyecto ICICV, por cada proceso se creó una clase interfaz, donde solamente se declaran los métodos, y una clase donde se implementa cada método declarado en la clase interfaz, también se creó una clase útil, donde se implementaron todas las operaciones y cálculos necesarios para el apoyo de la implementación de todas las funcionalidades.

Todos los métodos utilizados serán declarados en español, donde su primera palabra será en minúscula y se hará énfasis en el tipo de operación a realizar, ejemplo: *enviar, eliminar, proponer, terminar.*

Teniendo en cuenta que la aplicación manejará los datos de las operaciones del Convenio, no se puede dar margen al error, por lo que se realizará un minucioso tratamiento de errores sobre todo en la capa de lógica de negocio, que es la encargada de confirmar que todas los operaciones se realicen de forma correcta y no haya pérdida de dinero u otro tipo de recursos mediante cualquier transacción, por lo que cada dato antes de ser enviado a la capa de acceso a datos debe ser verificado.

Para asegurar la atomicidad de la aplicación se configuraron transacciones de chequeo a todos los métodos de la capa de negocio. Para ello se configuró un txAdvice, donde se definieron los métodos que serán interceptados y el aop:config, en el cual se configuraron las transacciones por paquetes haciendo una referencia a los métodos definidos en el txAdvice, con el objetivo de optimizar la aplicación.

administracion-applicationContext-bussines.xml | seguimiento-applicationContext-remoteClient.xml | seguimiento-

```
<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="modificarNoticia" propagation="NOT_SUPPORTED" />
        <tx:method name="eliminarNoticia" propagation="NOT_SUPPORTED" />
        <tx:method name="modificarNoticiaDao" propagation="REQUIRED" />
        <tx:method name="eliminarNoticiaDao" propagation="REQUIRED" />
        <tx:method name="eliminar*" propagation="REQUIRED" />
        <tx:method name="aceptar*" propagation="REQUIRED" />
        <tx:method name="rechazar*" propagation="NOT_SUPPORTED" />
        <tx:method name="presentar*" propagation="REQUIRED" />
        <tx:method name="asignar*" propagation="REQUIRED" />
        <tx:method name="cerrar*" propagation="REQUIRED" />
        <tx:method name="enviar*" propagation="REQUIRED" />
```

Figura 2.21: Configuración de las transacciones.

Las transacciones son unidades atómicas de ejecución, es decir, son un grupo de instrucciones que se ejecutan con éxito en su totalidad, o no se ejecuta ninguna. Cuando una transacción se inicia puede terminar de dos formas: con un commit (aceptada) o un rollback (rechazada). Un commit confirma toda la ejecución dentro de la transacción y se guardan todos los datos. Un rollback "vuelve atrás" todos los cambios que puedan haberse realizado y todos los datos quedan de la forma en que estaban antes de comenzar la operación, de esta forma se evita que se guarde alguna información errónea dentro de la base de datos.

Cada vez que se realice una operación por cualquier organismo (Ente, Ministerio, Secretaría Técnica) de ambos países, las partes contrarias deben ser notificadas para que puedan dar respuestas a las

solicitudes realizadas por su contraparte, para el uso de las notificaciones se crearon clases interceptoras con el uso de la programación orientada a aspectos (AOP), que también permiten crear las trazas de los proyectos y de las operaciones realizadas por los usuarios.

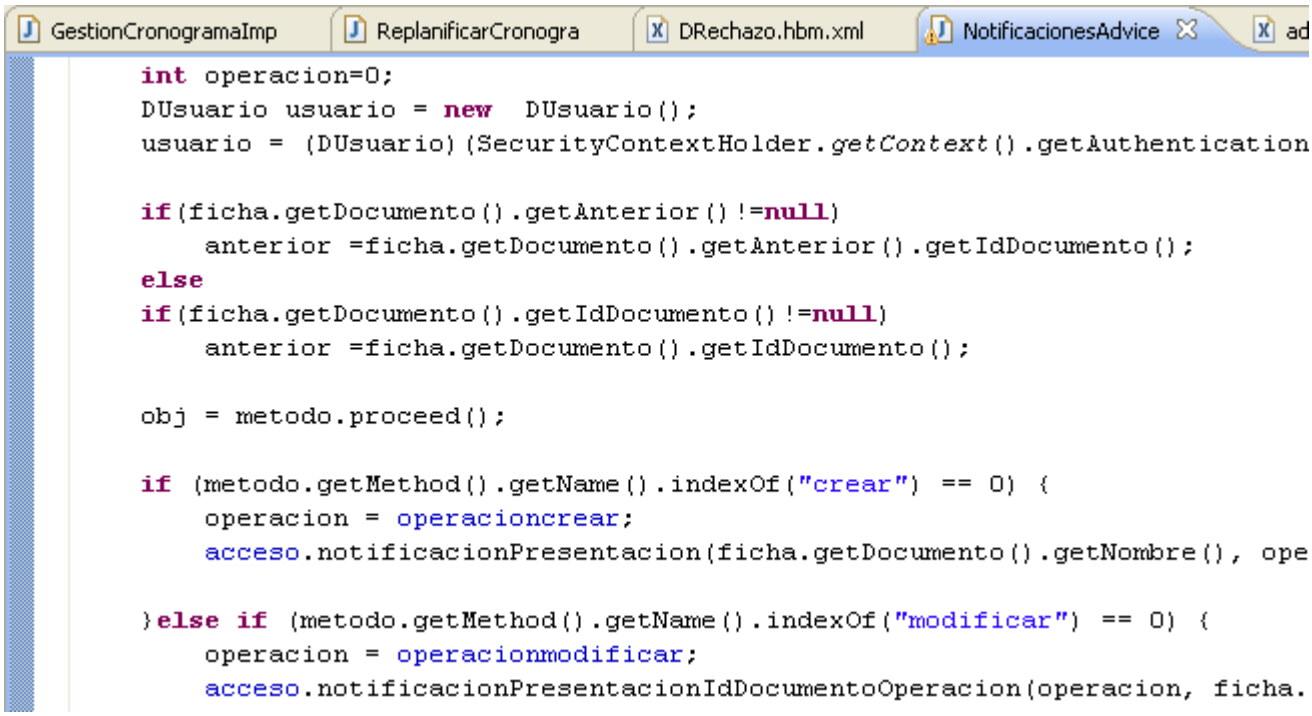

Figura 2.22: Notificación de las fichas de proyectos.

En la figura anterior se muestra una clase interceptora que permite crear todas las notificaciones a partir de las operaciones realizadas por los usuarios sobre las fichas de proyectos. Para esto, cada vez que se invoca una operación del negocio sobre una ficha, la clase NotificacionesAdvice la captura, espera que se ejecute el método y envía una notificación al usuario contraparte en caso de que no ocurra ningún error mientras se ejecutaba la operación.

2.8 INTEGRACIÓN

Una vez implementadas las clases necesarias de las capas de presentación y lógica de negocio se necesitan configurar los archivos de Spring para que estas puedan comunicarse y brindar los servicios para los cuales fueron implementadas. A continuación se muestra cómo interactúan los ficheros de configuración con las mismas.

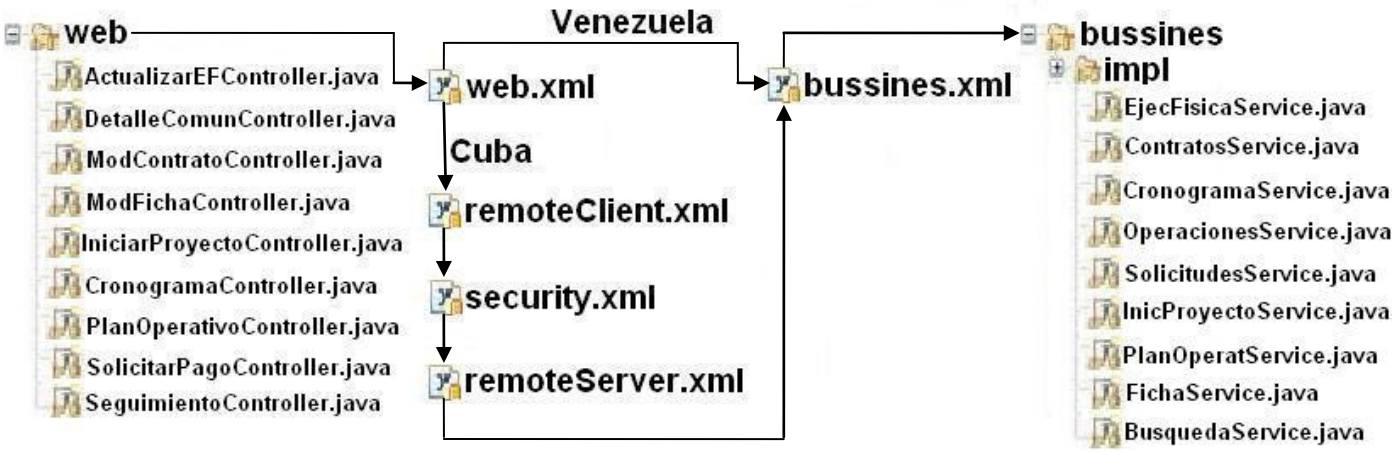

Figura 2.23: Integración de las capas de presentación y lógica de negocio.

Debido a que no todos los organismos cubanos tienen acceso a Internet, se publicarán dos aplicaciones, una en Cuba y otra en Venezuela, y como en esta última región se publicarán los métodos del negocio para brindar los servicios a los clientes de ambas naciones será necesario realizar dos configuraciones para lograr integrar ambas capas. Para lograr dicha integración Spring utiliza el patrón Inyección de Dependencia, el cual permite identificar los servicios que se brindan y localizarlos mediante los archivos XML configurados. Para esto se definieron los beans necesarios en el fichero **web.xml** inyectando las dependencias de cada controlador para acceder a los servicios del negocio. Cuando los clientes venezolanos realicen una petición, el controlador requerido se conecta directamente con los métodos expuestos en la interfaz del negocio inyectando las dependencias de los beans creados en el fichero **bussines.xml.** Cuando la petición sea realizada por los clientes cubanos, el controlador encargado de ejecutar la petición se conecta a través de un cliente remoto inyectando las dependencias de los beans configurados en el fichero **remoteClient.xml**, que permiten acceder a

los métodos publicados en el servidor remoto de Venezuela a los cuales tenga permiso el usuario que solicite el servicio.

Para comprender mejor lo expuesto anteriormente, se muestra a continuación una parte de los ficheros en los cuales se realizan las configuraciones necesarias para garantizar la integración de ambas capas.

*seguimiento-applicationContext-bussines.xml *seguimiento-applicationContext-web.xml

```
31    <bean id="IniciarProyecto" class="cu.uci.ccv.seguimiento.bussines.impl.IniciarProyectoImpl"
32          parent="ProcesoService">
33       <property name="documentoService" ref="DocumentoService"/>
34       <property name="facturaDAO" ref="facturaDao"/>
35       <property name="fichaService" ref="FichaService"/>
36       <property name="gestionOperaciones" ref="GestionOperaciones"/>
37    </bean>
38
39    <bean id="GestionCronograma" class="cu.uci.ccv.seguimiento.bussines.impl.GestionCronogramaImpl"
40          parent="CronogramEFService">
41       <property name="procesoService" ref="ProcesoService"/>
42       <property name="documentoService" ref="DocumentoService"/>
43       <property name="planOperativoDAO" ref="planOperativoDao"/>
44       <property name="facturaDAO" ref="facturaDao"/>
45       <property name="cronogramaDao" ref="cronogramaDao"/>
46       <property name="fichaService" ref="FichaService"/>
47    </bean>
```

Figura 2.24: Fichero de configuración de las clases del negocio.

En la imagen anterior se observa cómo se crearon los beans con un identificador que permite el acceso a la clase a la cual hace referencia, inyectando todas las dependencias que ésta posee con otras clases del negocio y las de acceso a datos, las cuales fueron configuradas en el fichero seguimiento-aplicationContext-dao.xml. En el ejemplo se muestra que para utilizar las funcionalidades de las clases “IniciarProyectoImpl” y “GestionCronogramaImpl” es necesario referenciar a los beans “IniciarProyecto” y “GestionCronograma” respectivamente.

seguimiento-applicationContext-remoteClient.xml | *seguimiento-applicationContext-remoteServer.xml

```
28 <bean name="/IniciarProyecto.remote"
29   class="org.springframework.remoting.httpinvoker.HttpInvokerServiceExporter">
30   <property name="service"> <ref bean="IniciarProyecto" /> </property>
31   <property name="serviceInterface" value="cu.uci.ccv.seguimiento.bussines.IniciarProyecto"/>
32 </bean>
33
34 <bean name="/GestionCronograma.remote"
35   class="org.springframework.remoting.httpinvoker.HttpInvokerServiceExporter">
36   <property name="service"> <ref bean="GestionCronograma" /> </property>
37   <property name="serviceInterface" value="cu.uci.ccv.seguimiento.bussines.GestionCronograma"/>
38 </bean>
```

Figura 2.25: Fichero de configuración del servidor remoto.

En este fichero es donde se configuran los beans para exponer los métodos en el servidor remoto de Venezuela utilizando el protocolo Http Invoker, a los cuales se accederán a través de la dirección URL definida para cada bean, haciendo referencia al bean que permite acceder a la clase del negocio que implementa la interfaz que brinda el servicio del negocio. Por ejemplo, si se hace referencia al bean "/IniciarProyecto.remote", se pueden utilizar los servicios brindados por la clase "IniciarProyectoImpl" con la inyección de la dependencia del bean "IniciarProyecto" que permite utilizar los métodos implementados en la clase.

*seguimiento-applicationContext-remoteClient.xml | *seguimiento-applicationContext-remoteServer.xml

```
28 <bean id="IniciarProyecto"
29   class="org.springframework.remoting.httpinvoker.HttpInvokerProxyFactoryBean">
30   <property name="serviceUrl" value="${url}/IniciarProyecto.remote" />
31   <property name="serviceInterface" value="cu.uci.ccv.seguimiento.bussines.IniciarProyecto"/>
32   <property name="httpInvokerRequestExecutor" ref="requestExecutor" />
33 </bean>
34
35 <bean id="GestionCronograma"
36   class="org.springframework.remoting.httpinvoker.HttpInvokerProxyFactoryBean">
37   <property name="serviceUrl" value="${url}/GestionCronograma.remote" />
38   <property name="serviceInterface" value="cu.uci.ccv.seguimiento.bussines.GestionCronograma"/>
39   <property name="httpInvokerRequestExecutor" ref="requestExecutor" />
40 </bean>
```

Figura 2.26: Fichero de configuración del cliente remoto.

En este fichero es en el que se configuran los beans del cliente remoto de Cuba, que permiten consumir los recursos expuestos en el servidor remoto de Venezuela a través de la dirección donde se

encuentran publicados. Por ejemplo, si el cliente es cubano y el controlador que atenderá la petición hace referencia al bean “IniciarProyecto”, éste se comunica a través de la dirección “${url}/IniciarProyecto.remote” que permitirá utilizar la funcionalidad requerida para atender la petición del usuario.

seguimiento-applicationContext-bussines.xml | *seguimiento-applicationContext-web.xml

```
<bean name="/DetallesComunes.htm" class="cu.uci.ccv.seguimiento.web.DetallesComunesController">
    <property name="nomencladorService" ref="NomencladorSevice" ></property>
    <property name="iniciarProyectoService" ref="IniciarProyecto"></property>
    <property name="cronogramaService" ref="CronogramEFService"></property>
    <property name="solicitudesService" ref="GestionSolicitudes"></property>
</bean>

<bean id="cronogramaController"
    class="cu.uci.ccv.seguimiento.web.ReplanificarCronogramaController">
    <property name="cronogramaService"> <ref bean="CronogramEFService" /></property>
    <property name="gestionCronogramaService"> <ref bean="GestionCronograma" /></property>
    <property name="busquedaService" ref="BusquedaService"></property>
    <property name="methodNameResolver">
        <bean
            class="org.springframework.web.servlet.mvc.multiaction.PropertiesMethodNameResolver">
            <property name="mappings">
              <props>
                 <prop key="/replanificarCronogramaAgregar.htm">agregarMonto</prop>
                 <prop key="/replanificarCronogramaModificar.htm">modificarMonto</prop>
                 <prop key="/replanificarCronogramaEliminar.htm">eliminarMonto</prop>
                 <prop key="/replanificarCronogramaEnviar.htm">enviarPropuestaCronograma</prop>
              </props>
            </property>
        </bean>
    </property>
</bean>
```

Figura 2.27: Fichero de configuración de las clases controladoras.

Para configurar el acceso a los controladores, se crearon los beans definiendo la dirección URL por la cual se va a permitir el acceso a los servicios que brinda cada uno, referenciando a los beans que resolverán las llamadas de cada controlador ya sean del negocio o del cliente remoto según sea el cliente que solicite el servicio. El primer bean definido se utiliza para acceder a un Controller (DetallesComunesController) a través de la dirección “/DetallesComunes.htm”. El segundo se utiliza para acceder a un MultiActionController (ReplanificarCronogramaController) utilizando una dirección URL diferente para acceder a cada uno de los métodos implementados, por ejemplo: para utilizar el método “agregarMonto” se utiliza la URL “replanificarCronogramaAgregar.htm”.

*common-applicationContext-security.xml seguimiento-applicationContext-remoteServer.xml

```
128 <bean id="filterSecurityInterceptor"
129   class="org.acegisecurity.intercept.web.FilterSecurityInterceptor">
130   <property name="authenticationManager" ref="authenticationManager" />
131   <property name="accessDecisionManager" ref="accessDecisionManager" />
132   <property name="objectDefinitionSource">
133     <value>
134       CONVERT_URL_TO_LOWERCASE_BEFORE_COMPARISON
135       PATTERN_TYPE_APACHE_ANT
136       <!-- controladores -->
137       /detallescomunes.htm/**=Func_Iniciar_Proyecto,Func_Planificar_Cronograma_Ejecuicon_Financiera
138       /replanificarcronograma*/**=Func_Planificar_Cronograma_Ejecuicon_Financiera
139       <!-- metodos remotos -->
140       /iniciarproyecto.remote**=Func_Iniciar_Proyecto,Func_Revisar_Reformulacion_Proyecto
141       /gestioncronograma.remote**=Func_Planificar_Cronograma_Ejecuicon_Financiera
```

Figura 2.28: Fichero de configuración de seguridad.

EL fichero de seguridad es uno de los más importantes, en él se configuran los accesos de los usuarios a cada controlador y a los métodos remotos implementados. Los usuarios que tengan acceso a alguna de las funcionalidades definidas para acceder a cada URL, tendrán acceso a los servicios que brinda cada una de las clases que controlan las peticiones solicitadas. Por ejemplo, el usuario que tenga permisos para utilizar la funcionalidad “Func_Iniciar_Proyecto” o “Func_Replanificar_Cronograma_Ejecucion_Financiera” podrá consumir los servicios brindados por el controlador configurado a través de la dirección “detallescomunes.htm”.

2.9 CONCLUSIONES

En este capítulo se presentó una propuesta de implementación e integración de las capas de presentación y lógica de negocio del módulo Seguimiento, donde se analizaron los casos de usos del sistema por cada uno de los procesos funcionales identificados, se estructuraron cada una de las capas por paquetes modelando los componentes de las clases que necesitaban ser implementadas en las capas de presentación y lógica de negocio, se trazó la estrategia de seguridad de la aplicación, se definió el estándar de codificación a seguir para la implementación, se abordaron las estrategias utilizadas en cada una de las capas mencionadas describiendo las clases que fueron implementadas y se describió el método empleado para integrar ambas capas.

CAPÍTULO 3: ANÁLISIS DE LOS RESULTADOS

3.1 INTRODUCCIÓN

Durante todo el proceso de implementación se fue realizando a la par la gestión de la calidad del producto mediante el uso de herramientas y técnicas de pruebas de software. Como parte de las técnicas se abordarán las pruebas de caja blanca que fueron realizadas a la capa de presentación, a la capa de lógica del negocio y a la integración de ambas capas, y las pruebas de caja negra que se llevaron a cabo sobre la interfaz del software. Se realizará un análisis de los resultados obtenidos durante el proceso de prueba, para conocer si se cumplió con el objetivo propuesto.

3.2 PRUEBAS DE CAJA BLANCA

Para las pruebas de caja blanca se utilizó el framework JUnit, el cual brinda un conjunto de librerías que se integran fácilmente al IDE de desarrollo seleccionado: Eclipse. JUnit permite la realización de las pruebas a los métodos de las clases implementadas en ambas capas. Para hacer uso de este framework se definieron los siguientes pasos:

- Crear un caso de prueba por cada clase implementada.
- Configurar en el caso de prueba, los ficheros que permiten comunicar las clases de las capas de presentación y lógica de negocio.
- Definir los métodos a probar dentro de cada caso de prueba, incluyendo los parámetros de entrada.
- Realizar pruebas a los métodos.

A continuación se muestran las imágenes que representan la realización de las pruebas de caja blanca, que se les aplicaron a las funcionalidades implementadas en las clases de las capas de presentación y lógica de negocio, para darle solución al caso de uso Re-planificar Plan operativo.

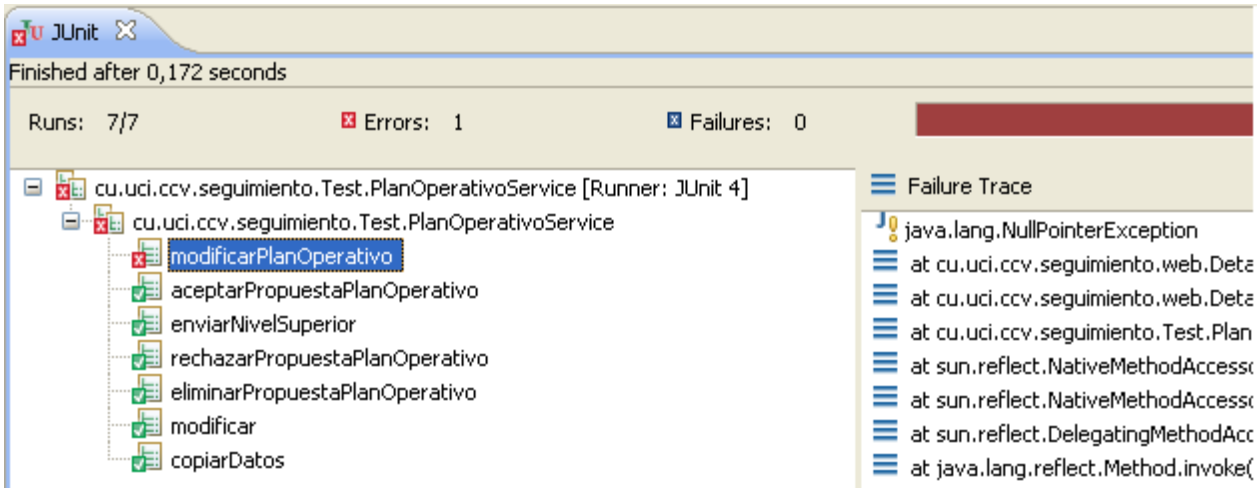

Figura 3.1: Prueba realizada a la clase PlanOperativoService.

En la figura anterior se muestra el resultado de una de las pruebas de caja blanca, en la cual se observa la existencia de un error, en este caso es la falta del tratamiento de errores cuando se trató de insertar un objeto nulo al modificar el plan operativo. Inmediatamente se procedió a realizar su corrección y se realizó nuevamente la prueba de caja blanca al método defectuoso.

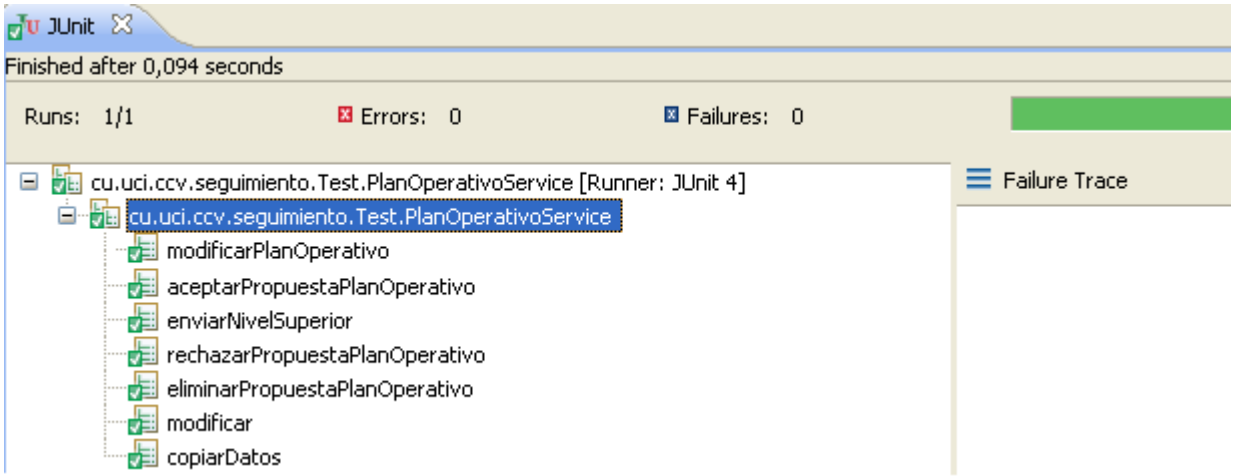

Figura 3.2: Resultado de la prueba con el método ya corregido

Ya en la segunda iteración sólo se le aplicó la prueba al método que arrojó el error y su resultado fue satisfactorio.

Una vez que se probaron los métodos de las clases de la capa de negocio se procedió a realizar pruebas de integración a los métodos de las clases de la capa de presentación, las cuales se muestran a continuación:

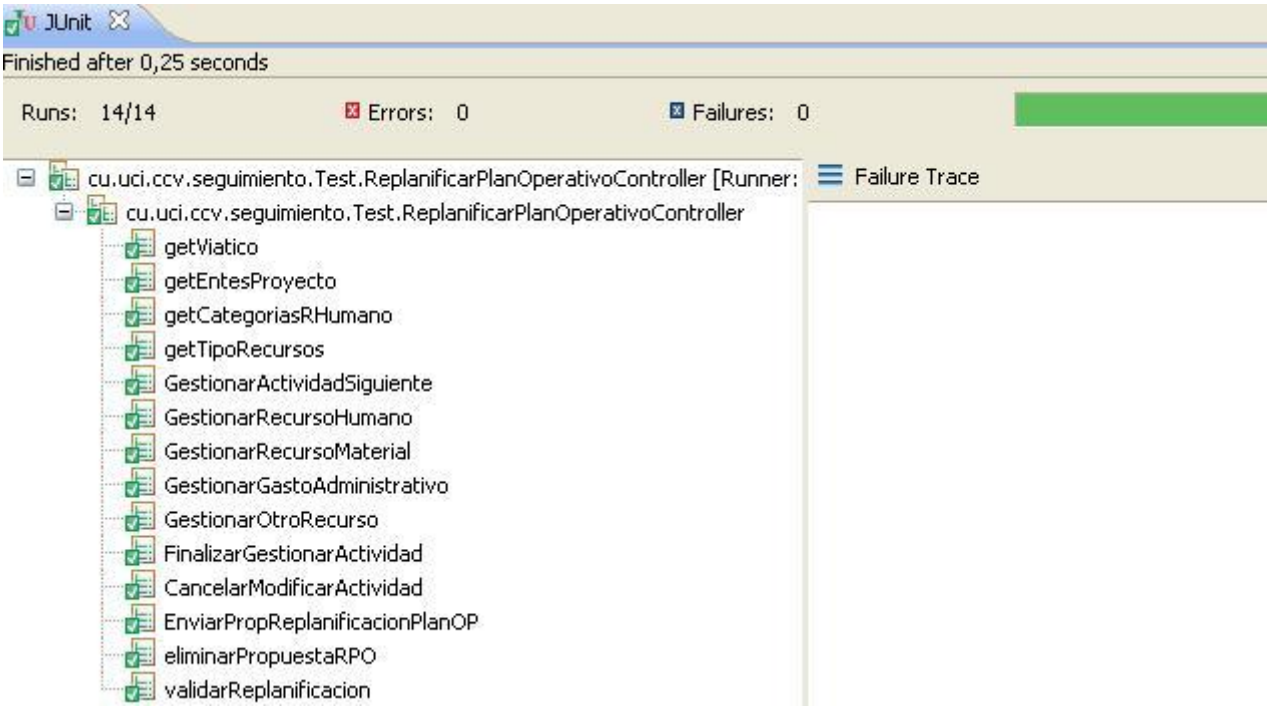

Figura 3.3: Prueba realizada a la clase ReplanificarPlanOperativoController

La figura anterior muestra la prueba realizada a la clase que contiene los métodos necesarios para replanificar el plan operativo de un proyecto cuyo resultado fue satisfactorio.

Utilizando la estrategia definida para usar el framework JUnit, se probaron todas las funcionalidades de cada una de las clases implementadas en cada capa. Para más información de estas pruebas dirigirse al **Anexo 3**.

3.3 PRUEBAS DE CAJA NEGRA

Para garantizar el correcto funcionamiento del software, el grupo de calidad del proyecto aplicó una serie de pruebas de caja negra. Para esto, se diseñaron casos de pruebas por cada caso de uso, describiendo las condiciones de ejecución y el flujo central de los eventos. Después de diseñar los diferentes casos de pruebas, se realizaron las pruebas correspondientes para cada escenario de los casos de usos. Para obtener información de algunas de las pruebas realizadas a los casos de usos más importantes ver el **Anexo 4**.

En la siguiente tabla se muestran algunos de los errores encontrados durante la realización de las pruebas de caja negra en varias de las funcionalidades de la aplicación y el tiempo de respuesta por parte del equipo de desarrollo.

Funcionalidad	Error	Tiempo de Solución Aproximado
Re-planificar cronograma de ejecución financiera.	Error al rechazar envío de la re-planificación del cronograma de ejecución financiera.	1 hora
Re-planificar plan operativo	Error al proponer re-planificación del plan operativo.	3 horas
Re-planificar cronograma de ejecución financiera.	Error al aceptar la re-planificación del cronograma de ejecución financiera.	1 hora
Iniciar proyecto	Error al proponer el inicio de un proyecto.	30 minutos
Revisar solicitud de pago	Error al modificar una factura de una solicitud de pago	2 horas

Tabla 3.1: Errores detectados durante la primera iteración de pruebas.

Durante la primera iteración de pruebas realizadas por el equipo de calidad se detectaron una serie de no conformidades (algunas de ellas se muestran en la tabla 3.1) que luego fueron entregadas al equipo de desarrollo. Se utilizó el framework JUnit que brinda facilidades para tracear todo el código de

la aplicación para localizar las inconsistencias del sistema. Una vez detectados los errores se corrigieron en el menor período de tiempo y posteriormente se volvió a realizar pruebas de caja blanca y caja negra.

A continuación se muestra una imagen que representa un error encontrado al probar el caso de uso aceptar re-planificación del cronograma de ejecución financiera:

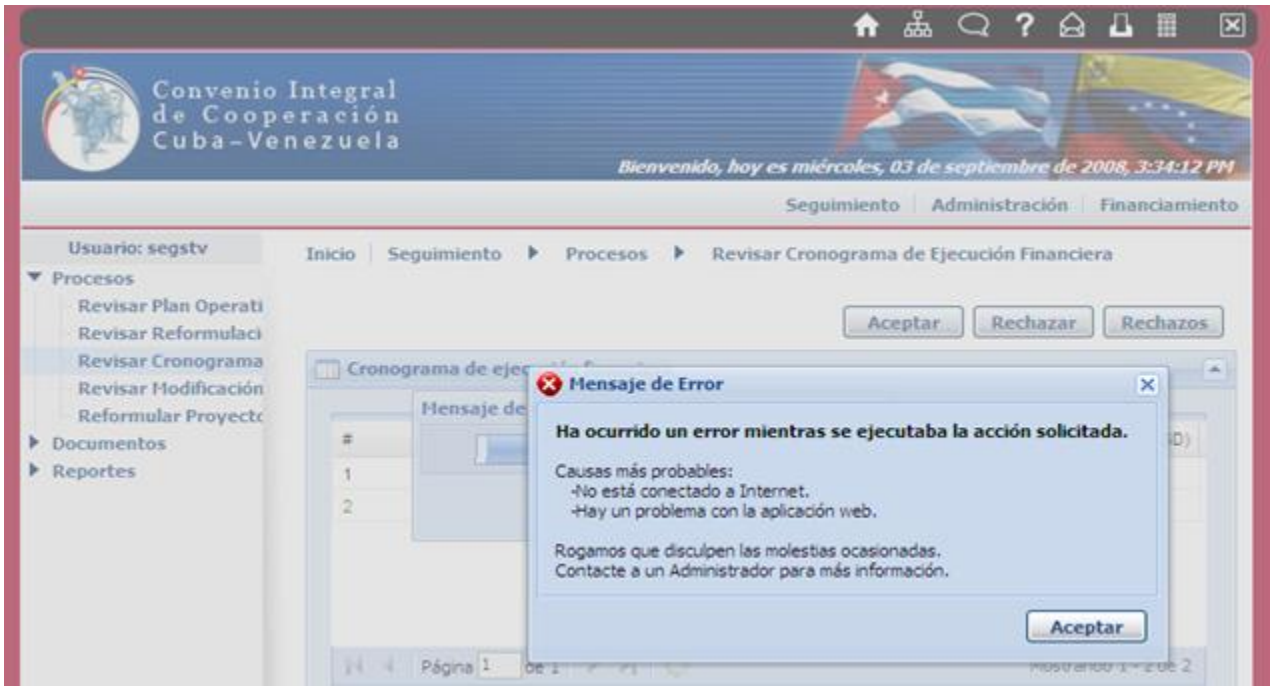

Figura 3.8: Error arrojado durante una prueba de caja negra.

En éste caso no era un error de codificación, sino de configuración del flujo de las fichas dentro de la base de datos, el cual fue corregido satisfactoriamente.

3.4 RESULTADOS

Después de la revisión del software por un grupo de especialistas del grupo de calidad UCI, se realizó un análisis de los resultados obtenidos de las pruebas de caja negra teniendo en cuenta los principales aspectos de medición, los cuales se describen a continuación:

Funcionalidad – La aplicación mostró ser completamente funcional, dando solución a todos los requisitos solicitados por los clientes, los cuales fueron identificados y validados en la etapa de Levantamiento de Requisitos.

Fiabilidad – Todos los problemas de tratamiento de errores que fueron descubiertos durante el desarrollo fueron solucionados, por lo que la aplicación es capaz de detectar cualquier tipo de error, recuperarse y enviar a los interesados el tipo de error ocurrido, manteniendo así todas sus funcionalidades.

Eficiencia – Debido a la estrategia de integración utilizada entre las capas de presentación y lógica de negocio, la aplicación es capaz de darle respuestas a todas las solicitudes del usuario en un tiempo relativamente corto, aunque en este tiempo de respuesta influyen de manera directa las condiciones de conexión que posean los clientes. También es importante señalar que influyen en el tiempo de respuesta las medidas tomadas en capas como la de interfaz de usuario y acceso a datos.

Usabilidad – Teniendo en cuenta que el software fue realizado para un tipo de usuario determinado, el cual posee conocimientos avanzados de los procesos que se han informatizado, la aplicación mostró que es muy fácil la navegación por sus páginas, aún para personas que poseen pocos conocimientos de los procesos. Además de poseer una interfaz de usuario que es muy agradable para la persona que la utiliza.

Mantenibilidad – Debido a que el software posee una arquitectura por capas su mantenibilidad es poco costosa. Como se cumplieron con todos los estándares de codificación, es fácil para cualquier persona que posea conocimientos del lenguaje de programación realizar los cambios solicitados, con vista a mejorar o adicionar nuevas funcionalidades.

Portabilidad – El software puede ser utilizado con varios navegadores web, como el Internet Explorer y el Mozilla Firefox, además puede ser utilizado en cualquier sistema operativo.

En el **Anexo 5** se puede observar el acta de finiquito (Fin) firmada por los representantes del proyecto por Cuba y por Venezuela, la cual evidencia que el software fue elaborado satisfactoriamente y cumple con las expectativas del cliente.

3.5 CONCLUSIONES

En este capítulo se abordaron algunas de las diferentes pruebas que se le realizaron al software. Durante todo el proceso de desarrollo se le aplicaron continuas pruebas de caja blanca, con el que se comprobó el correcto funcionamiento del sistema, así como de la integración de las capas de presentación y lógica de negocio. También se mostraron los resultados arrojados en la aplicación de las pruebas de caja negra por un grupo de especialistas al sistema desarrollado, y se comprobó que cumplía con todas las normas, parámetros y estándares establecidos por CALISOFT. Por todo lo antes expuesto se puede decir que el software posee una calidad aceptable y finalmente se le dio cumplimiento al objetivo propuesto: ¿Cómo contribuir al cumplimiento de los requisitos funcionales y no funcionales asociados al módulo Seguimiento del proyecto ICICV implementando e integrando las capas de presentación y lógica de negocio?

CONCLUSIONES

Al finalizar el presente trabajo se arribó a las siguientes conclusiones:

- Se valoraron las tecnologías a utilizar para el desarrollo de la aplicación.
- El framework Spring facilitó la implementación e integración de las capas de presentación y lógica de negocio garantizando una seguridad robusta para la aplicación.
- El estándar de codificación utilizado permitió implementar las capas con un código legible y fácil de entender por cualquier programador que conozca el lenguaje de programación Java.
- Se crearon todas las clases y operaciones necesarias para implementar las capas de presentación y lógica de negocio, configurando los archivos que permiten su integración para brindar los servicios que solicitan los usuarios.
- Las pruebas realizadas al software demuestran que el módulo Seguimiento cumple satisfactoriamente con los requisitos que garantizan su funcionamiento.

Con el desarrollo de este trabajo se le dio cumplimiento al objetivo general propuesto, implementándose una aplicación web que facilita gestionar el control de las actividades que comprenden los proyectos firmados en las Comisiones Mixtas de Cooperación entre Cuba y Venezuela.

RECOMENDACIONES

- Realizar una versión para que sea utilizada para el control de los proyectos entre los Organismos del estado.
- Realizar una versión que sea configurable para controlar los proyectos entre dos determinados países.
- Extender la vasta experiencia en el desarrollo de aplicaciones de gestión de proyectos de los miembros del equipo de trabajo de ICICV a los demás proyectos productivos de la facultad.
- Utilizar el framework Spring para facilitar la implementación de las capas de los futuros proyectos productivos.

BIBLIOGRAFÍA

1. **Cornejo, José Enrique González. 2001.** DocIris. [En línea] 25 de marzo de 2001. http://www.docirs.cl/arquitectura_tres_capas.htm.

2. **Mariñán, Martín Pérez. 2005.** [En línea] 6 de junio de 2005. http://linux.iingen.unam.mx/pub/Documentacion/CongresojavaHispano2003/JavaH/Modelos_de_desarrollo_de_aplicaciones_distribuidas_con_J2EE.pdf.

3. **Programación en capas. 2009.** Slideshare. [En línea] 2009. http://www.slideshare.net/Decimo/arquitectura-3-capas.

4. **Abián, Miguel Ángel. 2004. devjoker.** [En línea] 2004. http://www.devjoker.com/html/J2EE-Y-NET--LA-RIVALIDAD-PERMANENTE.html.

5. **Ortega, Aguila. 2008.** Análisis, diseño e implementación de la capa de lógica de negocio del módulo Análisis de Información del SIIPOL. Ciudad de la Habana : Universidad de las Ciencias Informáticas. s.n., 2008.

6**. Mario Alfredo Sánchez Rico. 2006**. UDLAP. Universidad de las Américas Puebla. [En línea] 2006. http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/sanchez_r_ma/indice.html.

7. **C. Walls, R. Breidenbach.** Spring in Action. [PDF] s.l.: Manning Publications, 2005.

8. **Fernández, Rodríguez. 2007**. Propuesta de Modelo Arquitectónico para Aplicaciones Empresariales sobre la Plataforma JEE. Ciudad Habana : Universidad de las Ciencias Informáticas. s.n., mayo de 2007.

9. **Jacobson, I., Booch, G., Rumbaugh, J**. El Proceso Unificado de Desarrollo del Sofware.

10**. Gattaca.SA.** (s.f.). *http://www.e-Gattaca.com/MSF.pdf.*

11. ***Programación Extrema*.2007.** [En línea] 14 de septiembre de 2007. http://www.programacionextrema.org/

12. **Nuñes, J. L.** [En línea] septiembre de 2008. Plataforma para la docencia virtual online. http://www.scribd.com/doc/6108306/PLATAFORMA-PARA-LA-DOCENCIA-VIRTUAL-ONLINE

13. **Merritt, E.** (s.f.). [En línea] Desarrollo Seminario Tecnologías SENA. http://desarrolloseminariotecnologias.wordpress.com/netbeans/

14. **Microsoft Corporation**. 2009. [En línea] 2009. *http://www.microsoft.com/spanish/MSDN/estudiantes/ingsoft/planificacion/msf.mspx.*

15. **Pacheco Iglesias, A. E., & Casanova Mutis, A. ..** junio del 2008. Diseño e implementación de funcionalidades que se llevan a cabo en los registros mercantiles: solicitudes de expedientes, copias de documentos y sellado de libros. Ciudad Habana.

16. **Hommel, S.** 20 de abril de 1999. Convenciones de Código para el lenguaje de programación Java TM.

17. **Usaola, M. P**. 2009. [En línea] 19 de febrero de 2009 *http://www.inf-cr.uclm.es*.

18. **Vicente, R.** 2009. [En línea] 20 de marzo de 2009 *http://www.programania.net/raul-vicente/*.

19. **Iglesias, N. F.** 2006. [En línea] noviembre de 2006. http://www.di.uniovi.es/~cueva/asignaturas/doctorado/2006/trabajos/SW_ligeros.pdf

20. **Taringa.2008**.[En línea] 15 de octubre de 2008. http:// www.taringa.net/posts/info/1649223/Que-IDE-usas---JAVA.html

21. **Juristo, N., Moreno, A. M., & Vegas, S. 2006.** TÉCNICAS DE EVALUACIÓN DE SOFTWARE.

22. **Babylon.com LTD**.[En línea] junio de 2007. babylon. http://www.babylon.com/definition/Groupware/Spanish

23. **Weblogs SL.** [En línea]25 de enero de 2008. GENBETA. http://www.genbeta.com/web/collabtive-sistema-de-gestion-de-proyectos-para-instalar-en-nuestro-propio-servidor-web

24. **IDG COMMUNICATIONS, S. A. U.** [En linea] 27 de abril de 2009. @World. http://www.idg.es/iworld/articulo.asp?id=148846

ANEXOS

Anexo 1: Estándar de codificación de Java.

1. Ficheros fuente Java

Cada fichero fuente Java contiene una única clase o interfaz pública. Cuando algunas clases o interfaces privadas están asociadas a una clase pública, pueden ponerse en el mismo fichero que la clase pública. La clase o interfaz pública debe ser la primera clase o interface del fichero.

Los ficheros fuentes Java tienen la siguiente ordenación:

- Comentarios de comienzo
- Sentencias package e import
- Declaraciones de clases e interfaces

1.1 Comentarios de comienzo

Todos los ficheros fuente deben comenzar con un comentario en el que se lista el nombre de la clase, información de la versión, fecha, y copyright:

```
/*
 * Nombre de la clase
 *
 * Informacion de la version
 *
 * Fecha
 *
 * Copyright
 */
```

2. Indentación

Se deben emplear cuatro espacios como unidad de indentación. La construcción exacta de la indentación (espacios en blanco contra tabuladores) no se especifica. Los tabuladores deben ser exactamente cada 8 espacios (no 4).

3. Longitud de la línea

Evitar las líneas de más de 80 caracteres, ya que no son manejadas bien por muchas terminales y herramientas.

Nota: Ejemplos para uso en la documentación deben tener una longitud inferior, generalmente no más de 70 caracteres.

3.1 Rompiendo líneas

Cuando una expresión no entre en una línea, romperla de acuerdo con estos principios:

- Romper después de una coma.
- Romper antes de un operador.
- Preferir roturas de alto nivel (más a la derecha que el "padre") que de bajo nivel (más a la izquierda que el "padre").
- Alinear la nueva línea con el comienzo de la expresión al mismo nivel de la línea anterior.
- Si las reglas anteriores llevan a código confuso o a código que se aglutina en el margen derecho, indentar justo 8 espacios en su lugar.

Dos ejemplos de ruptura de líneas en expresiones aritméticas. Se prefiere el primero, ya que el salto de línea ocurre fuera de la expresión que encierra los paréntesis.

```
nombreLargo1 = nombreLargo2 * (nombreLargo3 + nombreLargo4
               - nombreLargo5) + 4 * nombreLargo6; // PREFERIDA


nombreLargo1 = nombreLargo2 * (nombreLargo3 + nombreLargo4
                             - nombreLargo) + 4 * nombreLargo6;
// EVITAR
```

Dos ejemplos de indentación de declaraciones de métodos. El primero es el caso convencional. El segundo conduciría la segunda y la tercera línea demasiado hacia la izquierda con la indentación convencional, así que en su lugar se usan 8 espacios de indentación.

```
//INDENTACION CONVENCIONAL
unMetodo(int anArg, Object anotherArg, String yetAnotherArg,
         Object andStillAnother) {
    ...
}

//INDENTACION DE 8 ESPACIOS PARA EVITAR GRANDES INDENTACIONES
private static synchronized metodoDeNombreMuyLargo(int unArg,
        Object otroArg, String todaviaOtroArg,
        Object unOtroMas) {
    ...
}
```

Saltar de líneas por sentencias if deberá seguir generalmente la regla de los 8 espacios, ya que la indentación convencional (4 espacios) hace difícil ver el cuerpo. Por ejemplo:

```
//NO USAR ESTA INDENTACION
if ((condicion1 && condicion2)
    || (condicion3 && condicion4)
    ||!(condicion5 && condicion6)) { //MALOS SALTOS
    hacerAlgo();                    //HACEN ESTA LINEA FACIL
DE OLVIDAR
}

//USE THIS INDENTATION INSTEAD
if ((condicion1 && condicion2)
        || (condicion3 && condicion4)
        ||!(condicion5 && condicion6)) {
    hacerAlgo();
}
```

```
//O USAR ESTA
if ((condicion1 && condicion2) || (condicion3 && condicion4)
        ||!(condicion5 && condicion6)) {
    hacerAlgo();
}
```

4. Comentarios

Los programas Java pueden tener dos tipos de comentarios: comentarios de implementación y comentarios de documentación. Los comentarios de implementación son aquellos que también se encuentran en C++, delimitados por /*...*/, y //. Los comentarios de documentación (conocidos como "doc comments") existen sólo en Java, y se limitan por /**...*/. Los comentarios de documentación se pueden exportar a ficheros HTML con la herramienta javadoc.

Los comentarios de implementación son para comentar nuestro código o para comentarios acerca de una implementación particular. Los comentarios de documentación son para describir la especificación del código, libre de una perspectiva de implementación, y para ser leídos por desarrolladores que pueden no tener el código fuente a mano.

Se deben usar los comentarios para dar descripciones de código y facilitar información adicional que no es legible en el código mismo. Los comentarios deben contener sólo información que es relevante para la lectura y entendimiento del programa.

4.1 Formatos de los comentarios de implementación

Los programas pueden tener cuatro estilos de comentarios de implementación: de bloque, de una línea, de remolque, y de fin de línea.

4.1.1 Comentarios de bloque

Los comentarios de bloque se usan para dar descripciones de ficheros, métodos, estructuras de datos y algoritmos. Los comentarios de bloque se podrán usar al comienzo de cada fichero o antes de cada

método. También se pueden usar en otro lugares, tales como el interior de los métodos. Los comentarios de bloque en el interior de una función o método deben ser indentados al mismo nivel que el código que describen.

Un cometario de bloque debe ir precedido por una línea en blanco que lo separe del resto del código.

```
/*
 * Aqui hay un comentario de bloque.
 */
```

4.1.2 Comentarios de una línea

Pueden aparecer comentarios cortos de una única línea al nivel del código que siguen. Si un comentario no se puede escribir en una línea, debe seguir el formato de los comentarios de bloque. Un comentario de una sola línea debe ir precedido de una línea en blanco. Aquí un ejemplo de comentario de una sola línea en código Java:

```
if (condicion) {

    /* Código de la condicion. */
    ...
}
```

4.1.3 Comentarios de remolque

Pueden aparecer comentarios muy pequeños en la misma línea que describen, pero deben ser movidos lo suficientemente lejos para separarlos de las sentencias. Si más de un comentario corto aparece en el mismo trozo de código, deben ser indentados con la misma profundidad.

Aquí un ejemplo de comentario de remolque:

```
if (a == 2) {
    return TRUE;            /* caso especial */
} else {
    return isPrime(a);      /* caso gerenal */
}
```

4.1.4 Comentarios de fin de línea

El delimitador de comentario // puede convertir en comentario una línea completa o una parte de una línea. No debe ser usado para hacer comentarios de varias líneas consecutivas; sin embargo, puede usarse en líneas consecutivas para comentar secciones de código.

```
if (foo > 1) {

    // Hacer algo.
    ...
}
else {
    return false;          // Explicar aqui por que.
                                                    //}
```

4.2 Comentarios de documentación

Los comentarios de documentación describen clases Java, interfaces, constructores, métodos y atributos. Cada comentario de documentación se encierra con los delimitadores de comentarios /**...*/, con un comentario por clase, interface o miembro (método o atributo). Este comentario debe aparecer justo antes de la declaración:

```
/**
 * La clase Ejemplo ofrece ...
 */
public class Ejemplo { ...
```

La primera línea de un comentario de documentación (/**) para clases e interfaces no esta indentada, subsecuentes líneas tienen cada una un espacio de indentación (para alinear verticalmente los asteriscos). Los miembros, incluidos los constructores, tienen cuatro espacios para la primera línea y 5 para las siguientes.

Los comentarios de documentación no deben colocarse en el interior de la definición de un método o constructor, ya que Java asocia los comentarios de documentación con la primera declaración después del comentario.

5. Declaraciones

5.1 Cantidad por línea

Se recomienda una declaración por línea, ya que facilita los comentarios.

```
int nivel; // nivel de indentación
int tam;   // tamaño de la tabla
```

5.2 Inicialización

Intentar inicializar las variables locales donde se declaran. La única razón para no inicializar una variable donde se declara es si el valor inicial depende de algunos cálculos que deben ocurrir.

5.3 Colocación

Poner las declaraciones sólo al principio de los bloques (un bloque es cualquier código encerrado por llaves "{" y "}".) No esperar al primer uso para declararlas; puede confundir a programadores y limitar la portabilidad del código dentro de su ámbito de visibilidad.

```
void myMethod() {
    int int1 = 0;          // comienzo del bloque del método

    if (condition) {
        int int2 = 0;      // comienzo del bloque del "if"
        ...
    }
}
```

5.4 Declaraciones de class e interfaces

Al codificar clases e interfaces de Java, se siguen las siguientes reglas de formato:

- Ningún espacio en blanco entre el nombre de un método y el paréntesis "(" que abre su lista de parámetros
- La llave de apertura "{" aparece al final de la misma línea de la sentencia declaración.

- La llave de cierre "}" empieza una nueva línea indentada para ajustarse a su sentencia de apertura correspondiente, excepto cuando no existen sentencias entre ambas, que debe aparecer inmediatamente después de la de apertura "{".

```
class Ejemplo extends Object {
    int ivar1;
    int ivar2;

    Ejemplo(int i, int j) {
        ivar1 = i;
        ivar2 = j;
    }

    int metodoVacio() {}

    ...
}
```

- Los métodos se separan con una línea en blanco

6. Sentencias

6.1 Sentencias simples

Cada línea debe contener como mucho una sentencia. Ejemplo:

```
argv++;              // Correcto
argc--;              // Correcto
argv++; argc--;      // EVITAR!
```

6.2 Sentencias compuestas

Las sentencias compuestas son sentencias que contienen listas de sentencias encerradas entre llaves "{sentencias}".

- Las sentencias encerradas deben indentarse un nivel más que la sentencia compuesta.
- La llave de apertura se debe poner al final de la línea que comienza la sentencia compuesta; la llave de cierre debe empezar una nueva línea y ser indentada al mismo nivel que el principio de la sentencia compuesta
- Las llaves se usan en todas las sentencias, incluso las simples, cuando forman parte de una estructura de control, como en las sentencias if-else o for. Esto hace más sencillo añadir sentencias sin incluir bugs accidentalmente por olvidar las llaves.

6.3 Sentencias de retorno

Una sentencia return con un valor no debe usar paréntesis a menos que hagan el valor de retorno más obvio de alguna manera. Ejemplo:

```
return;

return miDiscoDuro.size();

return (tamanyo ? tamanyo : tamanyoPorDefecto);
```

6.4 Sentencias if, if-else, if else-if else

La clase de sentencias if-else debe tener la siguiente forma:

```
if (condicion) {
    sentencias;
}

if (condicion) {
    sentencias;
} else {
    sentencias;
}

if (condicion) {
    sentencia;
} else if (condicion) {
    sentencia;
} else{
    sentencia;
}


if (condicion) //EVITAR! ESTO OMITE LAS LLAVES {}!
    sentencia;
```

6.5 Sentencias for

Una sentencia for debe tener la siguiente forma:

```
for (inicializacion; condicion; actualizacion) {
    sentencias;
}
```

Al usar el operador coma en la clausula de inicialización o actualización de una sentencia for, evitar la complejidad de usar más de tres variables. Si se necesita, usar sentencias separadas antes de bucle for (para la clausula de inicialización) o al final del bucle (para la clausula de actualización).

6.6 Sentencias while

Una sentencia while debe tener la siguiente forma:

```
while (condicion) {
    sentencias;
}
```

6.7 Sentencias do-while

Una sentencia do-while debe tener la siguiente forma:

```
do {
     sentencias;
} while (condicion);
```

Sentencias switch

Una sentencia switch debe tener la siguiente forma:

```
switch (condicion) {
case ABC:
    sentencias;
    /* este caso se propaga */

case DEF:
    sentencias;
    break;

case XYZ:
    sentencias;
    break;

default:
    sentencias;
    break;
}
```

Cada vez que un caso se propaga (no incluye la sentencia break), añadir un comentario donde la sentencia break se encontraría normalmente. Esto se muestra en el ejemplo anterior con el comentario /* este caso se propaga */.

6.8 Sentencias try-catch

Una sentencia try-catch debe tener la siguiente forma:

```
try {
    sentencias;
} catch (ExceptionClass e) {
    sentencias;
}
```

Una sentencia try-catch puede ir seguida de un finally, cuya ejecución se ejecutará independientemente de que el bloque try se haya completado con éxito o no.

```
try {
    sentencias;
} catch (ExceptionClass e) {
    sentencias;
} finally {
    sentencias;
}
```

7. Espacios en blanco

7.1 Líneas en blanco

Las líneas en blanco mejoran la facilidad de lectura separando secciones de código que están lógicamente relacionadas.

Se deben usar siempre dos líneas en blanco en las siguientes circunstancias:

- Entre las secciones de un fichero fuente
- Entre las definiciones de clases e interfaces.
- Se debe usar siempre una línea en blanco en las siguientes circunstancias:
- Entre métodos
- Entre las variables locales de un método y su primera sentencia
- Antes de un comentario de bloque o de un comentario de una línea

- Entre las distintas secciones lógicas de un método para facilitar la lectura.

7.2 Espacios en blanco

Se deben usar espacios en blanco en las siguientes circunstancias:

- Una palabra clave del lenguaje seguida por un paréntesis debe separarse por un espacio.
- Debe aparecer un espacio en blanco después de cada coma en las listas de argumentos.
- . Todos los operadores binarios excepto. Se deben separar de sus operando con espacios en blanco. Los espacios en blanco no deben separar los operadores unarios, incremento ("++") y decremento ("--") de sus operando. Ejemplo:

```
a += c + d;
a = (a + b) / (c * d);

while (d++ == s++) {
    n++;
}
printSize("el tamaño es " + foo + "\n");
```

8. Hábitos de programación

8.1 Proporcionando acceso a variables de instancia y de clase

No hacer ninguna variable de instancia o clase pública sin una buena razón. A menudo las variables de instancia no necesitan ser asignadas/consultadas explícitamente, a menudo esto sucede como efecto lateral de llamadas a métodos. Un ejemplo apropiado de una variable de instancia pública es el caso en que la clase es esencialmente una estructura de datos, sin comportamiento. En otras palabras, si usarías la palabra struct en lugar de una clase (si Java soportara struct), entonces es adecuado hacer las variables de instancia públicas.

8.2 Referencias a variables y métodos de clase

Evitar usar un objeto para acceder a una variable o método de clase (static). Usar el nombre de la clase en su lugar. Por ejemplo:

```
metodoDeClase();              //OK
UnaClase.metodoDeClase();     //OK
unObjeto.metodoDeClase();     //EVITAR!
```

8.3 Constantes

Las constantes numéricas (literales) no se deben codificar directamente, excepto -1, 0, y 1, que pueden aparecer en un bucle for como contadores.

8.4 Asignaciones de variables

Evitar asignar el mismo valor a varias variables en la misma sentencia. Es difícil de leer.
Ejemplo:

```
fooBar.fChar = barFoo.lchar = 'c'; // EVITAR!
```

No usar asignación embebidas como un intento de mejorar el rendimiento en tiempo de ejecución. Ese es el trabajo del compilador. Ejemplo:

```
d = (a = b + c) + r;          // EVITAR!
```

Se debe escribir:

```
a = b + c;
d = a + r;
```

8.4.1 Paréntesis

En general es una buena idea usar paréntesis en expresiones que implican distintos operadores para evitar problemas con el orden de precedencia de los operadores. Incluso si parece claro el orden de precedencia de los operadores, podría no ser así para otros, no se debe asumir que otros programadores conozcan el orden de precedencia.

```
if (a == b && c == d)      // EVITAR!
if ((a == b) && (c == d)) // CORRECTO
```

8.4.2 Valores de retorno

Intentar hacer que la estructura del programa se ajuste a su intención. Ejemplo:

```
if (expresionBooleana) {
    return true;
} else {
    return false;
}
```

Anexo 2: Descripción de clases y operaciones.

Tabla 1: Descripción de la Clase ActualizarEFController.

Nombre:	ActualizarEFController	
Tipo de clase	Controladora (MultiActionController)	
Atributo		**Tipo**
gestionOperaciones fichaService actualizarEFService		GestionOperaciones FichaService ActualizarEFService
Responsabilidades		
Nombre:	**Descripción:**	
buscarCronogramasAEF(HttpServletRequest request, HttpServletResponse response)	Método para buscar el listado de los cronogramas de los proyectos a los cuales se les puede actualizar la ejecución financiera (EF) que cada actividad.	
detallesProyectoAEF(HttpServletRequest request, HttpServletResponse response)	Método para mostrar los detalles específicos del proyecto al cual se le desea actualizar la EF.	
actividadesProyectoAEF(HttpServletRequest request, HttpServletResponse response)	Método para mostrar todas las actividades a las que se les va a actualizar la EF.	
actualizarEF(HttpServletRequest request, DFicha_Proyecto ficha)	Método para actualizar la ejecución financiera del proyecto que se seleccionó.	

Tabla 2: Descripción de la Clase DetallesComunesController.

Nombre:	ActualizarEFController	
Tipo de clase	Controladora (Controller)	
Atributo		**Tipo**
nomencladorService iniciarProyectoService fichaService busquedaService cronogramaService solicitudesService		NomencladorService IniciarProyecto FichaService BusquedaService CronogramService GestionSolicitudes
Responsabilidades		
Nombre:	**Descripción:**	

getMinisteriosContraparte(HttpServletRequest request, HttpServletResponse response)	Método para buscar el listado de los ministerios (encargados de revisar las modificaciones propuestas de los entes) que son contrapartes del usuario logueado.
getEntesContraparte(HttpServletRequest request, HttpServletResponse response)	Método para buscar el listado de los entes (encargados de darle seguimiento a los proyectos) que son contrapartes del usuario loqueado.
getMixtas(HttpServletRequest request, HttpServletResponse response)	Método para buscar el listado de las mixtas (es donde se firman los proyectos) disponibles.
getPlanOpProyectosContratados(HttpServletRequest request, HttpServletResponse response)	Método para buscar el listado de los proyectos contratados que estén relacionados con el usuario logueado.
getDetallesGeneralesPlanOp(HttpServletRequest request, HttpServletResponse response)	Método para buscar la información general del plan operativo de un determinado proyecto.
getDetallesActividades(HttpServletRequest request, HttpServletResponse response)	Método para buscar la información del conjunto de actividades que conforman el plan operativo de un determinado proyecto.
getDetallesRecursosHumanos(HttpServletRequest request, HttpServletResponse response)	Método para buscar la información de los recursos humanos asociados a una determinada actividad.
getDetallesRecursosMateriales(HttpServletRequest request, HttpServletResponse response)	Método para buscar la información de los recursos materiales asociados a una determinada actividad.
getDetallesGastosAdministrativos(HttpServletRequest request, HttpServletResponse response)	Método para buscar la información de los gastos administrativos de una determinada actividad.
getDetallesOtrosRecursos(HttpServletRequest request, HttpServletResponse response)	Método para buscar la información de otros tipos de recursos asociados a una determinada actividad.
getCronogramas(HttpServletRequest request, HttpServletResponse response)	Método para buscar los datos de los cronogramas de ejecución financiera de los proyectos.
getDetallesCronograma(HttpServletRequest request, HttpServletResponse response)	Método para buscar los detalles del cronograma de ejecución financiera de un proyecto.
getDesembolsoPorActividades(request, HttpServletResponse response)	Método para obtener los desembolsos planificados en el cronograma de ejecución financiera, tanto el desembolso que se va a trasferir a Cuba como el que se va a invertir en Venezuela en cada uno de los meses que dura el proyecto.
getDesembolsos(request, HttpServletResponse response)	Método para mostrar todos los desembolsos por cada actividad en cada uno de los meses que dura un determinado proyecto.
getSolcitudesPago(HttpServletRequest request, HttpServletResponse response)	Método para buscar todas las solicitudes de pagos realizadas tanto por la parte cubana como por la venezolana.

descargarActaInicio(HttpServletRequest request, HttpServletResponse response)	Método que permite descargar un documento con el acta de inicio que deben adjuntarse a los proyectos que se desean iniciar.
descargarActaFin(HttpServletRequest request, HttpServletResponse response)	Método que permite descargar un documento con el acta de fin que deben adjuntarse a los proyectos que se desean terminar.

Tabla 3: Descripción de la Clase DetallesModContrato.

Nombre:	DetallesModContrato	
Tipo de clase	Controladora (MultiActionController)	
Atributo		**Tipo**
busquedaContratoService procesoService contratoService		BusquedaContratoService ProcesoService ContractService
Responsabilidades		
Nombre:	**Descripción:**	
busquedaContratos(HttpServletRequest request, HttpServletResponse response)	Método para buscar el listado de los contratos que han sido firmados.	
detallesContratoProcesos(HttpServletRequest request, HttpServletResponse response)	Método para mostrar los detalles específicos de un determinado contrato.	
descargarFileSeguimiento(HttpServletRequest request, HttpServletResponse response)	Método que permite descargar el documento de un contrato determinado.	
getProyectosPorContrato(**int** idContrato, HttpServletRequest request)	Método para mostrar los datos de los proyectos que están incluidos en un contrato.	
datosAdendum(**int** idContrato)	Método para mostrar los datos de los Adendums (documentos adjuntos a los contratos modificados) que están incluidos en un contrato.	

Tabla 4: Descripción de la Clase GestionarModificacionContrato

Nombre:	GestionarModificacionContrato	
Tipo de clase	Controladora (MultiActionController)	
Atributo		**Tipo**
gestionarModificacionContrato contratoService		GestionContratos ContractService
Responsabilidades		
Nombre:	**Descripción:**	
subirFichero(HttpServletRequest request, HttpServletResponse response)	Método que permite adjuntar un documento al contrato que se va a modificar.	

gestionarModificacion(HttpServletRequest request, HttpServletResponse response)	Método que permite modificar un contrato.
eliminarModificacionContrato(HttpServletRequest request, HttpServletResponse response)	Método que permite eliminar la modificación de un contrato.
aceptarModificacionContrato(HttpServletRequest request, HttpServletResponse response)	Método que permite aceptar la modificación de un contrato.
rechazarModificacionContrato(HttpServletRequest request, HttpServletResponse response)	Método que permite rechazar la modificación de un contrato.
gestionarEnvio(HttpServletRequest request, HttpServletResponse response)	Método que permite proponer el envío de la modificación de un contrato a un nivel superior (M o ST).
firmarContrato(HttpServletRequest request, HttpServletResponse response)	Método que permite firmar un contrato que haya sido modificado.

Tabla 5: Descripción de la Clase IniciarProyectoController

Nombre:	IniciarProyectoController	
Tipo de clase	Controladora (Controller)	
Atributo		**Tipo**
gestionOperaciones fichaService iniciarProyectoService ejecucionFisicaService		GestionOperaciones FichaService IniciarProyecto EjecucionFisica
Responsabilidades		
Nombre:	**Descripción:**	
getProyectosIniciadosTerminados(HttpServletRequest request, HttpServletResponse response)	Método que devuelve un listado de los proyectos que deben ser iniciados o terminados según el proceso de de Iniciar o Terminar un proyecto.	
iniciar_TerminarProyecto(HttpServletRequest request, HttpServletResponse response)	Método que permite proponer el inicio o terminación de un proyecto según sea el caso.	
proponerActualizacionEjFisica(HttpServletRequest request,HttpServletResponse response)	Método que permite proponer la actualización de la ejecución física de un proyecto que esté iniciado.	
aceptarRechazarAEjFisica(HttpServletRequest request, HttpServletResponse response)	Método que permite aceptar o rechazar la propuesta de actualización de la ejecución física de un proyecto.	
eliminarPropuesta(HttpServletRequest request, HttpServletResponse response)	Método que permite eliminar una propuesta, ya sea de actualización, de inicio o de terminación de un proyecto.	
adjuntarActaInicio(HttpServletRequest request, HttpServletResponse response)	Método que permite adjuntar el documento del acta de inicio de un proyecto al ser propuesto para ser iniciado.	

adjuntarActaFin(HttpServletRequest request, HttpServletResponse response)	Método que permite adjuntar el documento del acta de fin de un proyecto al ser propuesto para ser terminado.
verActa(HttpServletRequest request, HttpServletResponse response)	Método que permite ver el documento del acta de inicio y fin de un proyecto.

Tabla 6: Descripción de la Clase InspeccionController

Nombre:	InspeccionController	
Tipo de clase	Controladora (MultiActionController)	
Atributo		**Tipo**
inspeccionService fichaService		GestionObservacionInspeccion FichaService
Responsabilidades		
Nombre:	**Descripción:**	
buscarInspecciones(HttpServletRequest request, HttpServletResponse response)	Método para buscar los datos de las inspecciones que se le han realizado a los proyectos.	
buscarProyectos(HttpServletRequest request, HttpServletResponse response)	Método para buscar el listado de los proyectos que estén relacionados con el usuario logueado.	
buscarEntesPorProyecto(HttpServletRequest request, HttpServletResponse response)	Método para buscar los entes del proyecto al cual se le va a realizar la inspección.	
agregarInspeccion(HttpServletRequest request, HttpServletResponse response)	Método que permite realizarle una inspección a un proyecto.	
modificarInspeccion(HttpServletRequest request, HttpServletResponse response)	Método que permite modificar una determinada inspección.	
eliminarInspeccion(HttpServletRequest request, HttpServletResponse response)	Método que permite eliminar una determinada inspección.	

Tabla 7: Descripción de la Clase ModificarFichaController

Nombre:	ModificarFichaController	
Tipo de clase	Controladora (Controller)	
Atributo		**Tipo**
nomencladorService reformularFichaService fichaService		NomencladorService ReformularFichaService FichaService
Responsabilidades		
Nombre:	**Descripción:**	
getFichasProyectos(HttpServletRequest request, HttpServletResponse response)	Método para buscar los datos de las fichas de los proyectos.	

getModalidades(HttpServletResponse response)	Método para buscar las modalidades disponibles que pueden tener los proyectos.
getDetallesFichaProyecto(HttpServletRequest request, HttpServletResponse response)	Método que permite buscar los detalles de una determinada ficha de proyecto.
modificarDatosBasicosFichaProyecto(HttpServletRequest request, HttpServletResponse response)	Método que permite modificar los datos generales de una ficha de proyecto, guardando los datos en el servidor para luego ser enviados a la base de datos a través del siguiente método.
modificarFichaProyecto(HttpServletRequest request, HttpServletResponse response)	Método que envía la propuesta de reformulación de la ficha de proyecto.
aceptarProyecto(HttpServletRequest request, HttpServletResponse response)	Método que permite aceptar una propuesta de envío o reformulación de una determinada ficha de proyecto.
rechazarProyecto(HttpServletRequest request, HttpServletResponse response)	Método que permite rechazar una propuesta de envío o reformulación de una ficha de proyecto.
eliminarPropuestaRefFicha(HttpServletRequest request, HttpServletResponse response)	Método que permite eliminar una propuesta de reformulación de una ficha de proyecto.
validarReformularProyecto(HttpServletRequest request, HttpServletResponse response)	Método que permite validar una propuesta de reformulación de una ficha de proyecto.

Tabla 8: Descripción de la Clase ReplanificarCronogramaController

Nombre:	ReplanificarCronogramaController
Tipo de clase	Controladora (MultiActionController)
Atributo	**Tipo**
cronogramaService gestionCronogramaService busquedaService	CronogramService GestionCronograma BusquedaService
Responsabilidades	
Nombre:	**Descripción:**
agregarMonto(HttpServletRequest request, HttpServletResponse response)	Método que permite replanificar el monto a invertir o transferir de acuerdo a la parte que esté loqueada (Cuba, Venezuela) en un mes determinado, es decir, para agregar el monto que se va a cobrar en el mes especificado.
modificarMonto(HttpServletRequest request, HttpServletResponse response)	Método que permite modificar el monto que se va transferir o invertir en un determinado mes.
eliminarMonto(HttpServletRequest request, HttpServletResponse response)	Método que permite eliminar el monto replanificado para un determinado mes.
enviarPropuestaCronograma(HttpServletRequest request, HttpServletResponse response)	Método que permite enviar la propuesta de replanificación del cronograma de ejecución financiera de un proyecto.

eliminarPropuestaCronograma(HttpServletRequest request, HttpServletResponse response)	Método que permite eliminar la propuesta de replanificación del cronograma de ejecución financiera de un proyecto.
validarModificacionCronograma(HttpServletRequest request, HttpServletResponse response)	Método para validar la correcta modificación del cronograma de ejecución financiera.

Tabla 9: Descripción de la Clase RevisarCronogramaController

Nombre:	RevisarCronogramaController	
Tipo de clase	Controladora (Controller)	
Atributo		**Tipo**
cronogramaService gestionCronogramaService		CronogramService GestionCronograma
Responsabilidades		
Nombre:	**Descripción:**	
aceptarCronograma (HttpServletRequest request, HttpServletResponse response)	Método que permite aceptar una propuesta de replanificación del cronograma de ejecución financiera de un proyecto.	
rechazarCronograma (HttpServletRequest request, HttpServletResponse response)	Método que permite rechazar la propuesta de replanificación del cronograma de ejecución financiera de un proyecto.	

Tabla 10: Descripción de la Clase ReplanificarPlanOperativoController

Nombre:	ReplanificarPlanOperativoController	
Tipo de clase	Controladora (Controller)	
Atributo		**Tipo**
nomencladorService gestionOperaciones ejecucionFisica planOperativo fichaService		NomencladorService GestionOperaciones EjecucionFisica PlanOperativo FichaService
Responsabilidades		
Nombre:	**Descripción:**	
getViatico(HttpServletRequest request, HttpServletResponse response)	Método que permite obtener el viático de un recurso humano.	
getEntesProyecto(HttpServletRequest request, HttpServletResponse response)	Método que permite obtener los entes responsables de un proyecto.	

getCategoriasRHumano(HttpServletRequest request, HttpServletResponse response)	Método para buscar las diferentes tipos de categorías de recursos humanos.
getTipoRecursos(HttpServletRequest request, HttpServletResponse response)	Método para buscar los tipos de recursos que se le pueden asociar a una actividad.
gestionarActividadSiguiente(HttpServletRequest request, HttpServletResponse response)	Método que permite agregar y modificar los datos básicos de una actividad determinada.
gestionarRecursoHumano(HttpServletRequest request, HttpServletResponse response)	Método que permite agregar un nuevo recurso humano a una actividad y modificar o eliminar uno existente.
gestionarRecursoMaterial(HttpServletRequest request, HttpServletResponse response)	Método que permite agregar un nuevo recurso material a una actividad y modificar o eliminar uno existente.
gestionarGastoAdministrativo(HttpServletRequest request, HttpServletResponse response)	Método que permite agregar un nuevo gasto administrativo a una actividad y modificar o eliminar uno existente.
gestionarOtroRecurso(HttpServletRequest request, HttpServletResponse response)	Método que permite agregar otro tipo de recurso a una actividad y modificar o eliminar otro tipo de recurso existente.
finalizarGestionarActividad(HttpServletRequest request, HttpServletResponse response)	Método que permite agregar una actividad al plan operativo de un proyecto con todos los recursos asociados a esta y modificar o eliminar una existente.
cancelarModificarActividad(HttpServletRequest request)	Método que permite cancelar la modificación de una determinada actividad.
enviarPropReplanificacionPlanOP(HttpServletRequest request, HttpServletResponse response)	Método que permite enviar la propuesta de replanificación del plan operativo de un proyecto al ente contraparte.
eliminarPropuestaRPO(HttpServletRequest request, HttpServletResponse response)	Método que permite eliminar una propuesta de replanificación del plan operativo de un proyecto.
validarReplanificacion(HttpServletRequest request, HttpServletResponse response)	Método para validar la correcta replanificación del plan operativo de un proyecto garantizando que no se alteren los montos definidos al darle financiamiento al proyecto.

Tabla 11: Descripción de la Clase GestionarAprobacionComunController.

Nombre:	GestionarAprobacionComunController
Tipo de clase	Controladora (Controller)
Atributo	**Tipo**
planOperativo	PlanOperativo
Responsabilidades	

Nombre:	Descripción:
aceptarPropuestaPlanOP(HttpServletRequest request, HttpServletResponse response)	Método que permite aceptar la propuesta de envío o replanificación del plan operativo de un proyecto.
rechazarPropuestaPlanOP(HttpServletRequest request, HttpServletResponse response)	Método que permite rechazar la propuesta de envío o replanificación del plan operativo de un proyecto.
propuestaEnvioPlanOPNivelSuperior(HttpServletRequest request, HttpServletResponse response)	Método que propone enviar la propuesta de replanificación del plan operativo de un proyecto a un nivel superior (M o ST).

Tabla 12: Descripción de la Clase SolicitarPagoController.

Nombre:	SolicitarPagoController	
Tipo de clase	Controladora (Controller)	
Atributo		**Tipo**
gestionOperaciones solicitudesService		GestionOperaciones GestionSolicitudes
Responsabilidades		
Nombre:	**Descripción:**	
getCronogramasProyecto(HttpServletRequest request, HttpServletResponse response)	Método para buscar los cronogramas de los proyectos con los que está relacionado el usuario logueado para realizar las solicitudes de pago de los proyectos.	
getDatosSolicitudPago(HttpServletRequest request, HttpServletResponse response)	Método para obtener los datos generales de una determinada solicitud de pago de un proyecto.	
getFuentesF(HttpServletRequest request, HttpServletResponse response)	Método para buscar las fuentes de financiamiento a las cuales se les puede solicitar el pago del proyecto en un determinado mes.	
getFacturasSP(HttpServletRequest request, HttpServletResponse response)	Método para buscar los datos de las facturas de una determinada solicitud de pago.	
getMesesPorSolicitar(HttpServletRequest request, HttpServletResponse response)	Método para buscar los meses para los cuales no se ha solicitado el pago de un proyecto.	
agregarFactura(HttpServletRequest request)	Método que permite agregar una factura de una solicitud de pago de un proyecto.	
modificarFactura(HttpServletRequest request)	Método que permite modificar una factura de una solicitud de pago de un proyecto.	
eliminarFactura(HttpServletRequest request)	Método que permite eliminar una factura de una solicitud de pago de un proyecto.	
enviarSolicitudPago(HttpServletRequest request, HttpServletResponse response)	Método que permite enviar una solicitud de pago de un proyecto.	
aceptarRechazarSolicitud(HttpServletRequest request, HttpServletResponse response)	Método que permite a un ente venezolano aceptar y rechazar una solicitud de pago de un proyecto enviada por el ente cubano.	

modificarSolicitud(HttpServletRequest request, HttpServletResponse response)	Método que permite modificar una solicitud de pago de un proyecto tanto por la parte cubana como por la venezolana.
eliminarSolicitud(HttpServletRequest request, HttpServletResponse response)	Método que permite eliminar una solicitud de pago de un proyecto.
enviarSolicitudCubanaFF(HttpServletRequest request, HttpServletResponse response)	Método que permite que el ente venezolano envíe la solicitud de pago del ente cubano a la fuente de financiamiento encargada de realizar el pago.
confirmarPago(HttpServletRequest request, HttpServletResponse response)	Método que permite que el ente venezolano confirme el pago que solicitó.
validarModificacionSolicitud(HttpServletRequest request, HttpServletResponse response)	Método que permite validar la modificación de una solicitud de pago para evitar enviarla sin realizar cambios.

Tabla 13: Descripción de la Clase SeguimientoController.

Nombre:	SeguimientoController	
Tipo de clase	Controladora (Controller)	
Atributo		**Tipo**
Responsabilidades		
Nombre:	**Descripción:**	
handleRequest(HttpServletRequest request, HttpServletResponse response)	Método que permite controlar el acceso al módulo Seguimiento y las peticiones de los usuarios, enviando cada una de las vistas en dependencia de lo que solicite, en las cuales se va a mostrar la información que desee y la respuesta de las acciones ejecutadas.	

Tabla 14: Descripción de la Clase ValidacionDatosUtil.

Nombre:	ValidacionDatosUtil	
Tipo de clase	Util	
Atributo		**Tipo**
Responsabilidades		
Nombre:	**Descripción:**	
actFueModificada(DActividad actOrig, DActividad actMod)	Método que permite verificar si se modificó una determinada actividad de un proyecto.	
desembolsoFueModificado(DActividad actOrig, DActividad actMod)	Método que permite verificar si el desembolso que se va a transferir o invertir fue modificado y qué parte lo	

	modificó (cubana o venezolana).
cronogramaFueModificado(Set<DActividad> actOs,Set<DActividad> actMs)	Método que permite comprobar si el cronograma planificado de cada una de las actividades del proyecto fue modificado.
proyectoFueModificado(DFicha_Proyecto fO, DFicha_Proyecto fM)	Método que permite verificar si una determinada ficha de proyecto fue modificada.
proyectoModificadoOK(DFicha_Proyecto proyecto)	Método que permite verificar si el monto total del proyecto coincide con el monto replanificado en todas las actividades del proyecto.
rhWasModified(DRecurso_Humano rhO, DRecurso_Humano rhM)	Método que permite verificar si un recurso humano de una actividad fue modificado.
rmWasModified(DRecurso_Material rmO, DRecurso_Material rmM)	Método que permite verificar si un recurso material de una actividad fue modificado.
gaWasModified(DGasto_Administrativo gaO, DGasto_Administrativo gaM)	Método que permite verificar si un gasto administrativo de una actividad fue modificado.
orWasModified(DOtro_Recurso orO, DOtro_Recurso orM)	Método que permite verificar si fue modificado otro tipo de recurso de una actividad
monthWasPayed(List<Object[]> slctdes, String mes)	Método que permite verificar si se realizó el pago de un proyecto de las solicitudes realizadas en un mes determinado.
dsuario(HttpServletRequest request)	Método que permite obtener el usuario que está logueado.
duracionActividad(Date fI, Date fF)	Método que permite obtener la duración de una actividad en el formato X meses y X días si dura más de un mes, en caso contrario X días.
mesesProyecto(int mesI,int cantM, String year)	Método que permite obtener el listado de los meses que dura un proyecto teniendo en cuenta el mes y el año en que se inicia el mismo y la cantidad de meses que dura.
mesesDisponiblePorActividad(Set<DDesembolsoTransf> dt, Set<DDesembolsoInv> di,int mesI,int cantM, String year,List<Object[]> objSdes)	Método que permite obtener los meses a los cuales no se les ha solicitado el pago tanto por la parte cubana como por la venezolana.
estadoSolicitudPago(List<Object[]> sdes,List<String>meses,int parte)	Método que permite obtener el estado de todos los pagos solicitados de un proyecto.
convertBigDecimalToString(BigDecimal numero, int lugares)	Método que permite convertir un número del formato BigDecimal al formato String con los lugares decimales que se le especifique.
mes_YearByDate(Date f)	Método que permite convertir una fecha al formato String (cadena).
date_ByMesYear(String fecha)	Método que permite convertir una fecha del formato String al formato Date (fecha).

Tabla 15: Descripción de la Clase CalculosFichaUtil.

Nombre:	CalculosFichaUtil	
Tipo de clase	Util	
Atributo		**Tipo**
Responsabilidades		
Nombre:	**Descripción:**	
montoTotalTransferirInvertirPais(DFicha_Proyecto proyecto,DParte parte)	Método que permite obtener el monto total de lo que se va a invertir en Venezuela o a trasferir a Cuba de un proyecto.	
montoPaisActividad(DActividad act,DParte parte)	Método que permite obtener el monto de una actividad por cada parte (Cuba o Venezuela).	
montoTotalActividad(DActividad act)	Método que permite obtener el monto total de una actividad.	
viaticos(DFicha_Proyecto proyecto,DParte parte)	Método que permite obtener los viáticos de un proyecto por cada parte (Cuba o Venezuela).	
pasajesInternacionales(DFicha_Proyecto proyecto,DParte parte)	Método que permite obtener el monto de los pasajes utilizados por cada parte en un proyecto.	
adquisicionMateriales(DFicha_Proyecto proyecto,DParte parte)	Método que permite obtener el monto de todos los recursos materiales de utilizados por cada parte en un proyecto.	
gastosAdministrativos(DFicha_Proyecto proyecto,DParte parte)	Método que permite obtener el monto de todos los gastos administrativos de cada parte en un proyecto.	
otrosRecursos(DFicha_Proyecto proyecto,DParte parte)	Método que permite obtener el monto de los otros tipos de recursos empleados por cada parte en un proyecto.	
tiempoEjecucion(DActividad actividad)	Método que permite obtener la duración de una actividad en días.	
tiempo(DRecurso_Humano recurso_Humano)	Método que permite obtener la duración de un recurso humano en meses.	
subTotalPasajes(DRecurso_Humano recurso_Humano)	Método que permite obtener el total del monto de los pasajes de un recurso humano.	
subTotalHospedaje(DRecurso_Humano recurso_Humano)	Método que permite obtener el total del monto de los hospedajes de un recurso humano.	
subTotalHonorarios(DRecurso_Humano recurso_Humano)	Método que permite obtener el total del monto de los honorarios de un recurso humano.	
subTotalViaticos(DRecurso_Humano recurso_Humano)	Método que permite obtener el total del monto de los viáticos de un recurso humano.	
montoTotalRecursoHumano(DRecurso_Humano recurso_Humano)	Método que permite obtener el monto total de un recurso humano.	

montoTotalParteRecursoHumano(DRecurso_Humano recurso_Humano,DParte parte)	Método que permite obtener el monto total de un recurso humano por cada parte.
montoEjecutadoFichaProyecto(DFicha_Proyecto ficha_Proyecto)	Método que permite obtener el monto ejecutado de un proyecto.

Tabla 16: Descripción de la Clase PlanOperativoService

Nombre:	PlanOperativoService	
Tipo de clase	Servicio	
Atributo		**Tipo**
documentoservice fichaService gestionOperaciones		DocumentoService FichaService GestionOperaciones
Responsabilidades		
Nombre:	**Descripción:**	
modificarPlanOperativo(DFicha_Proyecto fichaProyecto,DNivel nivel,int a)	Método para guardar en la base de datos una modificación de un plan Operativo, crea una copia del plan operativo Original.	
aceptarPropuestaPlanOperativo(DFicha_Proyecto fichaProyecto,DNivel nivel)	Método para aceptar la propuesta del plan operativo, elimina la copia de la propuesta original cuando esta operación la realiza la Secretaria Técnica.	
enviarNivelSuperior(DFicha_Proyecto ficha_Proyecto,DNivel nivel)	Método para cambiar el estado del plan operativo para que pueda ser visto por la entidad superior.	
eliminarPropuestaPlanOperativo(DFicha_Proyecto ficha_Proyecto)	Método que permite eliminar los rechazos que se le realizan a las propuestas de modificación del plan operativo.	
rechazarPropuestaPlanOperativo(DFicha_Proyecto fichaProyecto,DUsuario usuario,DRechazo rechazo)	Método para que le cambie el estado al plan operativo, para que pueda ser revisado por el ente ejecutor.	

Tabla 17: Descripción de la Clase IniciarProyectoService.

Nombre:	IniciarProyectoService	
Tipo de clase	Servicio	
Atributo		**Tipo**
documentoService facturaDAO fichaService gestionOperaciones		DocumentoService FacturaDAO FichaService GestionOperaciones
Responsabilidades		

Nombre:	Descripción:
aceptarIniciarProyecto(DFicha_Proyecto fichaProyecto,DNivel nivel,DParte parte)	Método que le cambia el estado a una ficha de ficha contratada a ficha iniciada.
rechazarIniciarProyecto(DFicha_Proyecto fichaProyecto, DNivel nivel, DParte parte,DRechazo rechazo)	Método que permite a un usuario rechazar el inicio de un proyecto.
proponerIniciarTerminarproyecto(DFicha_ Proyecto fichaProyecto,DNivel nivel,int proces)	Método que le permite al usuario realizar una propuesta de inicio de una ficha de proyecto.
aceptarIniciarProyecto(DFicha_Proyecto fichaProyecto,DNivel nivel,DParte parte)	Método que permite una vez que los dos entes estén de acuerdo en darle inicio a una determinada ficha de proyecto, pasarlo al estado de proyecto iniciado.
eliminarIniciarTerminarProyecto(DFicha_Proyecto fichaProyecto)	Método que permite eliminar el rechazo de las propuestas de inicio y terminación de la fichas de proyectos.
cargarMixtas()	Método que carga todas las mixtas que se han realizado.
proyectosInciador_PropuestoInicio(DNivel nivel,String nombre, DEnte ente, DMinisterio ministerio, DMixta mixta)	Método que muestra todos los proyectos que se han iniciado o han sido propuestos para iniciar.
proyectosContratadosPropuestoTerminar(DNivel nivel,String nombre, DEnte ente, DMinisterio ministerio, DMixta mixta)	Método que muestra todos los proyectos que ya tienen su ejecución física al cien por ciento y están listos para terminar.
observacionesPorActividades(int idActividad)	Método que muestra todas las observaciones que se le han realizado a una actividad.

Tabla 18: Descripción de la Clase ReformularFichaService.

Nombre:	ReformularFichaService	
Tipo de clase	Servicio	
Atributo		**Tipo**
fichaService planOperativoDAO		FichaService PlanOperativoDAO
Responsabilidades		
Nombre:	**Descripción:**	
aceptarProyecto(DFicha_Proyecto fichaProyecto, DParte parte,NTipo_Nivel tipoNivel)	Método que permite aceptar la reformulación de una ficha de proyecto.	
modificarProyecto(DFicha_Proyecto fichaProyecto,DParte parte,NTipo_Nivel tipoNivel)	Método que permite modificar una ficha de proyecto por los entes, crea una copia de la ficha del proyecto original.	

modificarProyectoSecretaria(DFicha_Proyecto fichaProyecto,DParte parte,NTipo_Nivel tipoNivel)	Método que permite modificar una ficha de proyecto por las Secretarías Técnicas. Crea una copia de la ficha del proyecto Original.
eliminarProyecto(DFicha_Proyecto ficha_Proyecto)	Método que permite eliminar una propuesta de una reformulación de una ficha de proyecto.
cargarDocumentosProceso(DDocumento d, DParte p, NTipo_Nivel tipoNivel, DNivel nivel, String nombreFicha, DEnte ente, DMinisterio ministerio, DMixta mixta, int primero, int cantidad)	Método que permite buscar todos los proyectos que se encuentran en el proceso de reformular Plan Operativo o son posibles candidatos a ser reformulados.
rechazarProyecto(DFicha_Proyecto fichaProyecto,DParte parte,DRechazo rechazo,NTipo_Nivel tipoNivel)	Método que permite rechazar la propuesta de una reformulación de una ficha técnica, crea una nota de rechazo.

Tabla 19: Descripción de la Clase GestionCronogramaService

Nombre:	GestionCronogramaService	
Tipo de clase	Servicio	
Atributo		**Tipo**
procesoService documentoService planOperativoDAO facturaDAO rechazoDao cronogramaDao fichaService		ProcesoService DocumentoService DPlanOperativoDAO DFacturaDAO RechazoDao CronogramaDao FichaService
Responsabilidades		
Nombre:	**Descripción:**	
BuscarCronogramas(DNivel nivel, String nombre,int ente, int ministerio, int mixta)	Método para buscar el listado de los cronogramas de los proyectos a los cuales se les puede actualizar la ejecución financiera (EF) de cada actividad.	
aceptarPropuestaCronograma(DCronograma cronograma, DNivel nivel)	Método para aceptar las propuestas de las modificaciones realizadas.	
rechazarPropuestaCronograma(DCronograma cronograma, DNivel nivel, DRechazo rechazo)	Método que permite rechazar una propuesta de modificación de un cronograma de ejecución financiera.	
enviarNivelSuperior(DCronograma cronograma, DNivel nivel)	Método que permite enviar a un organismo superior la propuesta de la modificación de un cronograma de ejecución financiera.	
modificarPropuestaCronograma(DCronograma cronograma, DNivel nivel,int a)	Método permite realizar modificaciones a los cronogramas de ejecución financiera.	
eliminarPropuestaCronograma(DCronograma cronograma, DNivel nivel)	Método que permite eliminar las propuestas de modificación de los cronogramas de ejecución financiera.	

Tabla 20: Descripción de la Clase EjecucionFisicaService.

Nombre:	EjecucionFisicaService	
Tipo de clase	Servicio	
Atributo		**Tipo**
fichaService documentoservice planOperativoDAO facturaDAO		FichaService DocumentoService DPlanOperativoDAO DFacturaDAO
Responsabilidades		
Nombre:	**Descripción:**	
aceptarActualizacionActividad(DFicha_Proyecto fichaProyecto)	Método que permite actualizar la ejecución física de las actividades de una ficha de proyecto.	
rechazarActualizacionActividad(DFicha_Proyecto fichaProyecto, DNivel nivel, DRechazo rechazo)	Método que permite rechazar las actualizaciones de la ejecución física de las actividades de una ficha de proyecto. Crea una nota de rechazo.	
modificarActualizarEF(DFicha_Proyecto fichaProyecto, DNivel nivel)	Método que permite modificar el por ciento de la ejecución física de las actividades de una ficha de proyecto.	
BuscarProyectosActualizacion(DNivel nivel,String nombre, DEnte ente, DMinisterio ministerio, DMixta mixta)	Método que permite buscar todos los proyectos iniciados a los que se les puede actualizar la ejecución física.	
eliminarActualizacionEF(DFicha_Proyecto ficha_Proyecto)	Método que permite eliminar la propuesta de una actualización física de las actividades de una ficha de proyecto.	

Tabla 21: Descripción de la Clase GestionContratosService.

Nombre:	GestionContratosService	
Tipo de clase	Servicio	
Atributo		**Tipo**
adendumDao contratoDao contractService elaborarContratoService		AdendumDao ContratoDao ContractService ElaborarContratoService
Responsabilidades		
Nombre:	**Descripción:**	
crearAdendum(DAdendum adendum, DParte parte)	Método que permite crear un adendum que no son más que cambios que se adjuntan a un contrato.	
modificarAdendum(DAdendum adendum)	Método que permite realizar una modificación a los adendums adjuntos a los contratos.	

eliminarAdendum(DAdendum adendum)	Método que permite eliminar un adendum adjunto a un contrato.
enviarContrato(DContrato contrato, NTipo_Nivel tipoNivel, DParte parte)	Método que permite enviar a los organismos superiores la modificación de un contrato.
rechazarModificacion(DContrato contrato, NTipo_Nivel tipoNivel, DRechazo rechazo, DParte parte)	Método que permite realizar un rechazo a un contrato enviado por su contraparte.
modificarContrato(DContrato contrato, NTipo_Nivel tipoNivel, DParte parte)	Método que permite realizarle una modificación a un contrato existente.
aceptarModificacion(DContrato contrato, NTipo_Nivel tipoNivel, DParte parte	Método que permite aceptar la modificación de un contrato propuesto por su contraparte.
enviarContrato(DContrato contrato, NTipo_Nivel tipoNivel, DParte parte)	Método que permite enviar la modificación de un contrato a los organismos superiores.
aceptarEnvio(DContrato contrato, NTipo_Nivel tipoNivel, DParte parte)	Método que permite aceptar la propuesta de envío realizada por la contraparte.
rechazarEnvio(DContrato contrato, NTipo_Nivel tipoNivel, DRechazo rechazo, DParte parte)	Método que permite realizar un rechazo a una propuesta de envío realizada por la contraparte.
aceptarContrato(DContrato contrato, NTipo_Nivel tipoNivel, DParte parte)	Método que permite aceptar la propuesta de modificación de un contrato realizada por la contraparte.
rechazarContrato(DContrato contrato, NTipo_Nivel tipoNivel, DRechazo rechazo, DParte parte)	Método que permite rechazar la propuesta de modificación de un contrato realizada por la contraparte.
procesarOperacion(DContrato contrato, NTipo_Nivel tipoNivel, int operacion, DParte parte)	Método que permite tras realizada una operación, guardar los cambios en la base de datos
firmarContrato(DContrato contrato, NTipo_Nivel tipoNivel, DParte parte, List<DAdendum> adendums)	Método que permite a los ministerios luego de que la modificación del contrato sea aprobada por todos los niveles, firmar el contrato para hacer oficial el cambio.
eliminarAdendumDadoContrato(DContrato contrato)	Método que permite eliminar el adendum de un contrato.
cargarAdendumPorContrato(int idContrato)	Método que permite cargar todos los adendums adjuntos a un contrato.

Tabla 22: Descripción de la Clase BusquedaContratoService.

Nombre:	BusquedaContratoService
Tipo de clase	Servicio
Atributo	**Tipo**
contratoDao adendumDao rechazoDao	ContratoDao AdendumDao RechazoDao

Responsabilidades	
Nombre:	**Descripción:**
cantidadContratos(String nombreFicha, int idEnte,int idMinisterio, int idNivel,int idParte,int TipoNivel,int idMixta)	Método que permite saber la cantidad de parámetros dados los parámetros de búsquedas.
cargarContratos(String nombreFicha, int idEnte,int dMinisterio, int idNivel, int parte, int tipoNivel,int primerResultado, int cantidad,int idMixta)	Método que permite buscar todos los contratos que cumplen con los parámetros de búsqueda.
cargarDatosParaContratos(int idContrato)	Método que permite cargar todos los datos de un contrato.
cargarDocumentoAdjunto(int idContrato)	Método que permite cargar el documento adjunto de un contrato.
cargarDatosParaAdendum(int idContrato)	Método que permite cargar los datos de un adendum.
cargarFichasPorContrato(int idContrato)	Método que permite conocer cuáles son todas las fichas de proyecto que pertenecen a un contrato.
cargarDocumentoAdjuntoDadoIdAdendum(int idAdendum)	Método que permite conocer el documento adjunto a un adendum.
cargarContratosParaFirmar(String nombreFicha,int idEnte, int idMinisterio, int idNivel, int parte, int primerResultado, int cantidad,int idMixta)	Método que permite conocer cuáles son todos los contratos que han sido aprobados por todos los niveles.
cantidadContratosParaFirmar(String nombreFicha, int idEnte, int idMinisterio, int parte, int idNivel,int idMixta)	Método que permite conocer la cantidad de contratos que han sido aprobados por todos los niveles.
cantidadContratosPorDocumentoParaSeguimiento(String nombreFicha,int idEnte, int idMinisterio, DNivel nivel,int marcoAprobacion)	Método que muestra la información de los contratos que cumpla con los parámetros de búsqueda.

Tabla 23: Descripción de la Clase GestionDocumentosService.

Nombre:	GestionDocumentosService
Tipo de clase	Servicio
Atributo	**Tipo**
facturaDAO planOperativoDAO inspeccionDAO	DFacturaDAO DPlanOperativoDAO DInspeccionDAO
Responsabilidades	
Nombre:	**Descripción:**

cargarProyectosCotratados(DNivel nivel,String nombre, DEnte ente, DMinisterio ministerio, DMixta mixta)	Método que permite cargar todos los proyectos que ya han sido contratados y sean visibles para los entes o ministerios que se encuentren autenticados.
cargarProyectosCotratadosSecretaris(DNivel nivel,String nombre, DEnte ente, DMinisterio ministerio, DMixta mixta)	Método que permite cargar todos los proyectos que ya han sido contratados, este método es sólo para el caso de las Secretarías Técnicas.
rechazosPorDocumentos(int idDocumento)	Método que permite cargar todos los rechazos que se le han hecho a un documento determinado.
listaContratos(DNivel nivel, String nombre,DEnte ente, DMinisterio ministerio, DMixta mixta)	Método que permite conocer todos los contratos que se han realizado.
listaCronograma(DNivel nivel,String nombre, int ente, int ministerio, int mixta)	Método que permite conocer todos los cronogramas de ejecución financiera.
listaFichaProyectos(DNivel nivel, String nombre,DEnte ente, DMinisterio ministerio, DMixta mixta)	Método que permite conocer todas las fichas de proyecto que estén contratadas e iniciadas.
listaPlanOperativo(DNivel nivel, String nombre,DEnte ente, DMinisterio ministerio, DMixta mixta)	Método que permite conocer todos los planes operativos de todas las fichas de proyecto que se encuentren contratadas e iniciadas.
nombreProyectosPorUsuarios(int idNivel, int tipoNivel,int idMinisterio,int idEnte)	Método que permite conocer todos los nombres de todos los proyectos que pertenecen a un determinado usuario.
actasPorFichas(int idficha)	Método que permite conocer todas las actas de inicio y fin de todas las fichas que estén contratadas, iniciadas o terminadas.

Tabla 24: Descripción de la Clase GestionEjecucionFinancieraService.

Nombre:	GestionEjecucionFinancieraService	
Tipo de clase	Servicio	
Atributo		**Tipo**
planOperativoDAO gestionOperaciones fichaService		DPlanOperativoDAO GestionOperaciones FichaService
Responsabilidades		
Nombre:	**Descripción:**	
modificarEjecucionFinanciera(DFicha_Proyecto ficha_Proyecto)	Método que permite modificar la ejecución financiera de las fichas de proyecto que se encuentran en el estado de iniciada.	
cargarPropuestaActulizacionEF(DNivel nivel,String nombre, int idente, int idministario, int mixta)	Método que permite conocer cuáles son las fichas a las que se les ha propuesto actualizar la ejecución financiera.	

mesesSolicitudPagos(int idFicha, int parte)	Método que permite conocer cuáles son los meses en los cuales se harán las solicitudes de pagos del cronograma de ejecución financiera de un proyecto.

Tabla 25: Descripción de la Clase GestionObservacionInspeccionService.

Nombre:	GestionObservacionInspaccionService	
Tipo de clase	Servicio	
Atributo		**Tipo**
facturaDAO planOperativoDAO inspeccionDAO		DFacturaDAO DPlanOperativoDAO DInspeccionDAO
Responsabilidades		
Nombre:	**Descripción:**	
buscarFichasObservacionInspeccion(DNivel nivel,int idproyecto,DEnte ente,DMinisterio ministerio,DMixta mixta,java.util.Date desde,java.util.Date hasta)	Método que permite conocer todas las fichas de proyecto a las cuales se les ha realizado una observación o una inspección en un rango de fechas determinado.	
buscarProyectosEntes(DNivel nivel,DEnte ente,DMinisterio ministerio,DMixta mixta)	Método que permite conocer todos los proyectos que pertenecen al Ente que está autenticado.	
crearObsevacion(DInspeccion observacion)	Método que permite crear una inspección o una observación.	
modificarObservacion(DInspeccion obsevacion)	Método que permite modificar una observación o una inspección.	
eliminarObservacion(DInspeccion observacion)	Método que permite eliminar una observación o una inspección.	
InspeccionporId(int idinspeccion)	Método que permite conocer una inspección dado su identificador.	
entesPorNombreProyecto(int idFicha)	Método que permite conocer los dos entes que pertenecen a una ficha de proyecto.	

Tabla 26: Descripción de la Clase GestionSolicitudesService.

Nombre:	GestionSolicitudesService	
Tipo de clase	Servicio	
Atributo		**Tipo**
planOperativoDAO gestionOperaciones solicitudPagoDAO facturaDAO		DPlanOperativoDAO GestionOperaciones DSolicitudPagoDAO DFacturaDAO

documentoService	DocumentoService

Responsabilidades	
Nombre:	**Descripción:**
cargarSolicitudespagos(DNivel nivel,String nombre,int ente,int ministerio,int mixta)	Método que permite conocer cuáles son las fichas de proyectos a las que se les puede realizar solicitudes de pago.
fuenteFinanciamiento()	Método que permite conocer todas las fuentes de financiamiento existentes.
buscarRevisarSolicitudesPagos(DNivel nivel,String nombre, int ente, int ministerio, int mixta)	Método que permite revisar todas las solicitudes de pagos que se han realizado por el usuario.
solicitudPagoporId(int idsolicitud)	Método que permite conocer los datos de una solicitud de pago.
solicitudPorCronograma(int idCronograma)	Método que permite conocer todas las solicitudes de pagos realizadas según el cronograma.
aceptarSolicitud(DSolicitudPago solicitudPago, DNivel nivel)	Método que permite aceptar la solicitud de pago realizada por un ente cubano.
crearSolicitud(DSolicitudPago solicitudPago, DNivel nivel)	Método que permite crear una solicitud de pago.
rechazarSolicitud(DSolicitudPago solicitudPago, DNivel nivel,DRechazo rechazo)	Método que permite a un ente venezolano rechazar una solicitud de pago realizada por un ente cubano.
eliminarSolicitud(DSolicitudPago solicitudPago, DNivel nivel)	Método que permite eliminar una solicitud de pago rechazada.
enviarSolicitudFuentefinanciamiento(DSolicitudPago solicitudPago, DNivel nivel)	Método que permite a los entes venezolanos realizarles solicitudes de pagos a las fuentes de financiamiento.
modificarSolicitud(DSolicitudPago solicitudPago, DNivel nivel)	Método que permite modificar una solicitud de pago.
rechazarEnvioSolicitud(DSolicitudPago solicitudPago, DNivel nivel)	Método que permite rechazar el envío de una solicitud de pago a una fuente de financiamiento.
confirmarPagosSolicitud(DSolicitudPago solicitudPago)	Método que permite confirmar que ya se ha entregado el dinero de la solicitud de pago por parte de la fuente de financiamiento al ente solicitante.

Tabla 27: Descripción de la Clase GestionOperacionesService.

Nombre:	GestionOperacionesService	
Tipo de clase	Servicio	
Atributo		**Tipo**
facturaDAO		DFacturaDAO
Responsabilidades		
Nombre:	**Descripción:**	
porcientoEjecucionFisica(int idficha)	Método que permite conocer el porciento de ejecución física de un proyecto.	
porcinetoEjecucionFinanciera(int idficha)	Método que permite conocer el porciento de ejecución financiera de un proyecto.	
invertirEnVenezuela(int ficha)	Método que permite conocer la cantidad de dinero que se va a invertir por proyecto.	
transferirCuba(int ficha)	Método que permite conocer la cantidad de dinero a transferir por proyecto.	
financiamientoTotal(int ficha)	Método que permite conocer la cantidad de dinero que se la ha asignado a un proyecto.	
totaldeloRecursosHumanos(DRecurso_Humano humano)	Método que permite conocer la cantidad de dinero que se va a utilizar en el pago de recursos humanos.	
montoEjecutadoCuba(DFicha_Proyecto ficha_Proyecto)	Método que permite conocer la cantidad de dinero gastado por la parte cubana.	
montoEjecutadoVenezuela(DFicha_Proyecto ficha_Proyecto)	Método que permite conocer la cantidad de dinero gastado por la parte venezolana.	

Anexo 3: Pruebas de caja blanca aplicadas a las clases de las capas de presentación y lógica de negocio.

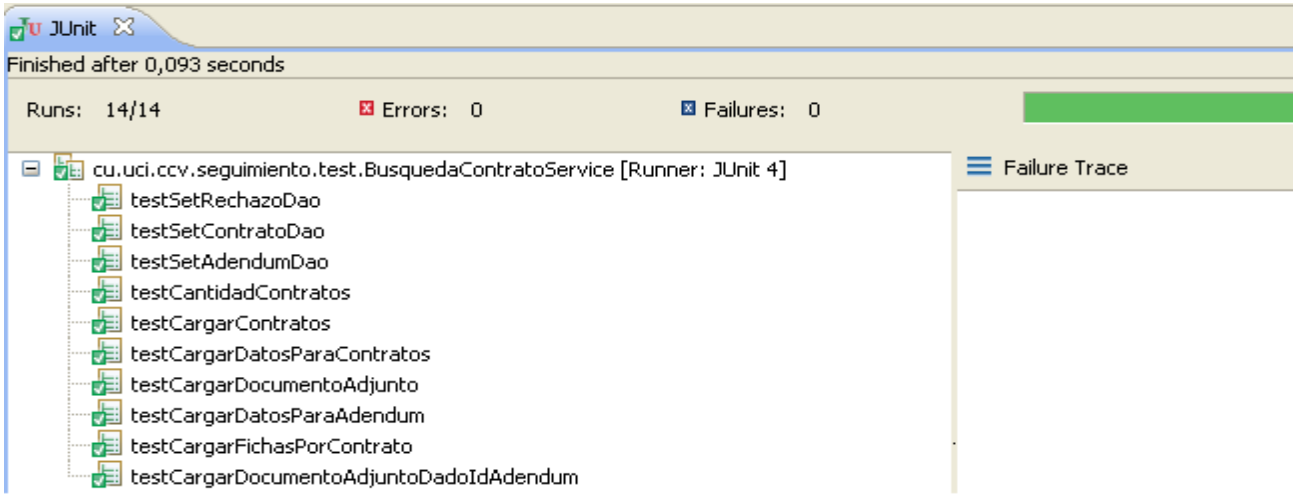

Figura1: Prueba realizada a la clase BusquedaContratoService.

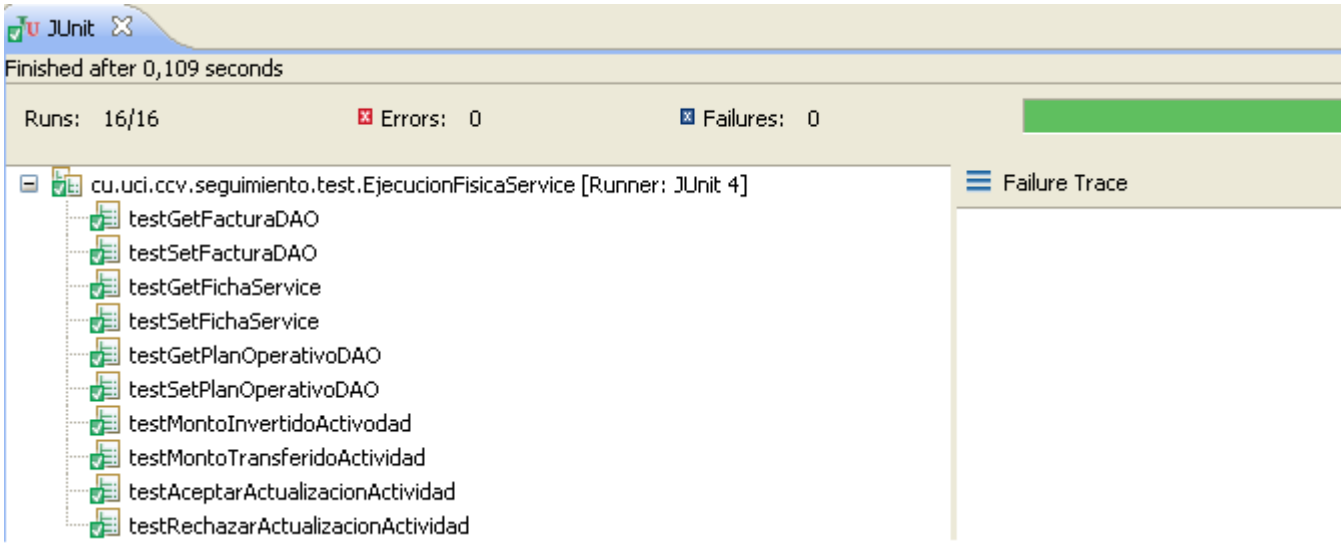

Figura 2: Prueba Realizada a la Clase EjecucionFisicaService

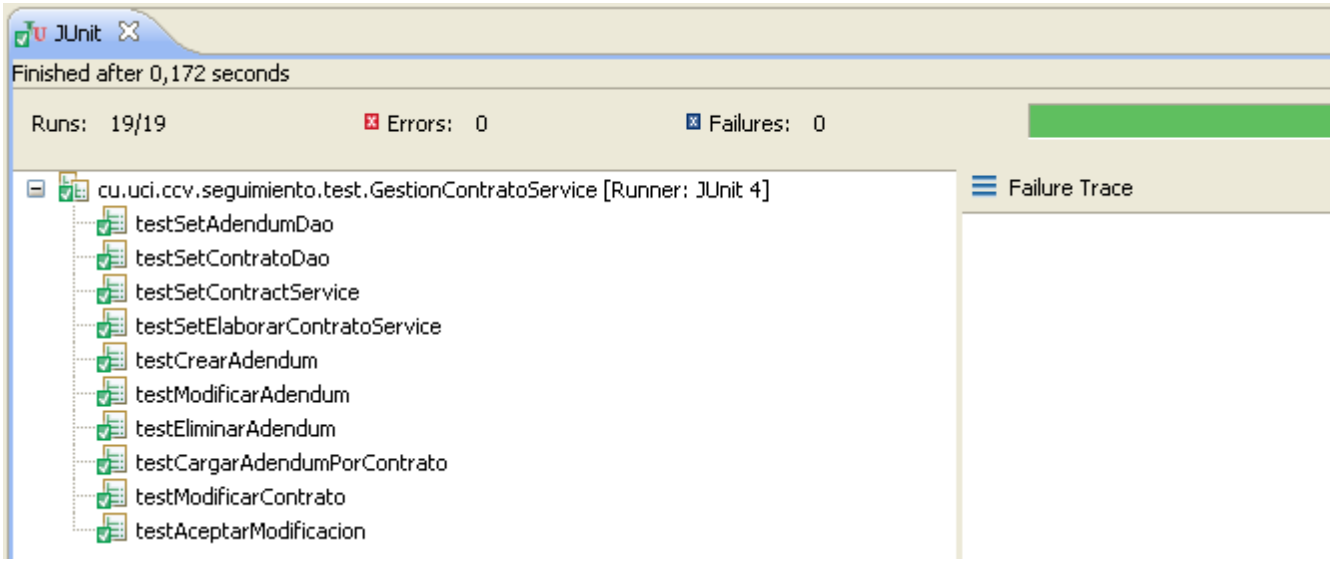

Figura 3: Prueba realizada a la clase GestionContratoService.

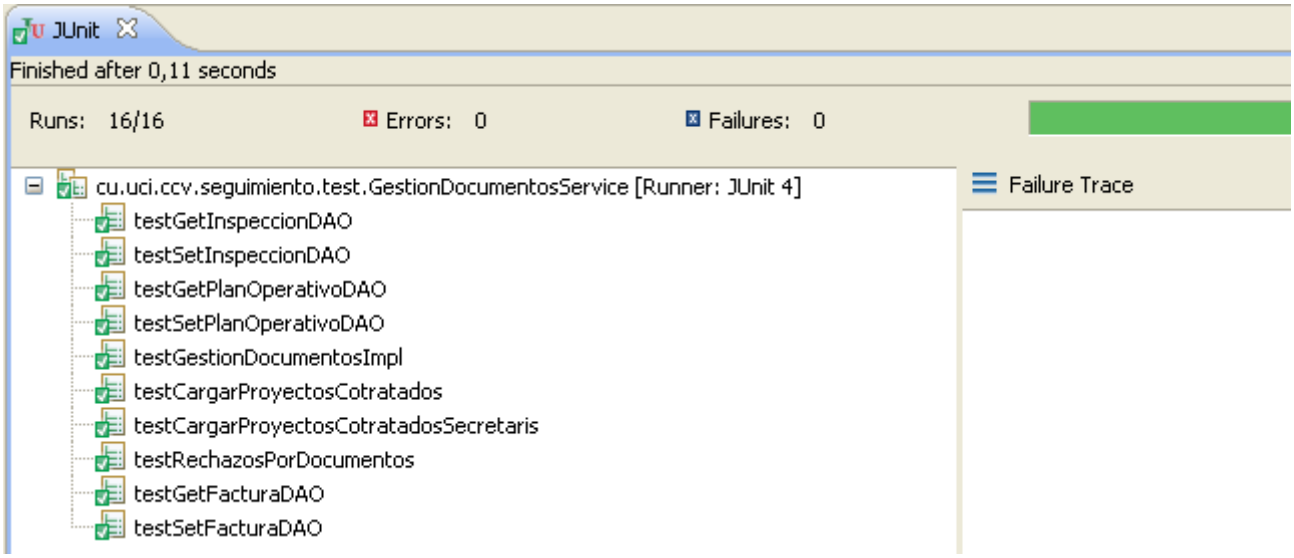

Figura 4: Prueba realizada a la clase GestionDocumentoService.

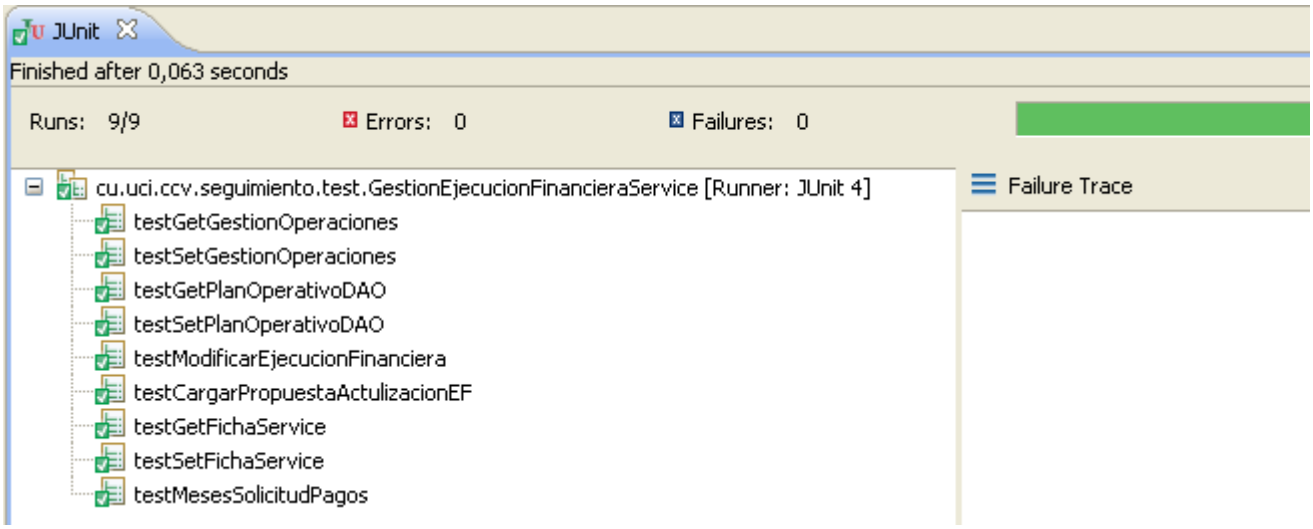

Figura 5: Prueba realizada a la clase GestionEjecucionFinancieraService.

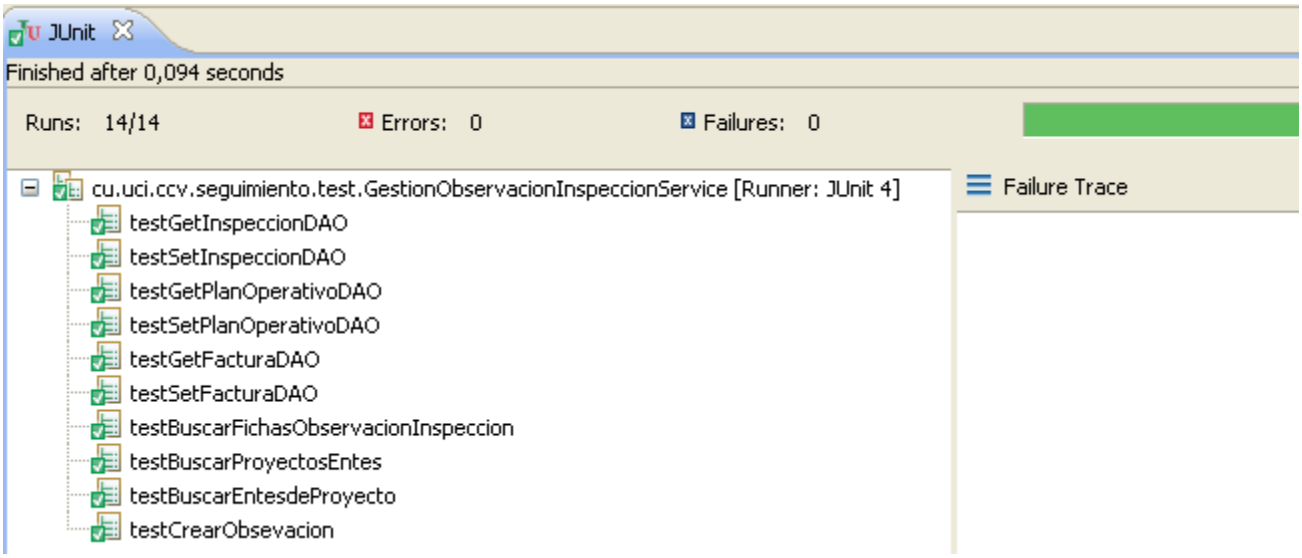

Figura 6: Prueba realizada a la clase GestionObservacionInspeccionService.

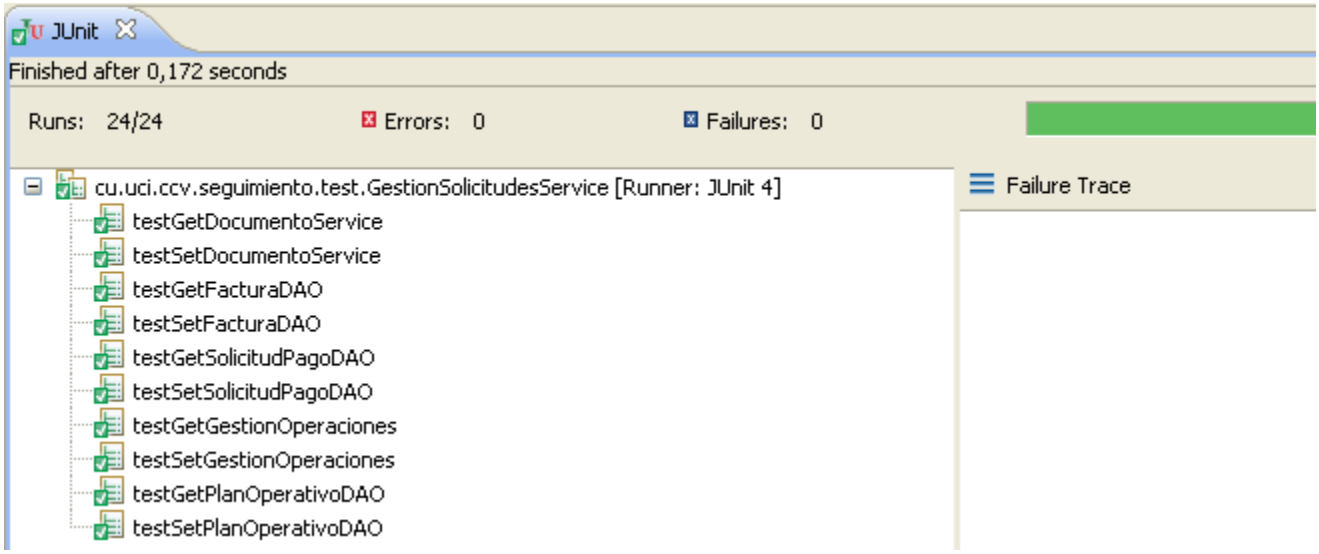

Figura 7: Prueba realizada a la clase de GestionSolicitudesService.

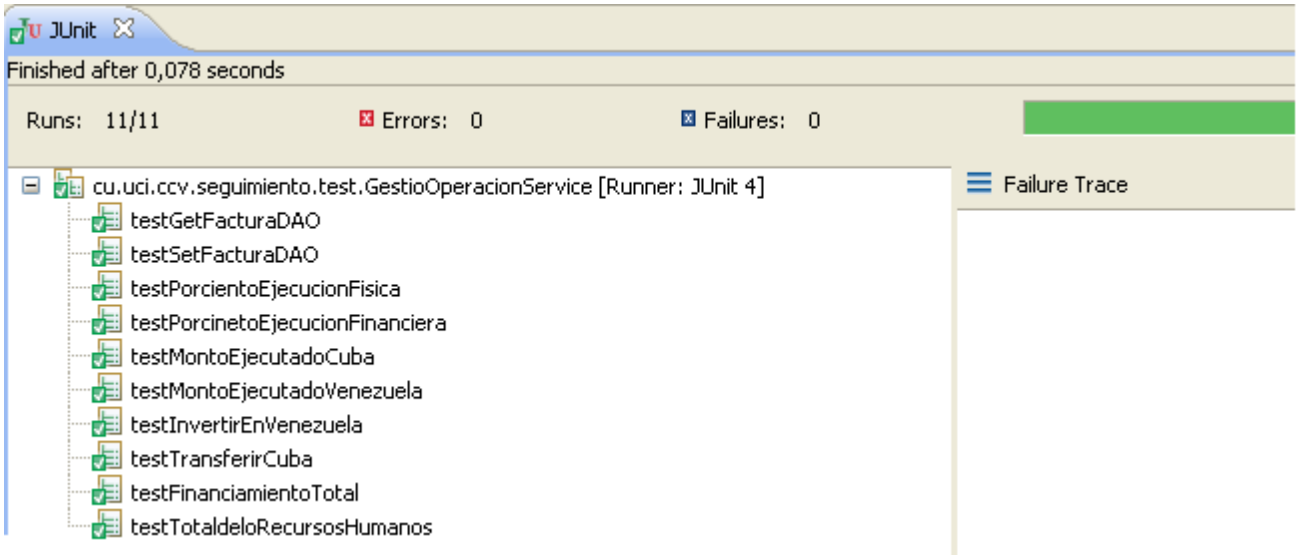

Figura 8: Prueba realizada a la clase GestionOperacionService.

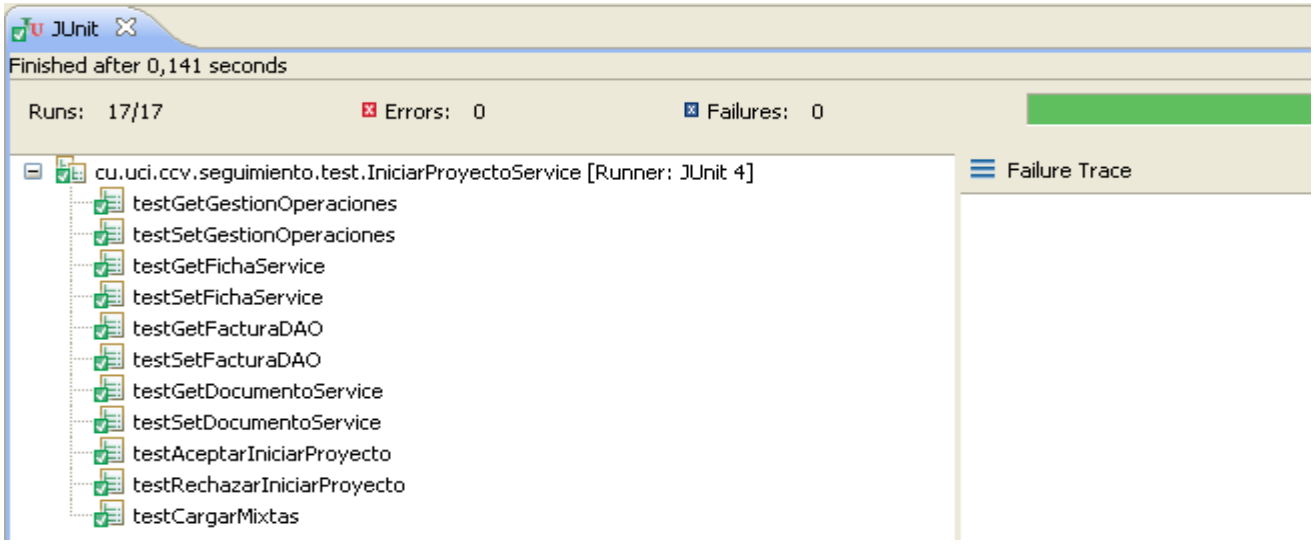

Figura 9: Prueba realizada a la clase IniciarProyectoService.

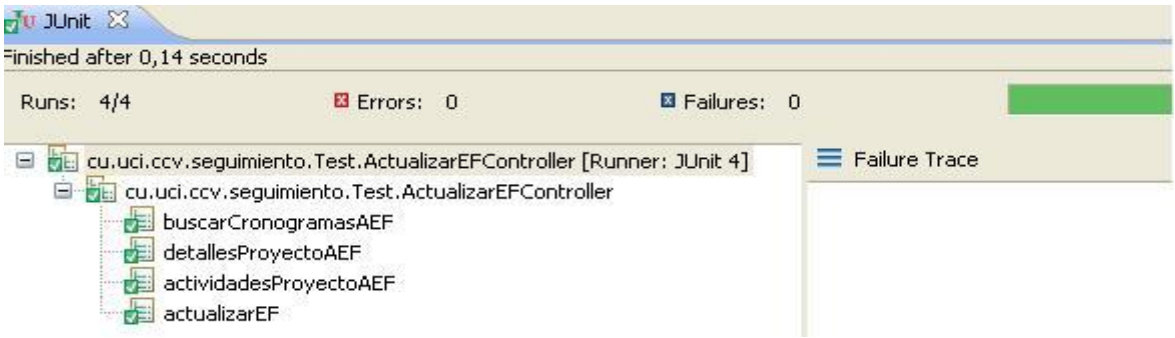

Figura 10: Prueba realizada a la clase ActualizarEFController.

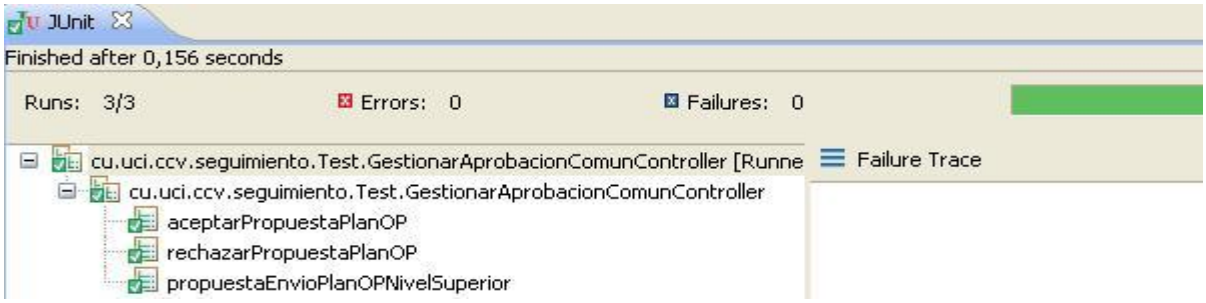

Figura 11: Prueba realizada a la clase GestionarAprobacionComunController.

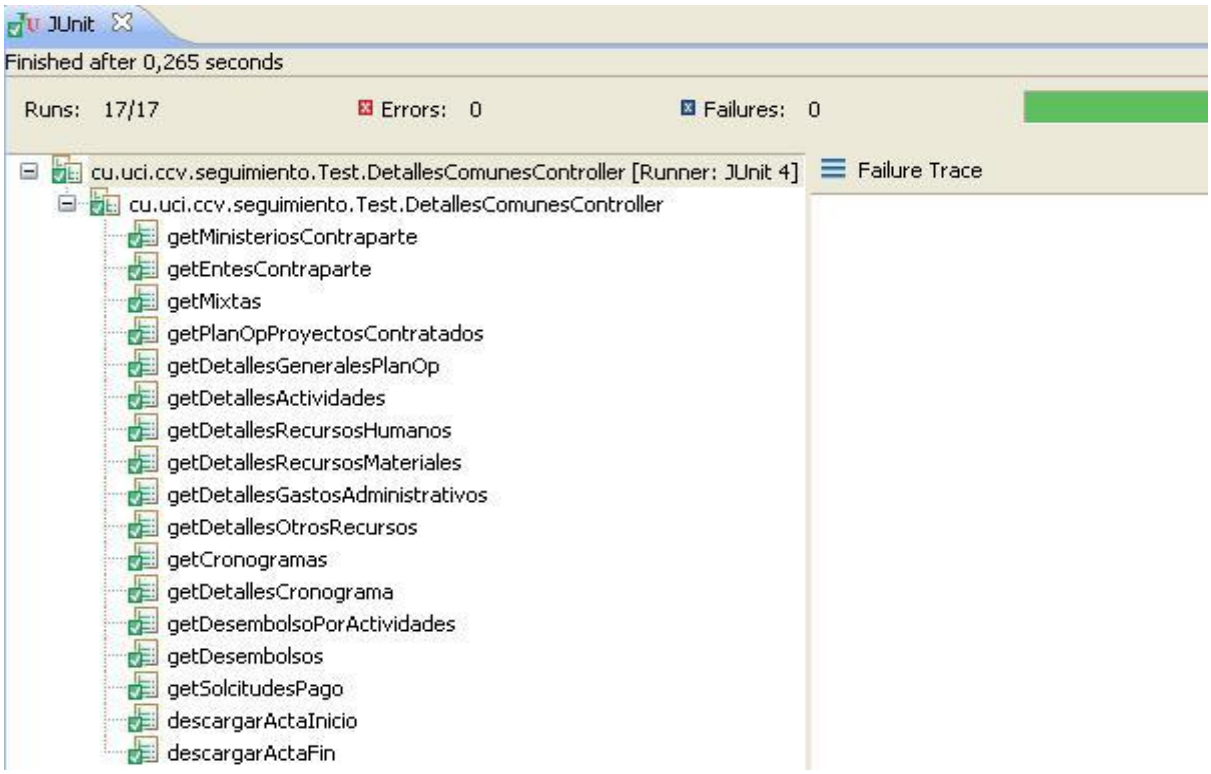

Figura 12: Prueba realizada a la clase DetallesComunesController.

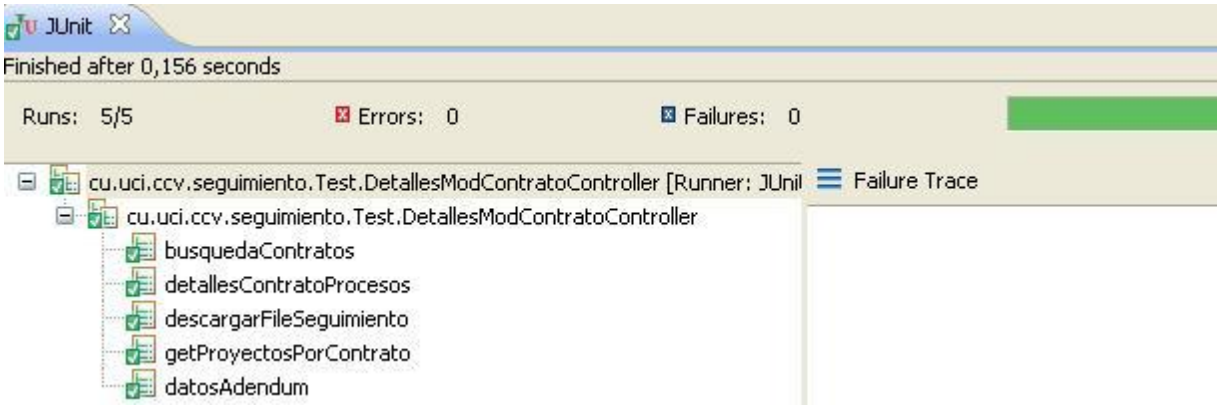

Figura 13: Prueba realizada a la clase DetallesModContratoController.

Figura 14: Prueba realizada a la clase GestionarModificacionContratoController.

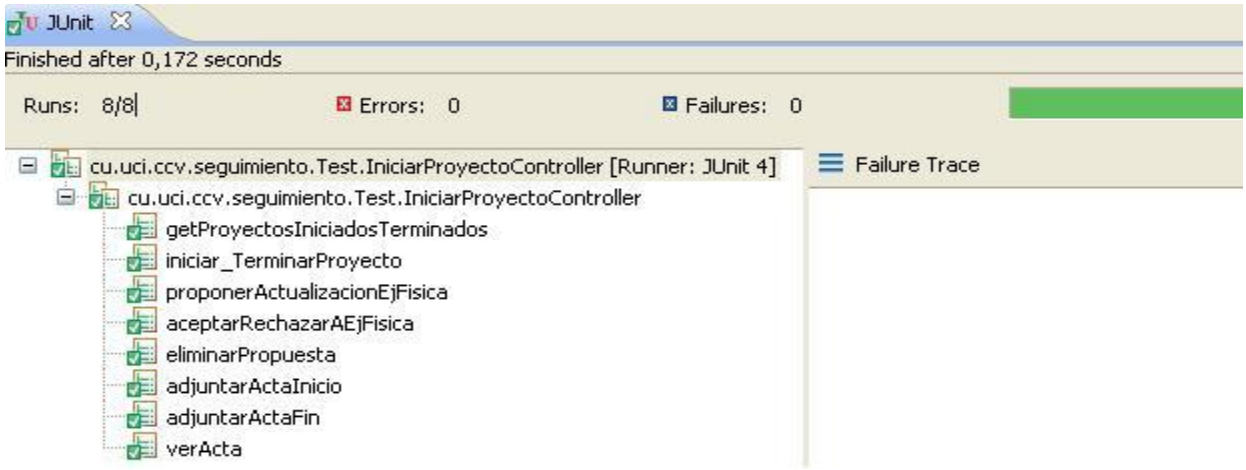

Figura 15: Prueba realizada a la clase IniciarProyectoController.

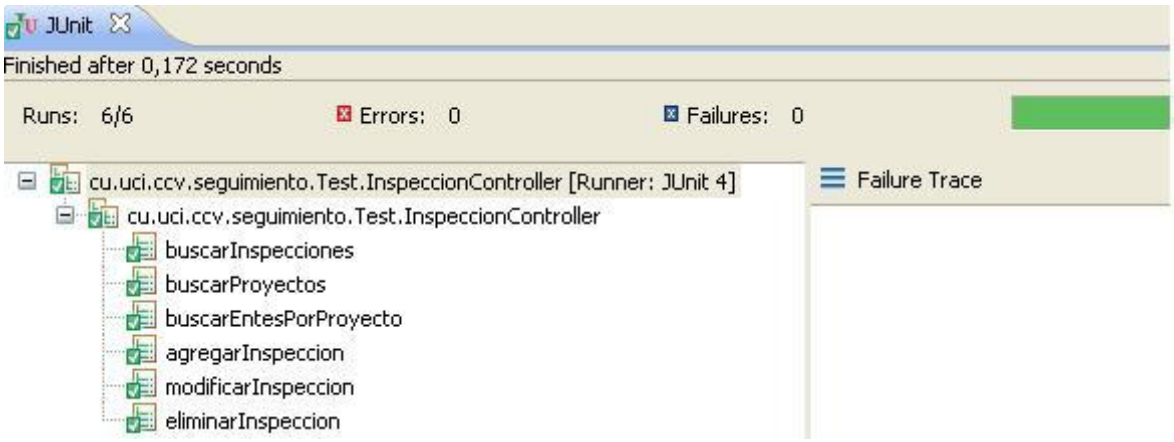

Figura 16: Prueba realizada a la clase InspeccionController.

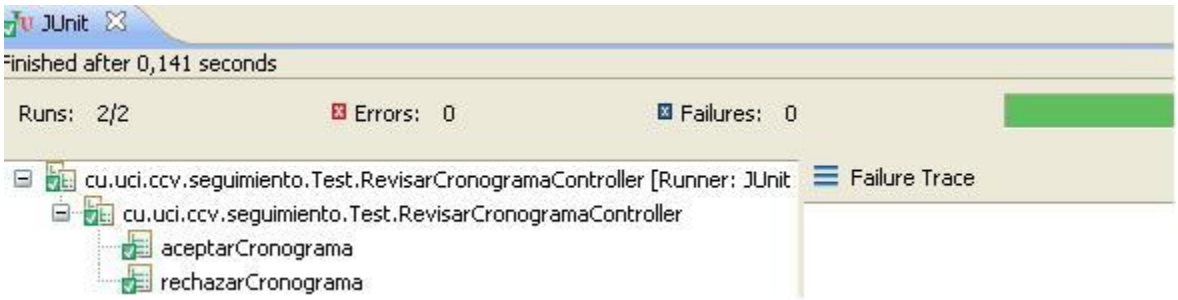

Figura 17: Prueba realizada a la clase RevisarCronogramaController.

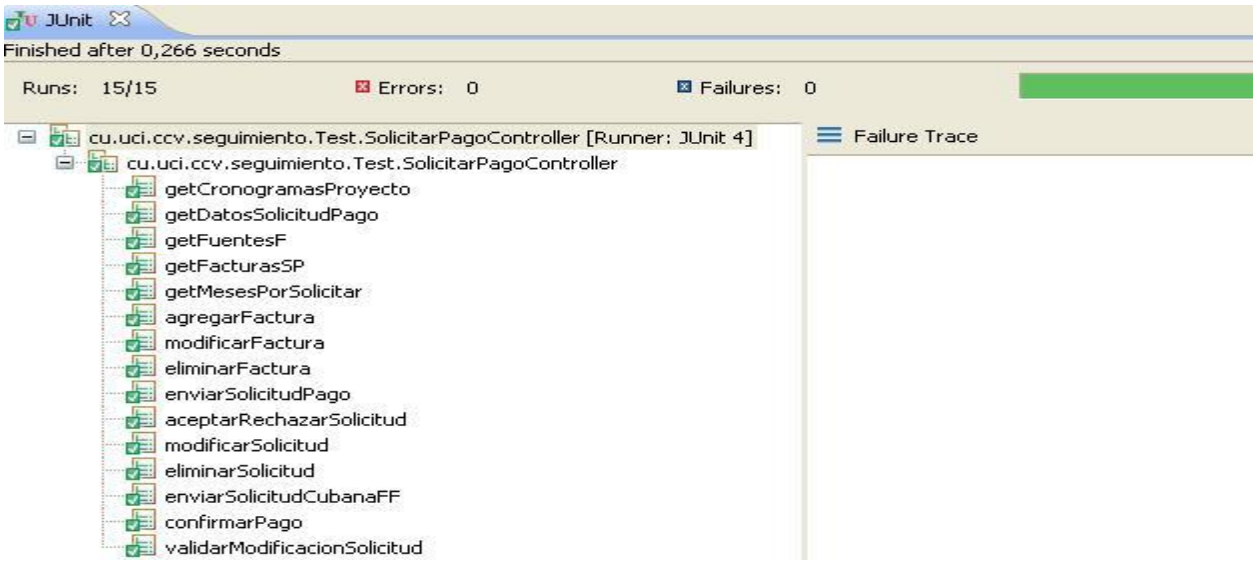

Figura 18: Prueba realizada a la clase SolicitarPagoController.

Anexo 4: Diseños de casos de prueba de caja negra realizadas al sistema.

Diseño del caso de prueba correspondiente al caso de uso Actualizar Ejecución Financiera.

Descripción General

El caso de uso inicia cuando un ente ejecutor necesita cambiar el estado de ejecución de una de las actividades del plan operativo, las mismas pueden estar en los siguientes estados: Sin ejecutar, En ejecución, Cerrada, Terminada.

Condiciones de Ejecución:

El sistema debe estar instalado y ejecutado correctamente.

El usuario debe estar autenticado con los permisos necesarios.

Secciones a probar en el Caso de Uso:

Nombre de la sección	Escenarios de la sección	Descripción de la funcionalidad	Flujo Central
SC 1: Cambiar estado a En ejecución	EC 1.1: El actor solicita la búsqueda de los proyectos	El sistema muestra los posibles estados en los que puede estar la actividad Sin ejecutar En ejecución El sistema modifica temporalmente el estado de la actividad de Sin ejecutar a En ejecución. El sistema cambia el estado del plan operativo a Actualización de plan operativo en espera de	Procesos/Actualizar Ejecución Física/ En ejecución

	EC 1.2: El actor cancela la operación.	El sistema no ejecuta ninguna acción, la actividad se queda en estado de Sin ejecutar. El caso de uso termina.	El actor debe cancelar.
SC 2: Cambiar estado a Cerrada	EC 2.1: Cambiar estado a Cerrada	El sistema muestra los posibles estados en los que puede estar la actividad En ejecución Cerrada Terminada El sistema modifica temporalmente el estado de la actividad de En Ejecución a Cerrada. El sistema muestra un mensaje de redistribución del monto de la actividad que no se ha consumido. El sistema cambia el estado del plan operativo a Actualización de plan operativo en espera de aceptación por ente. El caso de uso termina.	Procesos/Actualizar Ejecución Física/ En ejecución/ Cerrada
	EC 2.2: El actor cancela la operación.	El sistema no ejecuta ninguna acción, la actividad se queda en estado de En ejecución. El caso de uso termina.	El actor debe cancelar.

SC 3: Cambiar estado a Terminada	EC 3.1: Cambiar estado a Terminada	El sistema muestra los posibles estados en los que puede estar la actividad En ejecución Cerrada	Procesos/Actualizar Ejecución Física/ En ejecución/
	EC 3.2: El actor cancela la operación.	El sistema no ejecuta ninguna acción, la actividad se queda en estado de En ejecución. El caso de uso termina.	El actor debe cancelar.

SC 1: Cambiar estado a En Ejecución

Id del escenario	**Escenario**	**Sin ejecutar**	**En ejecución**	**Respuesta del Sistema**	**Resultado de la Prueba**
EC 1	El actor solicita la búsqueda de los proyectos	NA	NA	El sistema cambia el estado del plan operativo a Actualización de plan operativo en espera de aceptación por ente.	Se obtiene un resultado satisfactorio.
EC 2	El actor cancela la operación.	NA	NA	El sistema no ejecuta ninguna acción, la actividad se queda en estado de En ejecución. El caso de uso termina.	Se obtiene un resultado satisfactorio.

SC 2: Cambiar estado a Cerrada

Id del escenario	Escenario	En ejecución	Cerrada	Terminada	Respuesta del Sistema	Resultado de la Prueba
EC 1	Cambiar estado a Cerrada	I	V	I	El sistema muestra un mensaje de redistribución del monto de la actividad que no se ha consumido. El sistema cambia el estado del plan operativo a Actualización de plan operativo en espera de aceptación por ente.	Se obtiene un resultado satisfactorio.
EC 2	El actor cancela la operación.	NA	NA	NA	El sistema no ejecuta ninguna acción, la actividad se queda en estado de En ejecución. El caso de uso termina.	Se obtiene un resultado satisfactorio.

SC 3: < Cambiar estado a Terminada >

Id del escenario	Escenario	En ejecución	Cerrada	Terminada	Respuesta del Sistema	Resultado de la Prueba
EC 1	Cambiar estado a Terminada	I	I	V	El sistema modifica temporalmente el estado de la actividad de En Ejecución a Terminada. El sistema cambia el estado del plan operativo a Actualización de plan operativo en espera de aceptación por ente.	Se obtiene un resultado satisfactorio.
EC 2	El actor cancela la operación.	NA	NA	NA	El sistema no ejecuta ninguna acción, la actividad se queda en estado de En ejecución. El caso de uso termina.	Se obtiene un resultado satisfactorio.

Diseño del caso de prueba correspondiente al caso de uso Gestionar Inspección.

Descripción General

La secretaría técnica tiene la posibilidad de crear, mostrar, modificar o eliminar una inspección de la ejecución física de un proyecto.

Condiciones de Ejecución:

El sistema debe estar instalado y ejecutado correctamente. El usuario debe estar autenticado con los permisos necesarios. Existencia de inspecciones de la EF de un proyecto.

1. Secciones a probar en el Caso de Uso:

Nombre de la sección	Escenarios de la sección	Descripción de la funcionalidad	Flujo Central
SC 1: " Crear inspección"	EC 1.1: Nueva inspección.	El actor solicita crear una nueva inspección.	Seguimiento/ Documentos/ Inspecciones/ Nueva Inspección.
	EC 1.2: Registrar datos.	Registrar datos de Ministerio, Entes, Marco de aprobación, Nombre del proyecto, Fecha, Nombre del supervisor, Observaciones de la inspección.	Seguimiento/ Documentos/ Inspecciones/ Nueva Inspección.

	EC 1.3: Editar los datos.	El actor edita los datos de la inspección y acepta la operación.	Seguimiento/ Documentos/ Inspecciones/ Nueva Inspección.
	EC 1.4 Verificar y registrar.	Verificar los datos y registrar la información.	Seguimiento/ Documentos/ Inspecciones/ Nueva Inspección/Crear.
SC 2: “Mostrar inspección”	EC 2.1: Mostrar una inspección.	El actor solicita mostrar una inspección.	Seguimiento/ Documentos/ Inspecciones/ Seleccionar una inspección/ Ver detalles.
	EC 2.2: Datos de una inspección.	Se muestran los datos de una inspección: Ente ejecutor, Ministerio, Marco de aprobación del proyecto y Nombre del proyecto. Fecha de emisión de la inspección Nombre del supervisor Observaciones de la inspección Número de inspección	Seguimiento/ Documentos/ Inspecciones/ Seleccionar una inspección/ Ver detalles.

SC 3: “Modificar inspección”	EC 3.1: Seleccionar la inspección.	El actor selecciona la inspección que desea modificar.	Seguimiento/ Documentos/ Inspecciones/ Seleccionar una inspección/ Modificar.
	EC 3.2: Detalles de la inspección.	Se muestran los detalles de la inspección y permite hacer modificaciones en todos los datos de la inspección.	Seguimiento/ Documentos/ Inspecciones/ Seleccionar una inspección/ Ver detalles.
	EC 3.3: Modificar y aceptar.	El actor modifica los datos de la inspección y acepta la operación.	Seguimiento/ Documentos/ Inspecciones/ Seleccionar una inspección/ Modificar/Aceptar.

	EC 3.4: Verificar y modificar.	Se verifican los datos de la inspección y se modifica la información en la base de datos.	Seguimiento/ Documentos/ Inspecciones/ Seleccionar una inspección/Modificar/Aceptar.

SC 4: “Eliminar inspección”	EC 4.1: Eliminar una inspección.	El actor solicita eliminar una inspección de un determinado proyecto.	Seguimiento/ Documentos/ Inspecciones/ Seleccionar una inspección/ Eliminar.
	EC 4.2: Se muestra un mensaje.	Se muestra un mensaje de confirmación de eliminación de la inspección.	Seguimiento/ Documentos/ Inspecciones/ Seleccionar una inspección/ Eliminar.
	EC 4.3: Aceptar la operación	El actor acepta la operación de eliminación.	Seguimiento/ Documentos/ Inspecciones/ Seleccionar una inspección/ Eliminar/Si.

	EC 4.4: Eliminar	El sistema elimina la inspección.	Seguimiento/ Documentos/ Inspecciones/ Seleccionar una inspección/ Eliminar/Si.

1.1 SC 1: “Crear inspección”

Id del escenario	Escenario	Ministerio	Entes	Marco de aprobación	Nombre del proyecto	Fecha	Nombre del supervisor	Observaciones de la inspección	Respuesta del Sistema	Resultado de la Prueba
EC 1.1	Nueva inspección.	NA	NA	NA	NA	NA	NA	NA	El sistema da la opción de crear una nueva inspección	La prueba resultó satisfactoria.
EC 1.2	Registrar datos.	NA	NA	NA	NA	NA	NA	NA	El sistema muestra una interfaz para registrar.	La prueba resultó satisfactoria

EC 1.3	Editar los datos.	I	V	V	V	V	V	V	El sistema muestra un mensaje informativo.	La prueba resultó satisfactoria.
EC 1.4	Verificar y registrar.	NA	NA	NA	NA	NA	NA	NA	Se escribe el resultado que se espera al realizar la prueba.	La prueba resultó satisfactoria.

1.2 SC 2: “Mostrar inspección”

Id del escenario	**Escenario**	**Ente ejecutor**	**Ministerio**	**Marco de aprobación del proyecto**	**Nombre del proyecto**	**Fecha de emisión de la inspección**	**Nombre del supervisor**	**Observaciones de la inspección**	**Número de inspección**	**Respuesta del Sistema**	**Resultado de la Prueba**
EC 2.1	Mostrar una inspección.	NA	NA	NA	NA	NA		NA	NA	El sistema muestra las inspecciones	La prueba resultó satisfactoria.

EC 2.2	Datos de una inspección.	NA	NA	NA	NA	NA		NA	NA	El sistema muestra una interfaz con los datos de la inspección.	La prueba resultó satisfactoria.

1.3 SC 3: “Modificar inspección”

Id del escenario	**Escenario**	**Ministerio**	**Entes**	**Marco de aprobación**	**Nombre del proyecto**	**Fecha**	**Nombre del supervisor**	**Observaciones de la inspección**	**Respuesta del Sistema**	**Resultado de la Prueba**
EC 3.1	Seleccionar la inspección.	NA	NA	NA	NA	NA	NA	NA	El sistema muestra la pantalla de listado de proyectos.	La prueba resultó satisfactoria.

EC 3.2	Detalles de la inspección.	NA	NA	NA	NA	NA	NA	NA	El sistema permite hacer modificaciones en todos los datos de la inspección.	La prueba resultó satisfactoria.
EC 3.3	Modificar y aceptar.	I	V	V	V	V	V	V	El sistema te permita aceptar los cambios.	La prueba resultó satisfactoria.
EC 3.4	Verificar y modificar.	NA	NA	NA	NA	NA	NA	NA	El sistema modifica la base de datos.	La prueba resultó satisfactoria.

1.4 SC 4: “Eliminar inspección”

Id del escenario	**Escenario**	**Ministerio**	**Entes**	**Marco de aprobación**	**Nombre del proyecto**	**Fecha**	**Nombre del supervisor**	**Observaciones de la inspección**	**Respuesta del Sistema**	**Resultado de la Prueba**

EC 4.1	Eliminar una inspección.	NA	NA	NA	NA	NA	NA	NA	El sistema da la opción de eliminar una inspección.	La prueba resultó satisfactoria.
EC 4.2	Se muestra un mensaje.	NA	NA	NA	NA	NA	NA	NA	El sistema muestra mensaje para verificar que este seguro de querer eliminar.	La prueba resultó satisfactoria.

EC 4.3	Aceptar la operación	NA	NA	NA	NA	NA	NA	NA	El sistema acepta que sea eliminada la inspección.	La prueba resultó satisfactoria.
EC 4.4	Eliminar	NA	NA	NA	NA	NA	NA	NA	El sistema elimina la inspección.	La prueba resultó satisfactoria.

Diseño del caso de prueba correspondiente al caso de uso Revisar Re-planificación del Cronograma de Ejecución Financiera.

Descripción General

El caso de uso inicia cuando una de las Secretarías Técnicas necesita revisar la propuesta de re-planificación del cronograma de ejecución financiera de un proyecto determinado. El mismo acepta o rechaza la modificación.

Condiciones de Ejecución:

El sistema debe estar instalado y ejecutado correctamente. El usuario debe estar autenticado con los permisos necesarios.
Existencia de cronogramas de ejecución financiera en estado de: Envío de cronograma de ejecución financiera aceptado por los Ministerios y/o Cronograma de ejecución financiera aceptado por una Secretaría Técnica.

Secciones a probar en el Caso de Uso:

El caso de uso inicia cuando una de las Secretarías Técnicas necesita revisar la propuesta de re-planificación del cronograma de ejecución financiera de un proyecto determinado. El mismo acepta o rechaza la modificación.

Nombre de la sección	Escenarios de la sección	Descripción de la funcionalidad	Flujo Central
SC 1: Aceptar re-planificación del cronograma de ejecución financiera	EC 1.1: El actor acepta la opción de re-planificación del cronograma de ejecución financiera	El sistema cambia el estado del cronograma a Cronograma de ejecución financiera aceptado por una	Seguimiento/ Procesos/ Revisar Cronograma de Ejecución Financiera/ Buscar/ Seleccionar Cronograma en espera de aprobación de secretarias técnicas / Ver detalles/ Aceptar.
SC 2: Rechazar modificación del cronograma de ejecución financiera	EC 2.1: El actor rechaza la opción de re-planificación del cronograma de ejecución financiera.	El sistema cambia el estado del cronograma a Cronograma de ejecución financiera rechazado por una ST. El caso de uso termina.	Seguimiento/ Procesos/ Revisar Cronograma de Ejecución Financiera/ Buscar/ Seleccionar Cronograma en espera de aprobación de secretarias técnicas / Ver detalles/ Rechazar.

SC 1: Aceptar re-planificación del cronograma de ejecución financiera.

Id del escenario	Escenario	Respuesta del Sistema	Resultado de la Prueba
EC 1.1	El actor acepta la opción de re-planificación del cronograma de ejecución financiera	El sistema cambia el estado del cronograma a Cronograma de ejecución financiera aceptado por una ST. El caso de uso termina.	Se obtiene un resultado satisfactorio.

SC 2: Rechazar modificación del cronograma de ejecución financiera.

Id del escenario	Escenario	Respuesta del Sistema	Resultado de la Prueba
EC 2.1	El actor rechaza la opción de re-planificación del cronograma de ejecución financiera.	El sistema cambia el estado del cronograma a Cronograma de ejecución financiera rechazado por una ST. El caso de uso termina.	Se obtiene un resultado satisfactorio.

Registro de defectos y dificultades detectados

Elemento	No	No Conf.	Aspecto	Etapa de detección	Significativa	No Significativa	Recomendación	Estado NC	Respuesta del Equipo Desarrollo
Aplicación	1	Ocurrió un error en la aceptación del cronograma ver Figura 3.8	CU: Revisar cronograma (ST)	Primera iteración	x			PD	NP

Anexo 5: 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 N° 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 **Ibrahim Nápoles Albanés**, mayor de edad, portador de carné de identidad N° 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**.

Cuarto: Consta que la **Parte Venezolana** ha pagado la totalidad de **UN MILLÓN CUATROCIENTOS DIECISIETE MIL OCHOCIENTOS VEINTIOCHO DOLARES DE LOS ESTADOS UNIDOS DE AMERICA CON VEINTIUN CENTAVOS** mediante la forma de pago prevista en el Contrato ya mencionado, a entera satisfacción de la Parte Cubana, quedando a la firma de la presente **Acta**.

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

Ammar Jabour
Coordinador General

Por la PARTE CUBANA

Ibrahim Nápoles Albanés
Coordinador General

GLOSARIO DE TÉRMINOS

API: (Application Programming Interface - Interfaz de Programación de Aplicaciones) es el conjunto de funciones y procedimientos (o métodos si se refiere a programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

EJB: Enterprise JavaBeans, una de las API que forman parte del estándar de construcción de aplicaciones empresariales J2EE.

Inversión de Control: También conocido como inyección de dependencia, se refiere a la forma en que un objeto usa otro objeto.

JDBC: API de la plataforma Java para el acceso a sistemas gestores de base de datos.

JNDI: Interfaz de Nombrado y Directorio Java, es una Interfaz de Programación de Aplicaciones (API) para servicios de directorio.

JNI: Es un framework de programación que permite que un programa escrito en Java ejecutado en la máquina virtual de java (JVM) pueda interactuar con programas escritos en otros lenguajes.

JSF: Es un framework para aplicaciones Java basadas en web que simplifica el desarrollo de interfaces de usuario en aplicaciones J2EE.

JSP: API de la plataforma J2EE para la creación de páginas web dinámicas mediante el uso de librerías de tags.

SOAP: Protocolo estándar que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML

WSDL: Describe la interfaz pública a los servicios Web. Está basado en XML y describe la forma de comunicación, es decir, los requisitos del protocolo y los formatos de los mensajes necesarios para interactuar con los servicios listados en su catálogo.