ESCUELA POLITÉCNICA DEL EJÉRCITO

DPTO. DE CIENCIAS DE LA COMPUTACIÓN

CARRERA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA

DESARROLLO DIRIGIDO POR TEST (TDD) UTILIZANDO EL FRAMEWORK JUNIT EN UN SISTEMA WEB DE ASIGNACIÓN DE AULAS DE LOS LABORATORIOS GENERALES DE COMPUTACIÓN DE LA ESPE, APLICANDO LA METODOLOGÍA AGILE UNIFIED PROCESS (AUP)

Previa a la obtención del Título de:

INGENIERO DE SISTEMAS E INFORMÁTICA

POR: CARLOS JAVIER AUCANCELA MAGUANA

TATIANA CAROLINA POZO MOLINA

SANGOLQUI, 31 de Octubre del 2011

CERTIFICACIÓN

Certifico que el presente trabajo fue realizado en su totalidad por el Sr. Carlos Javier Aucancela Maguana y la Srta. Tatiana Carolina Pozo Molina como requerimiento parcial a la obtención del título de Ingenieros en Sistemas e Informática.

31 de Octubre del 2011.

_____

Ing. Cecilia Hinojosa

DEDICATORIA

El presente trabajo está dedicado a mis padres Carlos y Cecilia ya que con su apoyo, amor y paciencia han sido quienes me inspiran a cumplir mis metas planteadas y he aquí un fruto de su perseverancia.

A mis hermanos Edwin, Cecibel, David quienes me brindan felicidad el día a día y me siento orgulloso al verlos crecer.

A mis familiares Miguel, Anita, Tania, Marco, Manuel que han estado a mi lado brindándome su ejemplo y enseñándome lo valioso de la vida.

A mi enamorada Tatiana gracias por compartir este proyecto, me enorgullece estar a tu lado y poder confiar en ti.

CARLOS JAVIER AUCANCELA MAGUANA

AGRADECIMIENTO

Agradezco a Dios por brindarme la oportunidad de vivir cada día y enseñarme a seguir adelante frente a las dificultades que se presentan en la vida.

Agradezco a mis padres Cecilia y Carlos quienes me han demostrado que a pesar de adversidades, el amor de familia siempre se sobrepondrá ante las duras decisiones, este paso de mi vida se los debo a ustedes.

Agradezco a mis queridos hermanos quienes me han dado fuerza día a día para poder cumplir una meta más y demostrarles que en la vida nada es imposible, debemos luchar por nuestros sueños levantándonos a pesar de los tropiezos.

A mi tía Anita gracias por compartir tus consejos en momentos que más lo necesitaba, gracias por ser mi segunda madre.

Agradezco a mis tíos Miguel y Tania quienes me brindan su ejemplo y me enseñan los caminos de Dios.

Agradezco desde el fondo de mi corazón a Tatiana Pozo quien ha sido un pilar de apoyo personal y sentimental, gracias por entenderme y brindarme tu amor.

A mis verdaderos amigos que estuvieron apoyándome en todo sentido.

CARLOS JAVIER AUCANCELA MAGUANA

DEDICATORIA

Este trabajo va dedicado a mis padres Rocio Molina, Guillermo Pozo y a mi hermana Andrea Pozo ya que gracias a su esfuerzo, su amor y cariño me ayudaron a nunca rendirme y mantenerme firme durante toda mi vida académica.

A mi novio Carlos Aucancela que es un pilar importante para la culminación de mi carrera, ha estado en los momentos difíciles tanto de mi vida personal como académica, brindándome su apoyo para seguir adelante y no darme por vencida.

También dedico este trabajo a una persona muy especial en mi vida, mi sobrino Justyn Ariel quien con su simple sonrisa y sus travesuras me inspira a seguir adelante.

TATIANA CAROLINA POZO MOLINA

AGRADECIMIENTO

En primer lugar agradezco a Dios por permitirme cada día despertar y vivir nuevas experiencias, porque siempre está conmigo y me alienta a seguir adelante en esos momentos difíciles.

Gracias a mis padres Rocio Molina y Guillermo Pozo que siempre me apoyaron moralmente, económicamente, sobretodo me brindaron su amor y comprensión, gracias porque siempre confiaron y creyeron en mí, por inculcarme esos valores que me hacen ser la persona que soy ahora.

A mi hermana y abuelita porque siempre estuvieron pendientes de mí y de una u otra manera me supieron apoyar frente a los problemas que se presentaban durante mi vida, las quiero mucho.

A mi compañero de tesis Carlos Aucancela quien depositó en mí toda su confianza para poder realizar este trabajo, nunca dejó que me rindiera ni en esos momentos en los que ya quieres dejar todo, sus palabras me motivaron a luchar y seguir adelante.

A la ingeniera Cecilia Hinojosa y al ingeniero Aly Abdelrahaman quienes formaron parte de este proceso de trabajo ya que con sus conocimientos supieron guiarnos y poder realizar un buen trabajo.

Como olvidar a mis amigos que con ellos compartí momentos alegres y tristes, aventuras inolvidables tanto académicas como del diario vivir, siempre las llevaré en mi mente; gracias por su amistad, apoyo y comprensión.

A todos mis profesores que con sus palabras me enseñaron a luchar por mis objetivos y a nunca rendirme, principalmente gracias por brindarme su amistad y su conocimiento durante toda mi vida académica.

Una nueva etapa de mi vida acaba y fue una de las mejores experiencias que he vivido, nuevamente gracias a todos por ser parte de este proceso los llevaré siempre en mi corazón.

TATIANA CAROLINA POZO MOLINA

ÍNDICE DE CONTENIDO

LISTADO DE TABLAS

LISTADO DE FIGURAS

RESUMEN

El presente trabajo está orientado al estudio del desarrollo guiado por pruebas, o TDD una técnica de desarrollo que involucra dos prácticas: primero escribir las pruebas obteniendo la solución a especificaciones de requisitos y la refactorización de código con la finalidad de obtener calidad en el desarrollo.

El procedimiento de esta técnica empieza por escribir una prueba acorde a un requerimiento específico, en la cual se verifica el fallo inicial de dichas pruebas, luego se implementa el mínimo código que haga que la prueba pase satisfactoriamente cumpliendo el requerimiento solicitado para luego proceder a refactorizar el código escrito.

La aplicación práctica de TDD fue realizada en el desarrollo de un sistema web para reservar los laboratorios computacionales de la ESPE (SILVERLAB), siguiendo los lineamientos de la metodología de desarrollo de software AUP, la cual se basa en cortas iteraciones, desde el levantamiento de requisitos, análisis, diseño e implementación.

Para el proceso de pruebas se utilizó la herramienta JUNIT[1] con la finalidad de verificar, manejar y ejecutar conjuntos de pruebas automatizadas.

El producto software resultante es un sistema web distribuido, desarrollado con software libre en el lenguaje de programación JAVA, basado en el patrón de

[1]JUNIT.- conjunto de bibliotecas creadas por Erich Gamma y Kent Beck que son utilizadas para hacer pruebas unitarias de aplicaciones Java.

diseño Modelo Vista Controlador que permite separar en componentes dicho sistema, además posee un gestor de base de datos MYSQL.

Tras el desarrollo de la aplicación se evidenció las ventajas que provee TDD tales como:

- ✓ Permitir identificar lo que es realmente imprescindible implementar por lo que se ahorrará tiempo desarrollando código que luego no se usará.
- ✓ Fuerza a un estricto análisis y diseño, ya que el desarrollador no puede crear código de producción sin entender realmente cuales deberían ser los resultado deseados y como probarlos.
- ✓ El conjunto de test unitarios proporciona constante retroalimentación de que cada uno de los componentes sigue funcionando.

El aporte de este estudio ofrece más que una simple validación del cumplimiento de requisitos, ya que sirvió como guía al diseño de la aplicación práctica haciendo que el código que funciona retroalimente las nuevas decisiones de diseño.

CAPÍTULO I

INTRODUCCIÓN

Históricamente, las metodologías tradicionales han intentado abordar la mayor cantidad de situaciones de contexto del proyecto, exigiendo un esfuerzo considerable para ser adaptadas utilizando gran cantidad de artefactos, roles, entregables, etc. Ya que en las metodologías tradicionales se establece énfasis en la arquitectura del software. Por esa razón se puede establecer que las metodologías tradicionales como por ejemplo RUP han demostrado ser efectivas y necesarias en cierto tipo de proyectos, pero también han presentado problemas en otros, siendo demostradas en las estadísticas realizadas por la empresa Standish Group ***(Standish Group, 2000)*** donde se puede observar el alto porcentaje de la problemática en proyectos de software.

La evolución en el desarrollo del software ha tomado nuevas prácticas en los proyectos actuales, donde el entorno del sistema es muy cambiante, exige reducir drásticamente los tiempos de desarrollo, disminuir costos, y mantener una alta calidad.

De esta manera emergen las Metodologías Ágiles como una posible respuesta a los problemas en el desarrollo de software, modificando el proceso con la finalidad de reaccionar ante los cambios centrándose en el trabajo en equipo y responsabilidad propia.

La idea de Metodologías Ágiles nació a principios de la década de los 90 originalmente con el nombre Rapid Application Development. Durante 1996, Kent Beck después del éxito al implantar un sistema basado en prácticas innovadoras que él había ido definiendo a lo largo del tiempo; da origen a la programación extrema (XP) con un enfoque bastante revolucionario en su momento, fomentando prácticas como potenciar las relaciones interpersonales (colaboración con el cliente), desarrollo incremental del software con iteraciones muy cortas, diseño simple y realización de pruebas.

Una vez consolidadas las Metodologías Ágiles surge Agile Unifed Process (AUP) que es una manera simple y fácil de entender la forma de desarrollar aplicaciones de software usando técnicas ágiles como el Desarrollo Dirigido por Pruebas (Test Driven Development - TDD), Modelado Ágil, Gestión de Cambios Ágiles, y Refactorización de Base de Datos para mejorar la productividad.

Con el pasar del tiempo estas Metodologías Ágiles han definido ideas innovadoras plasmadas en una serie de prácticas que están cambiando la forma habitual en la que se desarrolla el software. Gracias a una buena estrategia de pruebas se puede incrementar la productividad a la vez que se mejora el control sobre los programas que se construyen. Una de las cualidades más destacables en una metodología ágil es su sencillez, tanto en su aprendizaje como en su aplicación, reduciéndose así los costos de implantación en un equipo de desarrollo.

Las prácticas más importantes de las metodologías ágiles han dado lugar a una nueva técnica de diseño denominada Desarrollo Guiado por Pruebas, o "*Test Driven Development"* (*TDD)* la cual trata de "convertir al programador en un verdadero desarrollador" ***(Blé, Enero 2010).***

El TDD intenta traducir el caso de uso en x números de ejemplos hasta que el número de ejemplos sea suficiente como para describir una tarea específica.

1.1.- PLANTEAMIENTO DEL PROBLEMA

El proceso del desarrollo de software de calidad se ve opacado cuando no existe el suficiente diálogo o un inadecuado entendimiento de las necesidades de los usuarios ya que, la fase de pruebas la realizan después de tener código implementado y es común que la prueba implique grandes esfuerzos al momento de realizar cambios en los requisitos.

Las insatisfacciones de los clientes por inaceptable o bajo desempeño del software también se debe a las validaciones tardías de requisitos y a la falta de un previo proceso de pruebas, lo que origina el fracaso de los proyectos o la restructuración por fallas, y si se realizan pruebas no hay un suficiente conocimiento de los diferentes tipos de test, entre los cuales sobresalen:

- Test de Aceptación.
- Test Funcionales.
- Test de Sistema.
- Test Unitarios
- Test de Integración.

Standish Group realizó un estudio con la finalidad de obtener información de los proyectos fallidos en tecnologías de la información (TI) y así poder encontrar y combatir las causas de los fracasos.

El estudio realizado obtuvo los siguientes resultados:

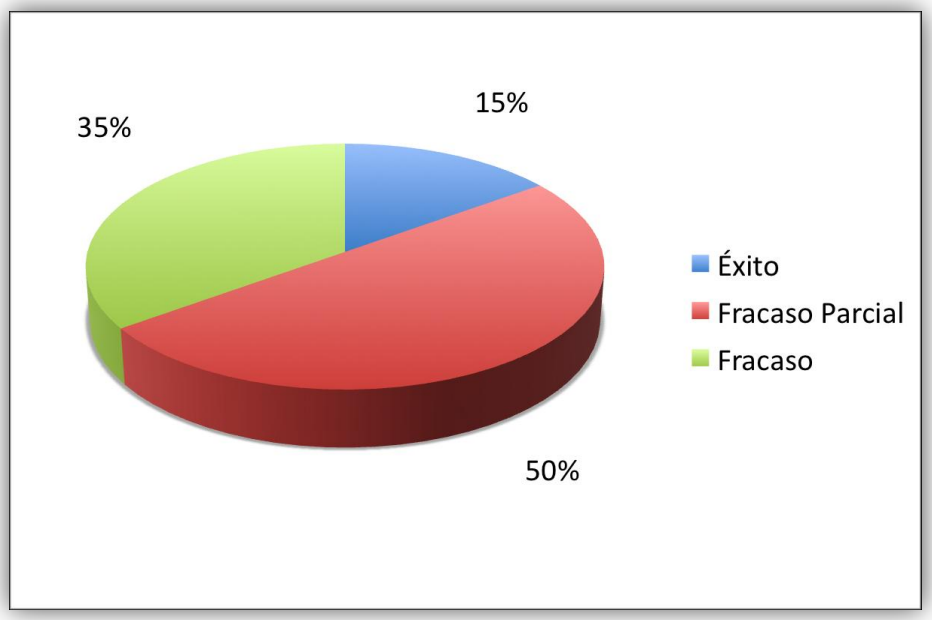


Figura 1.1: Estadística de Proyectos de Software *(Barros, 2010)*

Otras cifras con la que se puede comparar son las siguientes:

"- Chaos Report (1994) - 16.2% de los proyectos son exitosos. Del 70% de proyectos no exitosos, 52% corresponden a fallas parciales (expectativas, tiempos o dinero).

- Encuesta OASIG (1995) - Las respuestas más optimistas sitúan la tasa de éxito entre 20% y 30%.

- Encuesta KPMG Canada (1997) - 61% de los proyectos se consideran fracasados.

- Encuesta Conference Board (2001) - 40% de los proyectos no cumplen expectativas de negocios, luego de un año en operación

- Encuesta Robbins-Gioia (2001) - 51% de las implementaciones de ERP son fallidas

- Encuesta Revista Dr. Dobb's (2007) - 72% de los proyecto de desarrollo e implementación de Data Warehouse con metodologías ágiles, versus 63% con metodologías tradicionales." ***(Barros, 2010)***

Todos los estudios coinciden con las mismas causas de fracaso de los proyectos, las que se pueden resumir en:

- ✓ Especificaciones y requerimientos cambiantes o incompletos.
- ✓ Falta de involucramiento de usuarios.
- ✓ Pocos conocimientos técnicos del equipo de proyecto.
- ✓ Uso inadecuado de métodos y herramientas.
- ✓ Expectativas poco realistas falta de soporte gerencial.
- ✓ Gestión de proyectos débiles, lo que incluye la falta de identificación de riesgos, falta de planificación y comunicación deficiente.

1.2.- JUSTIFICACIÓN E IMPORTANCIA

En la actualidad el desarrollo de software se ha convertido en un proceso que no ha sido bien definido, por lo que la presente investigación pretende realizar un estudio sobre los beneficios que otorga el Desarrollo Dirigido por Test aplicado en las metodologías ágiles, con la finalidad de evitar correcciones en la fase de desarrollo disminuyendo errores y evitando la pérdida de costos y tiempo.

El desarrollo guiado por pruebas, es en primer plano, un método de diseño de software, y no solamente un método de pruebas, además se aplica una de las principales cualidades de una metodología ágil como es la sencillez, tanto en su aprendizaje como en su aplicación, reduciéndose así los costos de implantación en un equipo de desarrollo.

El Desarrollo Dirigido por Test se fundamenta en tres pilares:

- ✓ Realizar la implementación de las funciones justas y exactas que el cliente necesita y no más.
- ✓ La minimización del número de defectos que llegan al software en fase de producción.
- ✓ La producción de software modular, altamente reutilizable y preparado para el cambio a través del refactoring.

Actualmente el proceso de asignación de aulas de los laboratorios generales de computación de la ESPE se lo ha venido realizando manualmente, por lo que se busca automatizar dicho proceso mediante un sistema web en centrado a la reservación de laboratorios disponibles.

1.3.- OBJETIVOS

1.3.1.- OBJETIVO GENERAL

Realizar un estudio sobre la técnica del Desarrollo Dirigido por Tests (TDD) con enfoque en el proceso de pruebas utilizando el framework JUNIT y aplicar en el caso práctico, “Sistema web de asignación de aulas de los laboratorios generales de computación de la ESPE”, en base a los lineamientos de la metodología ágil AUP.

1.3.2.- OBJETIVOS ESPECÍFICOS

- ✓ Definir la metodología ágil y su alcance en el desarrollo de software.
- ✓ Investigar y analizar el funcionamiento y aplicación de la técnica del Desarrollo Dirigido por Test (TDD).
- ✓ Investigar el funcionamiento del framework JUNIT para aplicar sus propiedades y ventajas en el sistema web de asignación de aulas de los laboratorios generales de computación de la ESPE.

- ✓ Implementar las tres partes de la técnica TDD en el caso práctico del sistema web de asignación de aulas de los laboratorios generales de computación de la ESPE siguiendo las disciplinas de la metodología AUP.
- ✓ Establecer ventajas y desventajas de la técnica TDD.,
- ✓ Establecer conclusiones y recomendaciones de la importancia de la aplicación de la técnica TDD.

1.4.- ALCANCE

El proyecto va centrado al funcionamiento de la técnica TDD aplicada a las metodologías ágiles, con pruebas que serán suficientemente pequeñas como para que permita determinar si el SUT[2] pasa o no la prueba que ésta le impone. Además se justificará la importancia de cada tipo de test concerniente a la técnica TDD.

Previamente se realizará un estudio sobre la metodología AUP, para aplicarlo en el caso práctico conjuntamente con la técnica TDD de acuerdo a un análisis y especificación de requisitos para el sistema web de asignación de laboratorios generales de computación de la ESPE, el cual se basará en la reservación de horarios disponibles.

[2] Subject Under Test. Es el objeto que nos ocupa, el que se está diseñando a través de ejemplos.

El caso práctico será desarrollado con software libre: lenguaje java, tanto para el sistema operativo Windows como para sistemas Unix, utilizando el IDE[3] de Netbeans versión 6.7, con base de datos MySQL y la herramienta para pruebas JUNIT versión 4 con la finalidad de verificar, manejar y ejecutar conjuntos de pruebas automatizadas.

Como fase final el sistema será implantado en los laboratorios de computación de la ESPE, se entregará la documentación técnica y de usuario para garantizar el uso adecuado y la posibilidad de realizar el mantenimiento necesario al aplicativo.

[3] Integrated Development Environment. Es un programa compuesto por una serie de herramientas que utilizan los programadores para desarrollar código.

CAPÍTULO II

MARCO TEÓRICO

2.1. EL AGILISMO

El agilismo consiste en mejorar el proceso de desarrollo de software de una forma continua, maximizando el valor entregado al cliente. No es suficiente con que el proceso sea bueno hoy, sino que debe ser mejor mañana y no debe quedarse obsoleto.

2.1.1 ORIGEN DEL AGILISMO

En febrero de 2001, 17 representantes de nuevas metodologías y críticos de los modelos de mejora basados en procesos se reunieron en Utah-EEUU, convocados por Kent Beck, con la finalidad de discutir sobre el desarrollo de software, estos expertos conocían perfectamente las desventajas del clásico modelo en cascada donde primero se analiza, luego se diseña, después se implementa y, por último (en algunos casos), se escriben algunos tests automáticos.

En esta reunión nació el término “ágil” aplicado al desarrollo de software. Su objetivo fue proponer los valores y principios que deberían permitir a los equipos desarrollar software rápidamente y respondiendo a los cambios que puedan surgir a lo largo del proyecto.

Se pretendía ofrecer una alternativa a los procesos de desarrollo de software tradicionales, caracterizados por ser rígidos y dirigidos por la documentación que se genera en cada una de las actividades desarrolladas.

Tras esta reunión se creó The Agile Alliance ***(Anónimo, Agile Alliance, 2005),*** una organización, sin ánimo de lucro, dedicada a promover los conceptos relacionados con el desarrollo ágil de software y ayudar a las organizaciones para que adopten dichos conceptos. El punto de partida fue el Manifiesto Ágil, un documento que resume la filosofía "ágil".

2.1.2 MANIFIESTO ÁGIL

Los principios y valores que resaltan las metodologías ágiles fue formalizada en el manifiesto para el desarrollo de software ágil.

Este documento logra resumir en un conjunto de ideas, las prácticas que una metodología ágil debe llevar a cabo.

El manifiesto ágil se compone de cuatro valores ***(Kent Beck, 2005):***

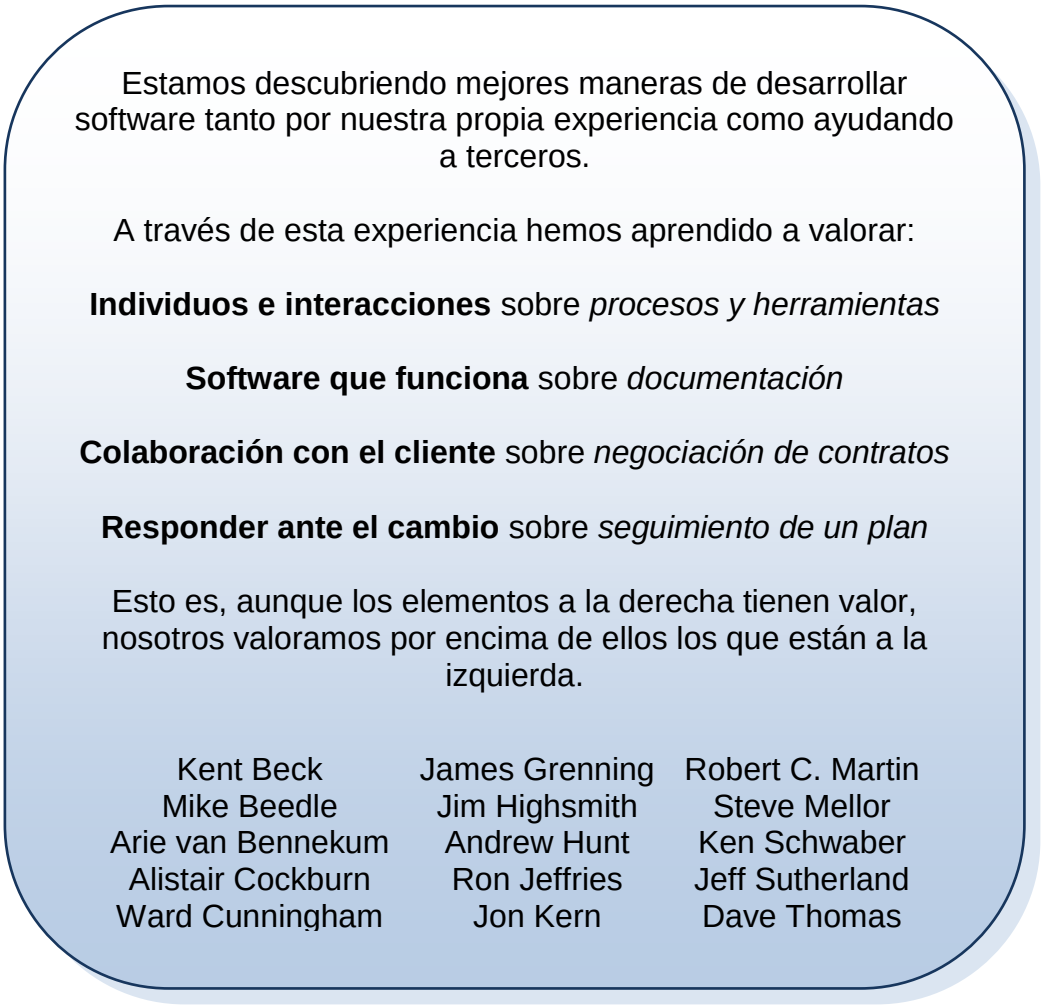


Figura 2.1: Manifiesto Ágil *(Kent Beck, 2005)*

A continuación se describe los valores que abarca el manifiesto ágil:

Los individuos y su interacción valen más que los métodos y herramientas.

Los métodos apoyados por herramientas ayudan a alcanzar resultados eficientes pero sin personas con conocimiento técnico y actitud adecuada, no se podrá obtener resultados deseados. Por lo que el elemento clave es la persona creativa, capaz de generar innovaciones a partir de lo que ve y escucha.

Los métodos constituyen una ayuda y un soporte para guiar el trabajo. Deben adaptarse a la organización, a los equipos y a las personas; y no al revés.

Desarrollar software que funciona más que conseguir una documentación exhaustiva.

La regla a seguir es no producir documentos que no aporten un valor directo al producto a menos que sean necesarios para tomar una decisión importante. Estos documentos deben ser cortos y centrarse en lo fundamental.

La colaboración con el cliente vale más que la negociación de un contrato.

En la innovación ágil el cliente es un miembro más del equipo, que se integra y colabora en el grupo de trabajo. Se propone que exista una interacción constante entre el cliente y el equipo de desarrollo. Esta colaboración entre ambos será la clave para asegurar el éxito en los proyectos.

Responder a los cambios vale más que el seguimiento de un plan.

La capacidad de responder a los cambios que puedan surgir a los largo del proyecto (cambios en los requisitos, en la tecnología, en el equipo, en el tiempo, etc.) determina también el éxito o fracaso del mismo. Por lo tanto, la planificación no debe ser estricta sino flexible y abierta.

Tras estos valores se encuentran doce principios, los cuales diferencian un proceso ágil de uno tradicional.

Los dos primeros principios son generales y resumen gran parte del espíritu ágil. El resto tienen que ver con el proceso a seguir y con el equipo de desarrollo, en cuanto metas a seguir y organización del mismo. Estos principios son:

1. *"Nuestra mayor prioridad es satisfacer al cliente a través de la entrega temprana y continua de software con valor.*
2. *Aceptamos requisitos cambiantes, incluso en etapas finales del desarrollo. Los procesos ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.*
3. *Entregamos software que funciona frecuentemente, con una periodicidad desde un par de semanas a un par de meses, con preferencia por los periodos más cortos posibles.*
4. *Los responsables de negocio y los desarrolladores deben trabajar juntos diariamente a lo largo del proyecto.*
5. *Construimos proyectos con profesionales motivados. Dándoles el entorno y soporte que necesitan, y confiando en ellos para que realicen el trabajo.*
6. *El método más eficiente y efectivo de comunicar la información a un equipo de desarrollo y entre los miembros del mismo es la conversación cara a cara.*
7. *Software que funciona es la principal medida de progreso.*
8. *Los procesos ágiles promueven el desarrollo sostenible. Los patrocinadores, desarrolladores y usuarios deben ser capaces de mantener un ritmo constante de forma indefinida.*

9. *La atención continua a la excelencia técnica y los buenos diseños mejoran la agilidad.*
10. *La simplicidad, es esencial.*
11. *Las mejores arquitecturas, requisitos y diseños surgen de equipos que se auto organizan.*
12. *A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo, entonces mejora y ajusta su comportamiento de acuerdo a sus conclusiones."* ***(Carlos Iglesias, 2005)***

2.1.3 ROLES EN EL EQUIPO

En un entorno de trabajo existen varios roles, los cuales ayudan a que lo realicen las personas mejor capacitadas con la finalidad de mejorar la calidad del producto. Roles distintos no necesariamente significa personas distintas, esto se aplica en especial en equipos reducidos. Una persona puede adoptar más de un rol, o rotar de rol a lo largo del día.

A continuación se resume los roles más comunes que existen en un proyecto de software, tomado de ***(Blé, Enero 2010):***

- ✓ **Dueño del producto:** Su misión es pedir lo que necesita, a él no le interesa como lo hagan sino el producto final funcionando correctamente, también es el encargado de aceptar o pedir correcciones sobre lo que se le entrega.
- ✓ **Cliente:** Es el dueño del producto y el usuario final.

- ✓ **Analísta de negocio:** Es el encargado de trabajar a la par con el cliente ya que será el encargado de interpretar los requisitos de la lógica del negocio en tests de aceptación con la finalidad que los desarrolladores sean capaces de entender que hay que hacer y poder aclarar dudas que se presenten.
- ✓ **Desarrolladores:** Utiliza la información adquirida por el analista de negocio para analizar como lo van a resolver además de implementar la solución.
- ✓ **Administradores de sistemas:** Se encargan de velar por los servidores y servicios que necesitan los desarrolladores.
- ✓ **Arquitecto de Software:** Es la persona capaz de tomar decisiones de diseño además de tener la capacidad de poder hablar directamente con el cliente y entender los requisitos de la lógica del negocio. En lugar de un rol, es una persona que adopta varios roles.

Hay que resaltar que en el agilismo todos los desarrolladores son arquitectos, es decir, se les permite tomar decisiones de arquitectura conforme se va escribiendo o refactorizando código.

Entender la función que cumple un rol en la industria del software es muy complicado ya que, no se está claro cuáles son los roles que desempeña cada uno, por lo que se recomienda distribuir los roles de acuerdo a las capacidades y experiencias que posee cada miembro del equipo de trabajo.

2.1.4 METODOLOGÍA TRADICIONAL VS. METODOLOGÍA ÁGIL

El desarrollo de software implica un sinnúmero de actividades y etapas, en las cuales es primordial la aplicación de una metodología adecuada y robusta acogida por el equipo de trabajo.

Las metodologías tradicionales se enfocan en ser muy exhaustivas en la documentación durante todo el ciclo del proyecto, mientras que las metodologías ágiles centran su importancia en la capacidad de respuesta a los cambios y al mantener una buena relación con el cliente para llevar al éxito el proyecto.

A continuación se presenta un resumen del estudio comparativo de las características de las metodologías ágiles vs. metodologías tradicionales, tomado por ***(Canós, 2006)***

Tabla 2.1: Diferencias entre Metodologías Tradicionales y Metodologías Ágiles

Metodologías Ágiles	Metodologías Tradicionales
Es más importante crear un producto software que funcione que escribir documentación exhaustiva	Se focalizan en documentación exhaustiva, planificación y procesos (Plantillas, técnicas de administración, revisiones, etc.)
Preparados para cambios durante el proyecto	Cierta resistencia a los cambios
Entre las principales: XP, SCRUM, ICONIX , AUP, que son las más aceptadas dentro de estas metodologías	Dentro de estas metodologías las más conocidos son Rational Unified Process (RUP) y Microsoft Solution Framework (MSF)

Proceso menos controlado, con pocos principios	Proceso mucho más controlado, con numerosas políticas/normas
No existe contrato tradicional o al menos es bastante flexible	Existe un contrato prefijado
El cliente es parte del equipo de desarrollo	El cliente interactúa con el equipo de desarrollo mediante reuniones
Grupos pequeños (<10 integrantes) y trabajando en el mismo sitio	Grupos grandes y posiblemente distribuidos
Pocos artefactos	Más artefactos
Pocos roles	Más roles
Menos énfasis en la arquitectura del software	La arquitectura del software es esencial y se expresa mediante modelos

2.2. DESARROLLO DIRIGIDO POR TESTS

2.2.1. DEFINICIÓN

Información obtenida de ***(Anónimo, TDD: Test Driven Development)***

Cuando se realiza el desarrollo de una aplicación, es muy importante probar el código que se está implementando, con la finalidad de verificar la calidad de un producto software. La prueba que comúnmente se realiza es ejecutar muchas veces la aplicación cuando está terminada y de igual manera mientras se está desarrollando, es decir, probar una y otra vez; pero eso no es óptimo, ya que se está repitiendo pruebas que ya se realizaron con anterioridad.

Para esos inconvenientes existe una solución mucho más eficiente, la cual consiste en hacer test automáticos de prueba, que se ejecutan cada vez que se compila la aplicación y para ello la mejor forma de asegurarse consiste en hacer primero los test (aplicar TDD). Primero se piensa una cosa concreta que

tiene que hacer la aplicación, luego se escribe el test que prueba que lo escrito cumple con el requerimiento específico y después (y sólo después de hacer el test), se implementa el código necesario para que el test pase correctamente.

Siguiendo esta filosofía, nace precisamente *Test Driven Development* (*TDD*). Una técnica de diseño e implementación de software, que se centra primero en hacer pruebas unitarias antes incluso de comenzar a escribir el código de un módulo, y solo después implementar el código necesario para que la prueba pase con éxito.

TDD es una técnica para diseñar software que se centra en tres pilares fundamentales ***(Blé, Enero 2010):***

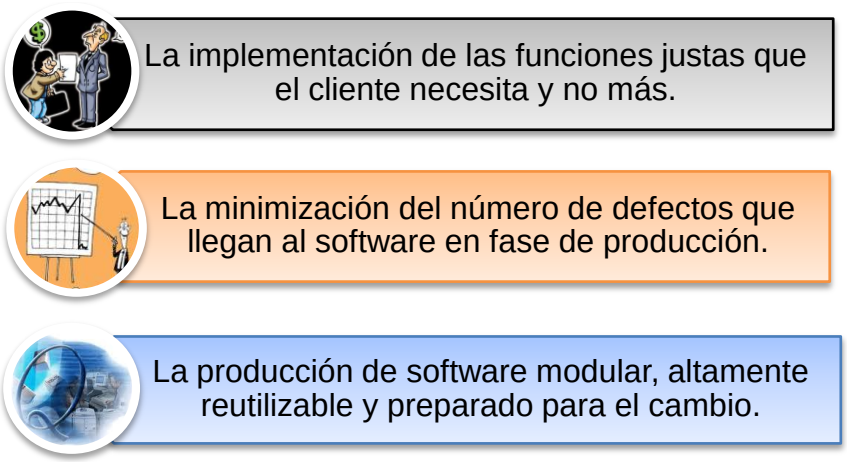


Figura 2.2: Fundamentos de la Técnica TDD

El propósito de esta técnica es lograr un código limpio que funcione. La idea es que los requerimientos sean traducidos en casos de pruebas, de este modo, cuando las pruebas sean exitosas, se garantizará que los requerimientos se hayan implementado correctamente. Con TDD se intenta traducir el caso de

uso o tarea en *X* ejemplos, hasta que el número de ejemplos sea suficiente como para describir la tarea sin lugar a malinterpretaciones de ningún tipo.

2.2.2. CARACTERÍSTICAS

La generación de pruebas para cada funcionalidad hace que el programador confíe en el código escrito. Esto permite hacer modificaciones profundas del código (posiblemente en una etapa de mantenimiento del programa) ya que si se logra hacer pasar todas las pruebas, se tendrá un código que funcione correctamente.

TDD trata de escribir el mínimo código posible, con la finalidad de obtener un resultado satisfactorio del test y así poder tener un código limpio y mantenible.

Requiere que el programador primero haga fallar los casos de prueba. La idea es asegurarse de que los test realmente funcionen pasando por la corrección de un error.

TDD permite un diseño más robusto tanto es así que a menudo se piensa a TDD como diseño manejado por las pruebas (Test Driven Design). La mayoría de las veces, el código basado en TDD es relativamente seguro, y sin duda, es bastante simple.

El código que es seguro y simple es más fácil de cambiar que el código que muestra complejidad y fragilidad. Ya que el código escrito con TDD se respalda en pruebas, cualquier cambio que rompe el código se descubre rápidamente. En esencia, el código escrito con TDD es más fácil de cambiar y más fácil de arreglar.

La calidad del software aumenta, obteniendo ***código altamente reutilizable***.

2.2.3. CICLO DE DESARROLLO DEL TDD

A continuación se explica el ciclo de desarrollo de TDD basado en el libro de *Ensayos de desarrollo impulsado por ejemplo* ***(Beck, Test Driven Development: by Example, 2003)***

2.2.3.1 Elegir un requerimiento

Consiste en que el desarrollador escribe un conjunto de requisitos que crea que dará mayor conocimiento del problema y que a la vez sea fácil de implementar. A partir de cada uno de los requisitos se crea un caso de prueba.

Ejemplo:

Requerimiento específico del caso práctico: El sistema permitirá tanto al usuario registrado como al usuario no registrado tener la opción de validar la verificación de disponibilidad de laboratorios.

2.2.3.2 Escribir una prueba

Para cada nueva función se escribe una prueba, la cual fallará en su primera ejecución ante la ausencia de código. Esta prueba deberá ser sencilla, escrita con el menor código posible.

Ejemplo:

A continuación se procede a escribir el código que cubre la prueba para el requerimiento establecido, el cual deberá ser muy sencillo, utilizando el mínimo código posible

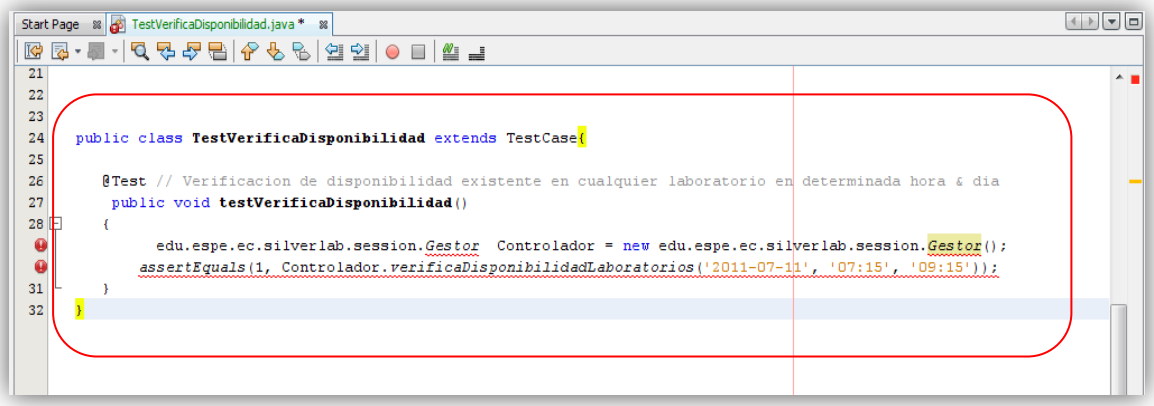


Figura 2.3: Ejemplo de Prueba Unitaria

2.2.3.3 Ejecutar la prueba y verificar si la misma falla

En este paso se valida si la prueba está ejecutada correctamente. En caso de que no falle, puede ser debido a que la funcionalidad ya esté implementada, por lo que el programador deberá entender claramente las especificaciones y los requisitos de la funcionalidad que está por implementar.

Ejemplo:

Al ejecutar el test se comprueba que el mismo no pasa con éxito debido a que intenta crear una instancia de una clase que no se encuentra implementada, por lo que al no existir el objeto llamado 'Gestores', muestra el resultado fallido de la prueba.

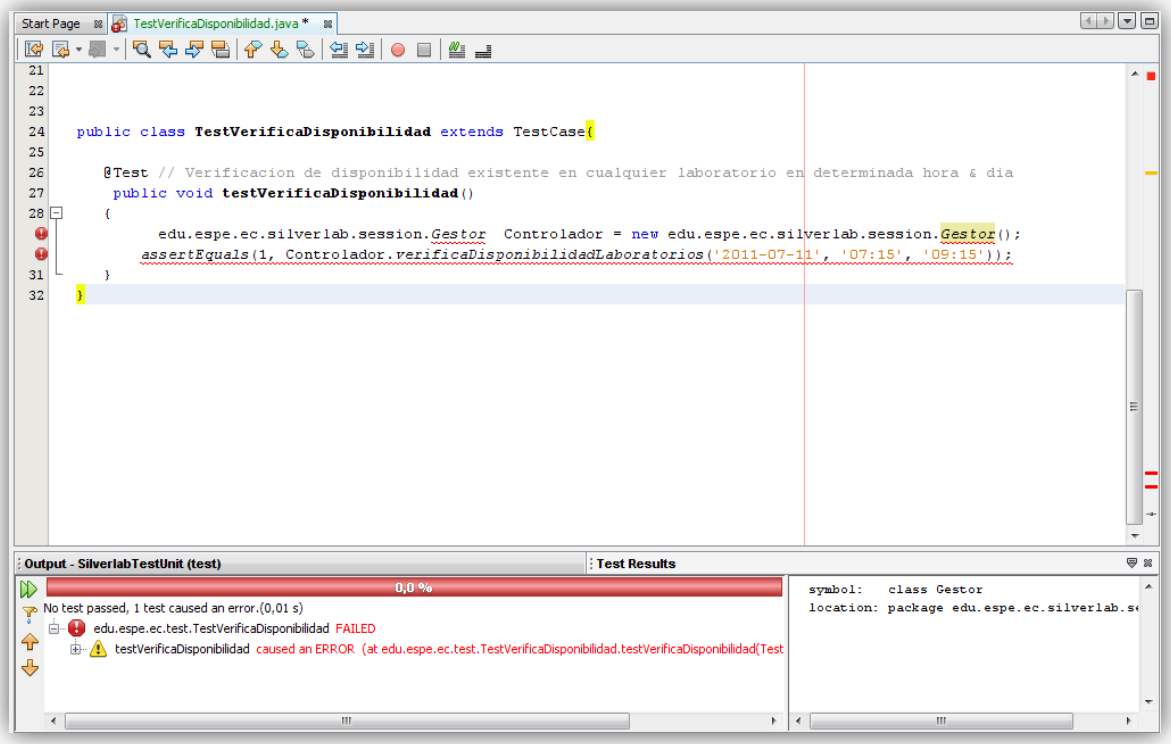


Figura 2.4: Ejecución Fallida de la Prueba Unitaria

En toda primera ejecución de un nuevo requisito, no se puede proseguir debido a la ausencia de clases necesarias para su respectiva instanciación.

2.2.3.4 Escribir la implementación

Escribir el código más sencillo hará que la prueba funcione. La idea es que la función escrita no sea lo más elegante y óptima posible, sino que simplemente, permita pasar el test unitario exitosamente.

Ejemplo:

A continuación se crea la clase Gestores y el respectivo método VerificaDisponibilidad () para poder ser instanciado.

En este punto es necesario saber exactamente cuáles son los parámetros que recibirá el método y que variable de retorno nos presentara la misma.

La función establecida valida los datos necesarios para realizar la verificación de disponibilidad de laboratorios.

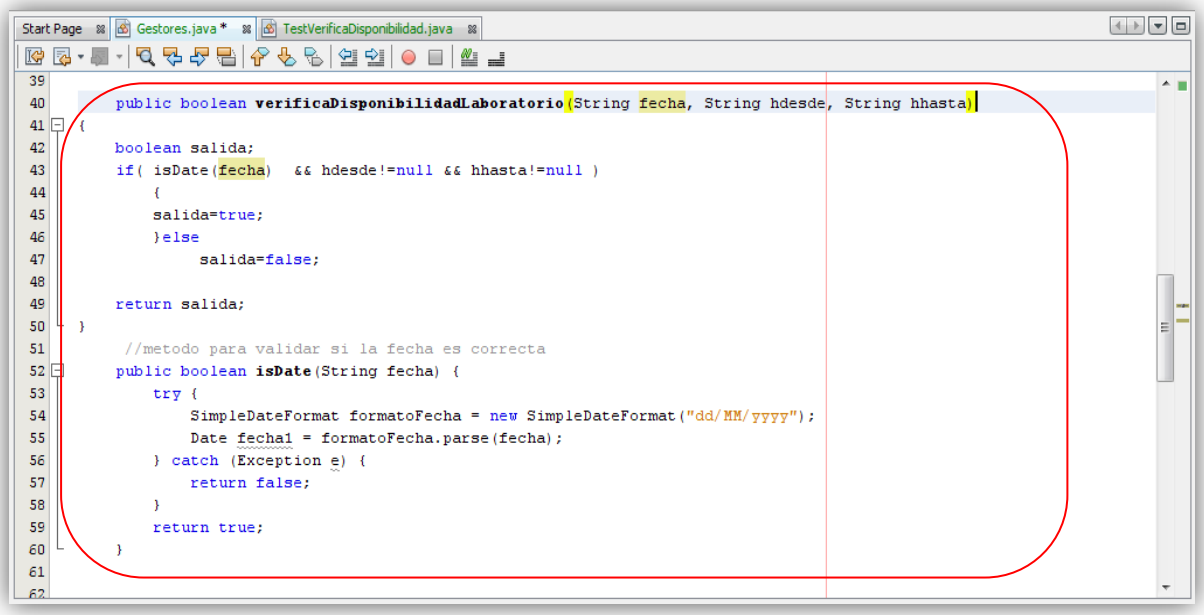


Figura 2.5: Implementación de Código

2.2.3.5 Ejecutar las pruebas automatizadas

Verificar si todo el conjunto de pruebas funciona correctamente. Si pasan, el programador puede garantizar que el código cumple todos los requisitos de la prueba. Si hay fallos, el código no resolvió dichos requisitos.

Ejemplo:

Se procede a ejecutar el test. Verificando que el mismo pase exitosamente.

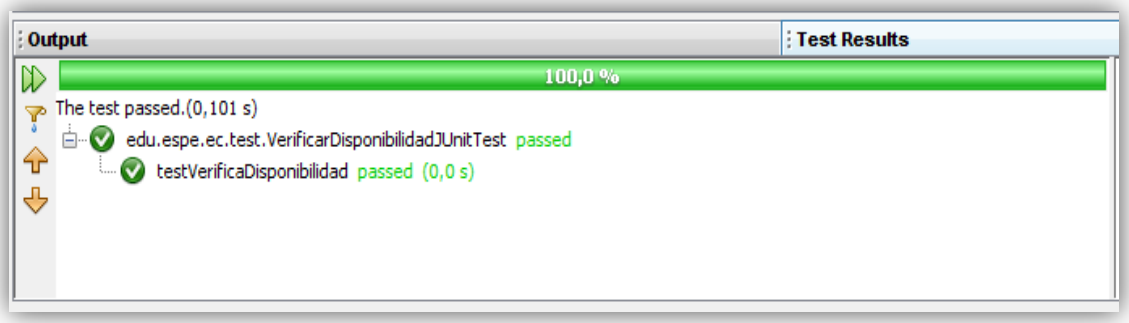


Figura 2.6: Ejecución Exitosa de la Prueba Unitaria

JUnit verifica que la clase probada funciona de la manera esperada mediante el uso de clases Asserts propias del framework.

JUnit verifica que la clase probada funciona de la manera esperada mediante el uso de clases Asserts propias del framework JUnit.

Adicionalmente se creó una lista de tests enviando parámetros de prueba con la finalidad de verificar si el código implementado cumple con el SUT.

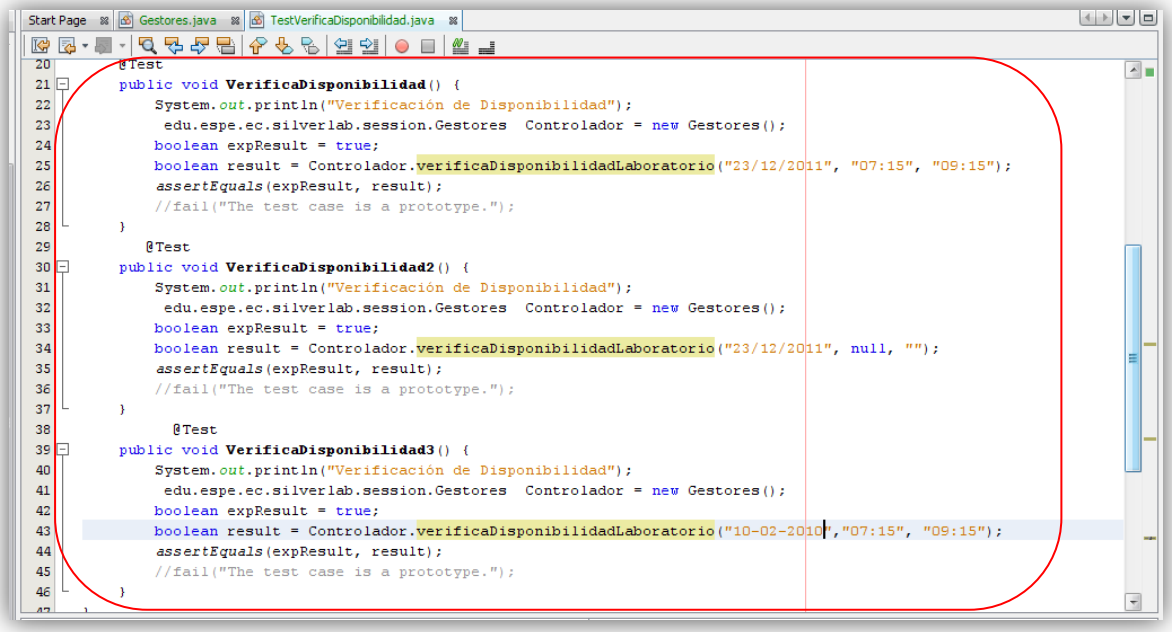


Figura 2.7: Conjunto de Pruebas

A continuación se muestra el resultado del conjunto de pruebas realizadas, donde se puede apreciar:

- ✓ Test 1.- el test pasa exitosamente debido a que los parámetros enviados al método son correctos.
- ✓ Test 2.- el test falló debido a que los parámetros correspondientes a hora_desde y hora_hasta no cumplen con lo requerido en el método.
- ✓ Test 3.- Existe fallo ya que el enviar el parámetro fecha no cumple con el formato requerido.

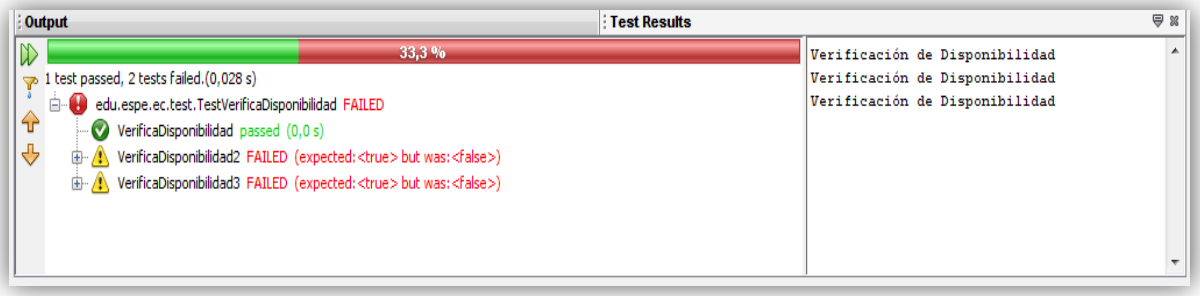


Figura 2.8: Ejecución del Conjunto de Pruebas

2.2.3.6 Refactorización

Ésta fase consiste en la restructuración del código escrito hasta el momento, sin provocar cambios en la estabilidad del conjunto creado. A la refactorización también se le llama comúnmente "limpiar el código", y es una base fundamental en la metodología del TDD, ya que permite introducir nuevas funcionalidades y casos de prueba sin poner en riesgo el comportamiento final de la aplicación, gracias a que los test se encargarán de que todo funcione correctamente, dejando un código consistente y limpio.

Ejemplo:

Con la finalidad de refactorizar el código implementado, se pudo constatar que era necesario crear variables que describan el tipo de dato que recibe el método VerificaDisponibilidadLaboratorio(), para lo cual se aporto a la semántica del código con la creación de las variables (fecha,hora_desde,hora_hasta).

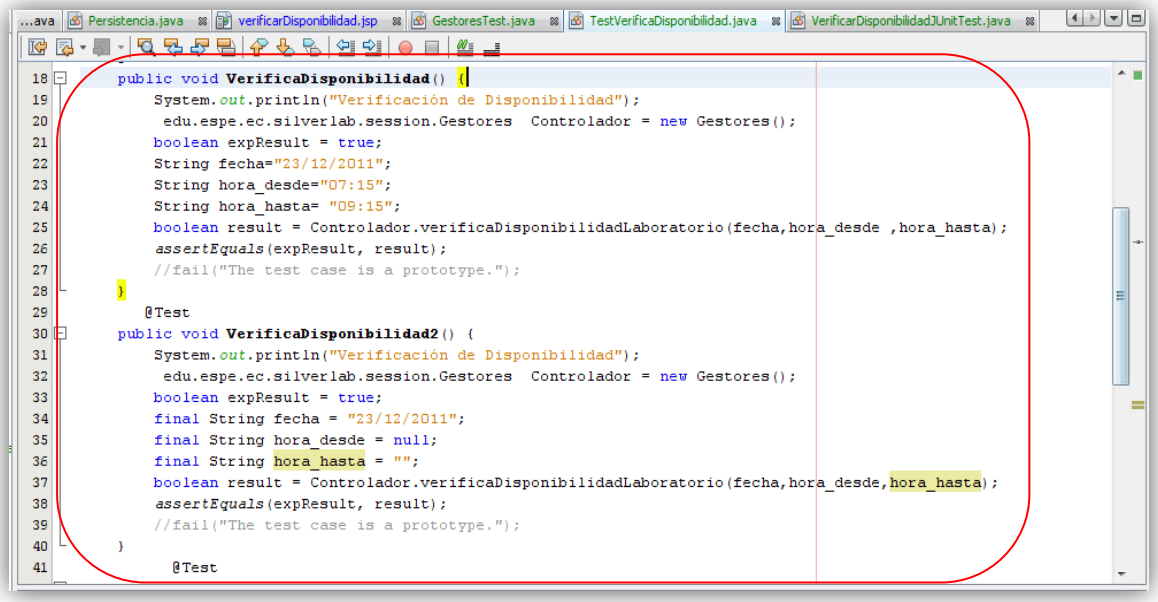


Figura 2.9: Refactorización de Código

Una vez refactorizado el código se actualiza la lista de requerimientos y se vuelve a repetir el ciclo para crear un nuevo test con un nuevo requerimiento.

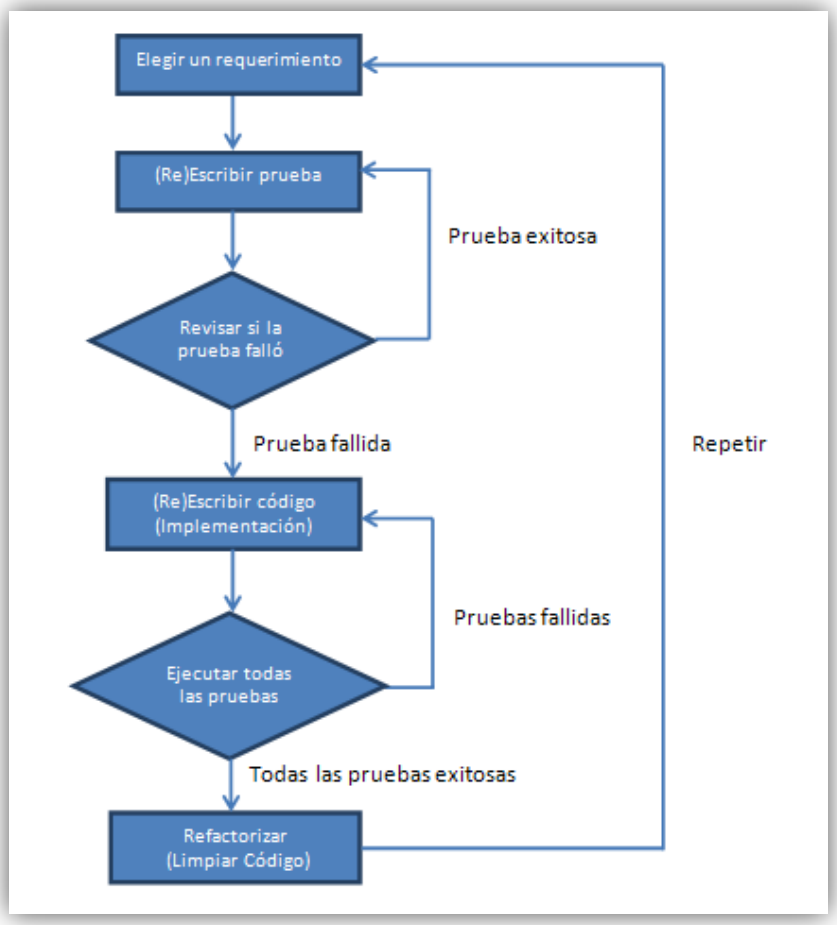


Figura 2.10: Ciclo de Desarrollo de TDD (Completo)

También este ciclo se suele abreviar como:

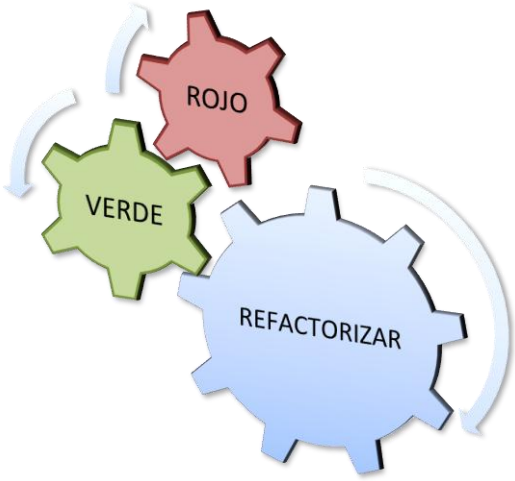


Figura 2.11: Ciclo de Desarrollo de TDD (Abreviado)

Es una descripción metafórica[4] ya que los frameworks de tests suelen colorear en rojo aquellas especificaciones que no se cumplen y en verde las que lo hacen. Así, cuando se escribe el test, el primer color es rojo porque todavía no existe código que implemente el requisito. Una vez implementado, se pasa a verde y se procede a la refactorización del código.

Integrar frecuentemente el trabajo con el del resto del equipo de desarrollo permite ejecutar todas las pruebas necesarias y así descubrir si nuestra última versión es compatible con el resto del sistema.

2.2.4. VENTAJAS Y DESVENTAJAS

VENTAJAS

El Desarrollo Dirigido por Test tiene las siguientes ventajas:

- ✓ Los casos de prueba sirven como documentación del sistema, ya que se definen de manera formal los requisitos que se esperan de la aplicación.
- ✓ Son de gran ayuda en las tareas para refactorizar y mantener código, ya que la ejecución de sus pruebas se realizan automáticamente con ayudas, por ejemplo, de bibliotecas como JUnit para aplicaciones Java.
- ✓ En un principio TDD significa escribir más código, pero la realidad indica que muchas veces ahorra código y tiempo, al disminuir la cantidad de defectos en cada uno de los métodos.

[4] Descripción Metafórica: Identificar un término real con uno imaginario con el que mantiene una relación de semejanza:

- ✓ Facilidad y fiabilidad al modificar código de las pruebas, ya que después de cualquier pequeño cambio inmediatamente se comprobará si se ha cometido un error.
- ✓ Disminuyen la necesidad de depuración.
- ✓ Posibilitan el trabajo en equipo.
- ✓ El desarrollar la prueba de una función antes de ser implementada, ayuda a pensar en cómo será la función, ordenarla y ver que errores posee.

DESVENTAJAS

Existe diversidad de opiniones acerca de las desventajas que puede conllevar el uso de esta técnica. A continuación se detallará algunas de ellas:

- ✓ En primer lugar el hecho de escribir todos los requerimientos al inicio podría provocar el abuso en la cantidad de los elegidos con el fin de dejar en el olvido alguno. El hecho de crear requerimientos de más, provoca una sobrecarga en el proyecto y también suele producir la redundancia de requerimientos lo cual trae consigo la redundancia de test.
- ✓ Escribir las pruebas antes de la implementación de código, implica que el programador deberá hacer las labores de los encargados de las pruebas. Dado que son funciones diferentes, aunque tengan relación, la buena realización de esta fase dependería del nivel de conocimiento del programador en una tarea que no es estrictamente a la que debería dedicarse.

- ✓ A veces sucede que las propias pruebas unitarias resultan más costosas, desde el punto de vista de la implementación, ya que a veces puede resultar más dificultoso encontrar los posibles fallos en los que puede caer la aplicación que la codificación de la misma.
- ✓ Si las pruebas que se crean son incorrectas entonces se implementará un código incorrecto, esto en principio sería bastante grave ya que se crearía una aplicación que no tendría el valor que uno esperaba al inicio.

2.3. LAS TRES PARTES DEL TEST

La esencia de TDD es sencilla de entenderla, lo importante es conocer cómo se realiza un test, por lo que consta de tres partes, que se identifican con las siglas AAA en inglés. *Tomado de* ***(Blé, Enero 2010):***

- ✓ **Preparar (Arrange):** Escribir la especificación del requisito convirtiéndolo en un ejemplo o test automático.
- ✓ **Actuar (Actuar):** Implementar el código según dicho ejemplo, escribiendo la mínima cantidad de código posible con la finalidad que compile el mencionado test.
- ✓ **Afirmar (Assert):** Refactorizar[5] el código con el objetivo de eliminar duplicidad y hacer mejoras.

[5] Refactorizar.- Técnica de la ingeniería de software para reestructurar un código fuente, alterando su estructura interna sin cambiar su comportamiento externo.

```
package PruebasTDD;
import org.junit.Test;
import static org.junit.Assert.*;

public class claseEjemplo {
@Test
public void NombreMetodo() {
// Preparar
        ……
        ……
// Actuar
        ……
        ……
// Afirmar
        ……
        ……
```

Figura 2.12: Ejemplo de las Partes de la Técnica TDD

En el esquema presentado se puede apreciar las tres partes del TDD, delimitadas con comentarios (//). Por ser un ejercicio práctico se lo ha comentado con la finalidad de explicar las partes mencionadas.

2.3.1. PREPARAR

Este paso es esencial ya que se debe tener claro cuál es el requisito, con la finalidad de poder plasmarlo y expresarlo en forma de test.

En este punto es cuando nace la pregunta *¿Es posible escribir un test para un código que todavía no existe?* La respuesta es, *si es posible*, por citar un ejemplo, las JSR (Java Specification Request) ***(Oracle Corporation)*** son documentos formales que describen las especificaciones y tecnologías propuestas que serán añadidas a la plataforma java, estas son creadas antes de ser implementadas posteriormente por terceras personas.

Por esa razón se concluye que un test no es inicialmente un test sino un ejemplo o especificación. Para poder escribir el test, se tiene que pensar y tratar de esforzarse por imaginar cómo sería la funcionalidad del requisito y como se comprobaría que, efectivamente, hace lo que se le pidío que haga.

Como ejemplo se realizará un test que realice la suma de dos números.

```
public void PruebaSuma()
{
assertEquals(6,
FuncionesMatematicas.multiplicacion(2,3));
}
```

Figura 2.13: Primer Paso de la Técnica TDD (Preparar)

En este punto el test escrito ni siquiera va a compilar, ya que se puede apreciar que no existe la clase ***FuncionesMatematicas*** y menos aún el método llamado ***multiplicación ().*** Por lo que se tiene que pensar que quiere que la aplicación haga.

Se necesita una clase llamada ***FuncionesMatematicas*** la cual contenga un método llamado ***multiplicación (),*** el cual recibirá dos parámetros para poder proceder a realizar la operación matemática, este método devolverá el resultado de la multiplicación.

Este sencillo caso muestra que en la realidad, muchas veces no se sabe exactamente qué clases hacer o qué métodos ponerle. Es más, muchas veces se pierde el tiempo haciendo métodos que se piensa que serán útiles en algún momento, sabiendo que estos métodos nunca van a ser utilizados. Con TDD sólo se hace lo que realmente se necesita en ese momento.

2.3.2. ACTUAR

Una vez teniendo el ejemplo escrito de forma básica donde se tiene claras las especificaciones, se procede a codificar lo mínimo posible con la finalidad de cumplir la o las especificaciones y que de esta manera pueda pasar el test.

Mínimo código significa menor número de caractéres, es decir escribirlo en menos tiempo, ya que en las siguientes iteraciones se va a proceder a refactorizar el código escrito.

En el ejemplo es necesario crear la clase ***FuncionesMatematicas*** y añadirle el método ***multiplicación ()*** el cual deberá tener dos parámetros que son el ***multiplicando*** y el ***multiplicador***, este método debe devolver algo, cualquier entero, para que al menos compile.

```
public class FuncionesMatematicas  {
   public static int multiplicacion (int a, int b) {
      return 0;
   }
}
```

Figura 2.14: Segundo Paso de la técnica TDD parte I (Actuar)

En este punto se puede compilar el test y la clase, pero si se ejecuta el test, éste fallará, ya que el método ***multiplicación ()*** devuelve 0 y no 6 como resultado de la multiplicación.

Como se muestra en la imagen, al ejecutar el test no permite pasar con éxito ya que el resultado esperado debería ser 6.

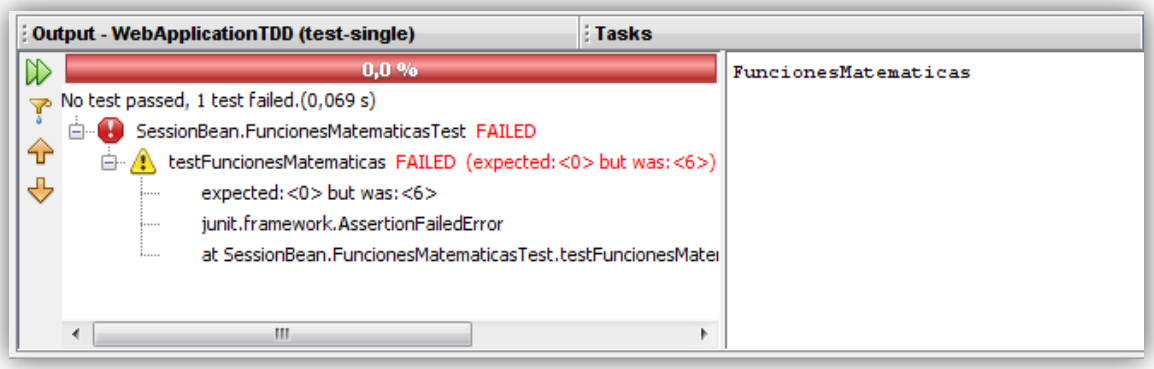


Figura 2.15: Test Fallido con JUnit

Se procede a corregir la clase de la forma inmediata para que pase el test. Y al referirse a lo más inmediato es hacer que devuelva 6.

```
public class FuncionesMatematicas  {
   public static int multiplicacion (int a, int b) {
      return 6;
   }
}
```

Figura 2.16: Segundo Paso de la Técnica TDD Parte II (Actuar)

De esta manera el método ***multiplicación ()*** realiza la función adecuada pasando con éxito el test.

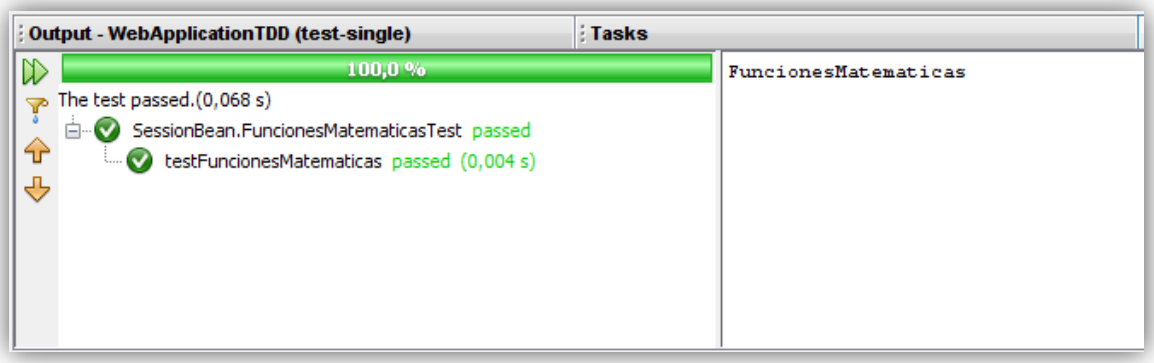


Figura 2.17. Test Exitoso con JUnit

2.3.3. AFIRMAR

Este paso es muy importante ya que se aplica la refactorización de código, cuya finalidad es reestructurar el código, cabe recalcar que al refactorizar no se debe reescribir el código, sino modificar el diseño sin alterar su comportamiento.

Se debe arreglar o modificar sobretodo las duplicidades. A veces, estas duplicidades son evidentes ya que se lo puede apreciar fácilmente en el código repetido, pero otras, como en el ejemplo, no son tan evidentes.

En el ejemplo intencionalmente se escribió la respuesta (return 6) ya que éste es el resultado de la multiplicación, la cual se la ha resuelto mentalmente. Pero al aplicar la refactorización ésta línea de código debería ser optimizada y remplazada por algo como ***(return 2 x 3).***

De igual manera los parámetros que se envía al método se los podría expresar como variables ***(int a, int b)*** quedando el ejemplo como:

```
public class FuncionesMatematicas  {
   public static int multiplicacion (int a, int b) {
      return a*b;
   }
}
```

Figura 2.18: Tercer Paso de la Técnica TDD (Afirmar)

Con la refactorización se busca mejorar el diseño del software mientras que su comportamiento sigue siendo el mismo, además ayuda al desarrollador a escribir código más claro y fácil para su mantenimiento.

Las últimas versiones de IDEs como Netbeans, Eclipse, Visual Studio, entre otros han añadido opciones automáticas de refactorización, las cuales simplifican este proceso, en ellas recomiendan mejoras como: el cambio de nombre de clases, de métodos, extracción de código a un nuevo método entre otros.

2.4 TIPOS DE TEST

El Desarrollo Dirigido por Test con la finalidad de poder entregar software de calidad, se centra en la generación de **procedimientos automatizados** que chequean el código en busca de errores, aplicando métricas (sintaxis del código, etc.) y realizando tareas regulares como la documentación.

A continuación se describirá los diferentes tipos de test tomado de ***(Blé, Enero 2010):***

2.4.1. TESTS DE ACEPTACIÓN

Los test de aceptación son el comienzo del ciclo iterativo en el desarrollo de software ya que se comienza por escribir una lista de ejemplos (test), llamados historia de usuario, en este proceso están involucrados dueños del producto, desarrolladores y responsables de calidad.

En estos test se describe los requerimientos que el cliente quiere, son escritos en lenguaje natural (cliente) y deben ser explicados en frases con alrededor de

5 palabras resumiendo los deseos del cliente. La finalidad de este tipo de test es permitir comprobar que el software cumple con un requisito de negocio.

Ejemplos de historias de usuario:

- ✓ Formulario de contáctanos.
- ✓ Darse de alta en un foro.
- ✓ Establecer un contador de visitas.
- ✓ Login en el sistema.

Se puede apreciar que estas ideas son cortas y concretas, las cuales son el resultado de escuchar al cliente y ayudarle a resumir el requisito en una sola frase. Cabe recalcar que una historia de usuario no se considera completa hasta que ha pasado sus pruebas de aceptación.

Como siguiente es necesario formar los test de aceptación para lo cual se debe preguntar acerca de los múltiples contextos y situaciones que se podrán dar con cada historia de usuario, como ejemplo se tomará la historia de usuario (“Login del sistema”).

- ✓ ¿Qué pasa si el usuario ingresado no consta en la base de datos?
- ✓ ¿Si un usuario intenta ingresar más de tres veces la contraseña que debería pasar?
- ✓ ¿Qué pasaría si mi contraseña ha sido olvidada?

Las respectivas respuestas a cada pregunta son afirmaciones, ejemplos, los cuales se deberán transformar en tests de aceptación. Por tanto, cada historia de usuario tiene asociado uno o varios test de aceptación:

- ✓ Datos ingresados incorrectamente produce el mensaje "ERROR, Verificar datos ingresados".
- ✓ Validar el máximo de tres intentos.
- ✓ Envió de contraseña al mail en caso de olvido.

En conclusión los tests de aceptación son afirmaciones en lenguaje natural que tanto el cliente, como los desarrolladores están en la capacidad de entenderlos.

2.4.2. TESTS FUNCIONALES

Son subconjuntos de los tests de aceptación que se basan en la ejecución, revisión y retroalimentación de las funcionalidades previamente establecidas.

Al hablar del aspecto funcional, se debe distinguir dos grupos:

Test Funcional.- Tienen por objetivo verificar el cumplimiento de las especificaciones aprobadas a través de la utilización de una amplia variedad de escenarios como por ejemplo: rangos de entradas normales y erróneas verificando los resultados si cumplen el requerimiento o no.

Test No funcional.- Se enfoca en la verificación de los requisitos no funcionales de una aplicación. Entre los principales se tiene:

- ✓ **Pruebas de Rendimiento:** Verifica tiempos de respuesta en la transaccionalidad de la información para eliminar los cuellos de botella y

establecer una línea base que pueda ser utilizada en un futuro para comparar el incremento en el rendimiento de la aplicación bajo pruebas.

- ✓ **Pruebas de Disponibilidad:** Tienen como objetivo verificar que los procesos de negocio estén funcionando correctamente, al margen del estado de sus recursos hardware. Se debe asegurar que el entorno esté funcionando como se espera y su disponibilidad está dentro de los límites que han sido previamente establecidos.
- ✓ **Pruebas de Seguridad:** Cubren el proceso de evaluación de la seguridad desde un punto de vista externo. Se debe utilizar políticas de seguridad para su respectivo análisis con el fin de encontrar fallos de seguridad, tanto en el diseño, como en la implementación.
- ✓ **Pruebas OAT (Pruebas de Aceptación Operacional):** El objetivo general de este tipo de pruebas es comprobar que la aplicación dispone de la fiabilidad y el soporte necesario para su puesta en marcha en producción.

2.4.3. TESTS DE SISTEMA

Estos tests recorren de extremo a extremo la aplicación, son probados simulando a un usuario común, en la que se detalla puntos de entrada reales, verificando el proceso y validando información almacenada en la base de datos. Para esto existen software que facilitan las pruebas como el plugin Selenium ***(Anónimo, SeleniumHQ)*** para el navegador Mozilla Firefox, el cual es un set de herramientas ofrecidas como Open Source que permiten desarrollar scripts para pruebas de aplicaciones Web. Además permite realizar

una especie de grabación de las actividades para luego poder reproducirlas y detectar cambios en la respuesta del sitio web.

El beneficio de usar Selenium es que evita el trabajo repetitivo, es decir, probar una y otra vez el mismo test manualmente.

2.4.4. TESTS UNITARIOS

Los test unitarios son los más importantes dentro del Desarrollo Dirigido por Test. Un test unitario (Unit Test) es la manera de probar código desarrollado con el objetivo de verificar que una rutina o método del código tiene un correcto y esperado funcionamiento.

Un test unitario consiste en un pequeño programa que instancia clases e invoca métodos de forma automática, la forma más fácil de usar test unitarios es usando frameworks de pruebas automatizadas como por ejemplo (JUnit ***(JUnit Group)***, NUnit ***(Beck, NUnit.org, 2002),*** MbUnit ***(Viklund, 2010),*** etc.)

En TDD los test unitarios son las bases, ya que fomenta a que se escriba primero la prueba y luego la implementación, y que los objetos aseguren su funcionamiento sin depender de otras implementaciones. Esto obliga a tener en claro las interfaces de negocio que se usarán, y ser responsables por el comportamiento de cada objeto.

Características del test unitario:

- ✓ **Atómico:** El test debe probar la mínima cantidad de funcionalidad posible, es decir limitarse al comportamiento exclusivo de un método de una clase definida.
- ✓ **Independiente:** Los test unitarios no pueden depender de otros para producir un resultado satisfactorio. No puede formar parte de una secuencia de tests que se deba ejecutar en un determinado orden.
- ✓ **Inocuo:** Las pruebas sobre los tests unitarios no deben alterar el estado del sistema. Al ejecutarlo una vez, produce exactamente el mismo resultado que al ejecutarlo n número de veces. Además el test no debe alterar la base de datos.
- ✓ **Rápido:** Ya que los tests unitarios son repetibles y se los ejecuta cada pocos minutos es improductivo tener que esperar durante la ejecución de un test, por lo que la velocidad de ejecución es un factor importante, por ese motivo Kent Beck, Erich Gamma y su equipo han desarrollado un conjunto de bibliotecas disponibles para los IDEs de Eclipse y Netbeans, llamado **JUnit,** cuya función es ejecutar los tests unitarios a la par que se escribe el código, de esta manera se optimiza el tiempo de ejecución de las pruebas.

En conclusión, los tests unitarios pretenden tener pequeñas partes de código, que se testearán por separado y de forma independiente**,** es decir cada uno de los métodos de las distintas clases que compongan el sistema, ayudando a la fácil, rápida y eficaz corrección de código ante posibles errores.

2.4.5. TESTS DE INTEGRACIÓN

Es la fase del testeo de software en la cual los módulos individuales de software son combinados y testeados como un grupo. Son las pruebas posteriores a las pruebas unitarias y preceden el testeo de sistema.

Los tests de integración se encargan de comprobar que los diferentes componentes del sistema funcionen como es debido.

"*Integrar*" adquiere un significado algo ambiguo y se refiere tanto a integrar dos piezas de software que se deben comunicar entre sí, como a integrar esas piezas de software en un entorno de ejecución determinado.

Para la ejecución de estas pruebas, lo ideal es mantener un equipo de desarrollo de tests de integración que se encargue de desplegar los componentes en un sistema similar al de producción y de probar la integración de estos componentes. Estos equipos de producción deben realizar el test del producto olvidándose del usuario final. Esto permite eliminar la dependencia de los casos de uso y alcanzar una cobertura mucho mayor en el software, ya que incluso aunque se estuviera cubriendo el 100% de los casos de uso definidos en la arquitectura, la realidad es que probablemente los casos de uso no están cubriendo el 100% de las acciones que los usuarios realizarán.

Mediante pruebas de integración no solo se permite probar la interacción de un módulo con otro dentro del mismo sistema, sino que se puede probar las conexiones con otros sistemas externos como por ejemplo un gestor de Base de Datos.

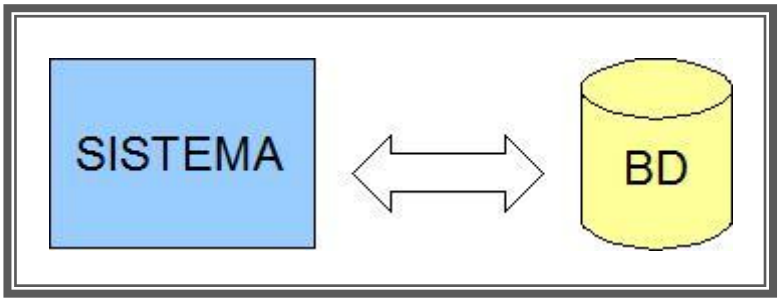


Figura 2.19: Pruebas de Integración con un Gestor de Base de Datos

2.4.6. IMPORTANCIA

Los tipos de test facilitan los cambios en la aplicación ya que las pruebas aseguran que los nuevos cambios no han introducido errores.

Separa la interfaz de la implementación, dada que la única interacción entre los casos de prueba y las unidades bajo prueba son las interfaces de éstas últimas, se puede cambiar cualquiera de los dos sin afectar al otro.

Los errores están más acotados y son más fáciles de localizar, dado que se tiene preparado un test para cada función que puede desenmascararlo.

Los test unitarios aceleran el desarrollo del software debido a que ayudan a tener un código desacoplado gracias a que cada una de las funciones está pensada para devolver un resultado que podrá ser testeado.

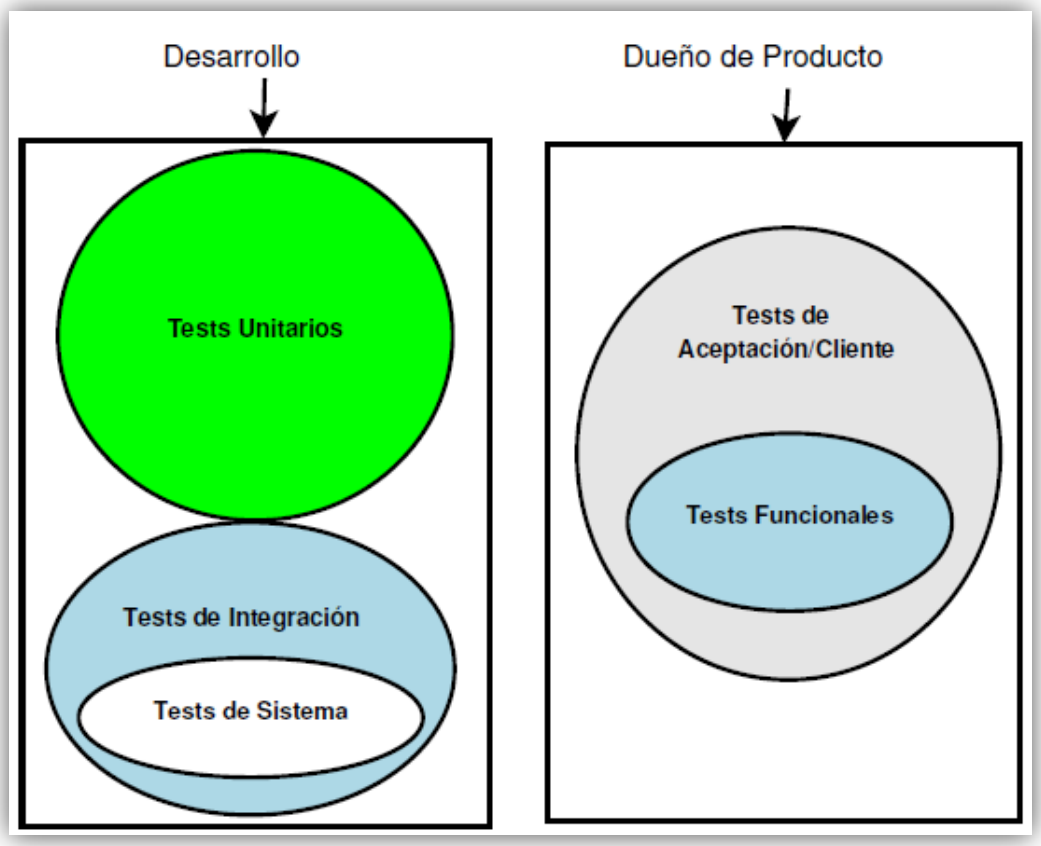


Figura 2.20: Tipos de TEST *(Blé, Enero 2010)*

2.5. METODOLOGÍA AGILE UNIFIED PROCESS (AUP)

2.5.1. INFORMACIÓN GENERAL

El Proceso Unificado es un desarrollo iterativo e incremental. La elaboración, construcción y fases de transición se dividen en una serie de iteraciones de duración fija. Cada iteración resulta en un *incremento,* es decir una versión del sistema mejorado. Los resultados de cada iteración, sobre todo en la fase de elaboración, deben ser seleccionados con el fin de garantizar que los mayores riesgos se abordan en primer lugar. ***(DesaSoft Desarrollo de Software, 2009)***

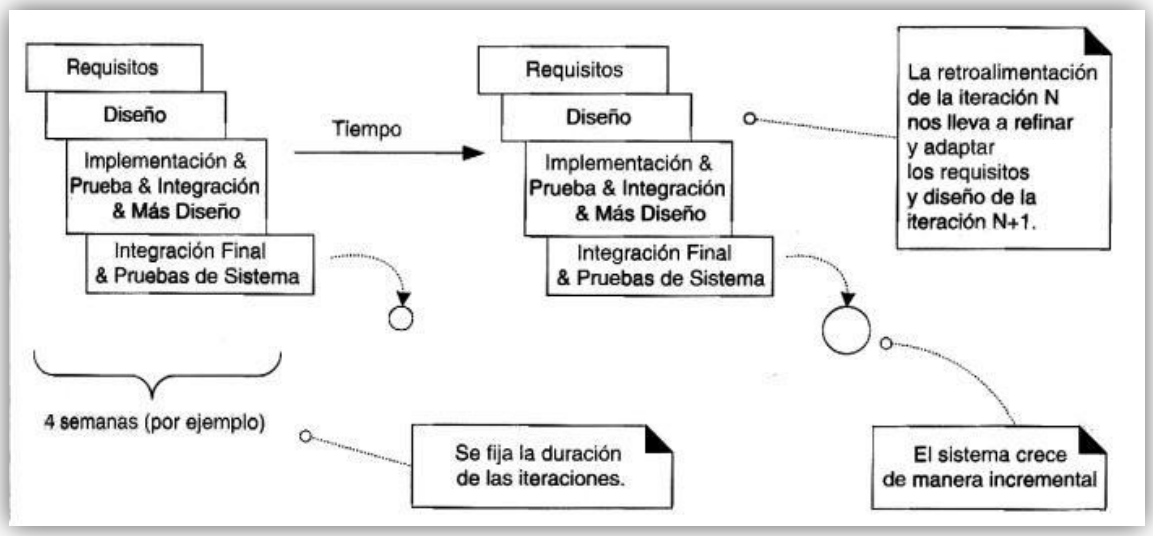


Figura 2.21: Desarrollo Iterativo e Incremental en el UP *(DesaSoft Desarrollo de Software, 2009)*

El Proceso Unificado Ágil (Agile UP) es una versión simplificada de Rational Unified Process (RUP) de IBM desarrollado por Scott Ambler[6].

Esta metodología describe un enfoque simple y fácil de entender para el desarrollo de software usando técnicas y conceptos que aún se mantienen vigentes en RUP. AUP se aplica a técnicas ágiles incluyendo el Desarrollo Dirigido por Test (TDD), Modelado Ágil, gestión del cambio ágil, y refactorización de base de datos para mejorar la productividad.

2.5.2. CICLO DE VIDA DE AUP

El ciclo de vida de AUP se divide en fases y disciplinas. En la siguiente figura se puede observar que para cada disciplina existe una fase.

[6] **Scott W. Ambler** (1966) canadiense ingeniero de software, consultor y escritor, actualmente Jefe de Práctica en Desarrollo Ágil de IBM Corporation en los métodos de grupo de IBM.

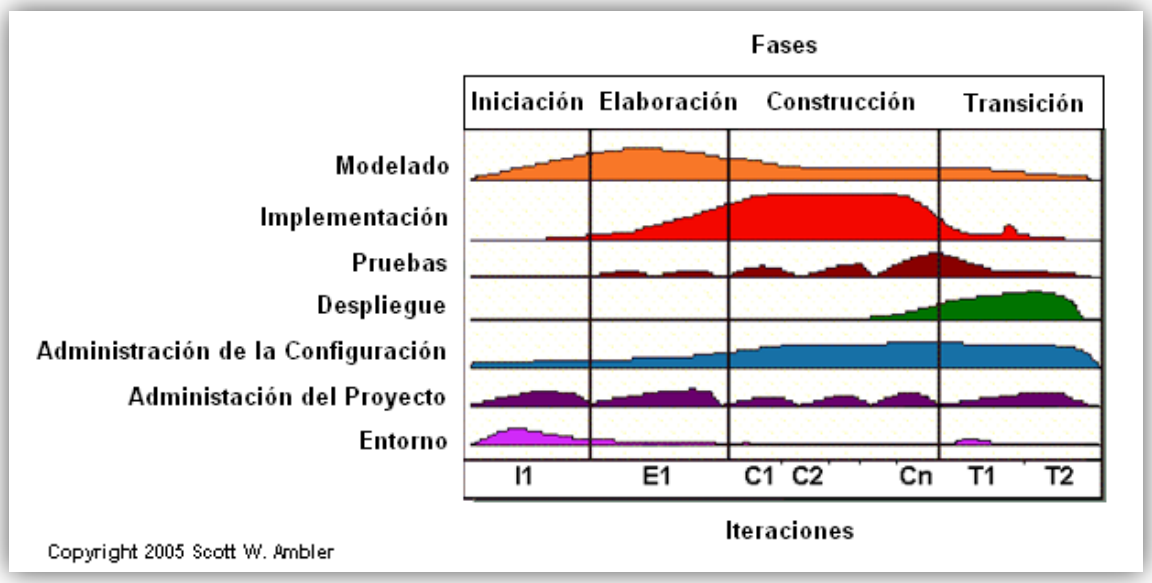


Figura 2.22: Ciclo de Vida de AUP *(Ambler, El Proceso Unificado Ágil, 2005)*

2.5.2.1 FASES DEL CICLO DE VIDA

A continuación se explicará la función que cumple cada fase del ciclo. ***(Ambler, Disciplinas de Agile UP, 2005)***

Tabla 2.2: Fases del Ciclo de Vida de AUP

FASES	OBJETIVO	ACTIVIDADES
INICIACIÓN	Definir la razón de ser y el alcance del proyecto.	-Identificar alcance del proyecto -Definir riesgos -Determinar la factibilidad del proyecto -Preparar el entorno del proyecto

ELABORACIÓN	Establecer un plan de proyecto y una arquitectura correcta del sistema.	-Definir la arquitectura básica -Analizar los riesgos -Planificar el proyecto En esta fase se incluye la creación de diagramas de casos de uso, diagramas de clases y diagramas de paquete.
CONSTRUCCIÓN	Desarrollar el sistema hasta que se encuentre listo para la preproducción de pruebas.	-Análisis -Diseño -Implementación / Codificación -Pruebas (individuales, de integración)
TRANSICIÓN	El sistema se lleva a los entornos de preproducción donde se somete a pruebas de validación y aceptación y finalmente se despliega en los sistemas de producción.	-Corregir defectos -Validar el sistema -Finalizar el modelo de pruebas

2.5.2.2 DISCIPLINA DEL CICLO DE VIDA

En las siguientes tablas se explicará la función que cumple cada disciplina en cada fase del ciclo, tomado de ***(Ambler, Disciplinas de Agile UP, 2005)***

2.5.2.2.1. MODELADO

El objetivo de esta disciplina es entender el negocio de la organización e identificar las mejores soluciones para los problemas que se presenten.

Tabla 2.3: Actividades por Fases de la Disciplina Modelado

FASES	ACTIVIDADES
INICIACIÓN	Se define el alcance inicial, por lo que se debe considerar: - Analizar la utilización de casos de uso. - Identificar las principales reglas del negocio y requerimientos técnicos. - Iniciar el desarrollo de un glosario que describa términos importantes tanto técnicos como del negocio. Para obtener un buen resultado lo ideal es unir al personal técnico, así como los desarrolladores para identificar y planear estrategias de arquitectura.
ELABORACIÓN	**Identificar riesgos técnicos:** Se debe tomar en cuenta los posibles riesgos técnicos que se pueden presentar en el proyecto ya sea por la introducción de nuevas tecnologías, actualización de las mismas, estrés en la aplicación o sistemas actuales externos. **Prototipado de interfaces de usuario:** Para la realización del prototipo se debe considerar solo las pantallas principales, páginas de su interfaz de usuario, ya que las necesidades de prototipos pueden cambiar y, por tanto, su trabajo tendría que ser desechado.
CONSTRUCCIÓN	Se deberá trabajar conjuntamente con los interesados del proyecto para poder entender las necesidades. En lugar de las descripciones de casos de uso, de reglas del negocio y de requerimientos técnicos, se puede escribir casos de prueba de aceptación. El objetivo es hacer sólo lo suficiente pensar en el diseño de un simple requerimiento, antes de la implementación del mismo. Aquí se puede crear: - Diagrama de secuencia de UML - Modelo de despliegue - Diagrama de Clases - Modelo físico de datos
TRANSICIÓN	En esta fase se deberá hacer algún modelado en el momento (Just In Time) para tratar de entender las causas principales de un defecto. Durante esta fase se finalizará la documentación general del sistema ya que el alcance debe estar bien establecido.

2.5.2.2.2 IMPLEMENTACIÓN

Ésta disciplina transforma los modelos en un código ejecutable y realiza un nivel básico de pruebas.

Tabla 2.4: Actividades por Fases de la Disciplina Implementación

FASES	ACTIVIDADES
INICIACIÓN	**Prototipado técnico:** Es necesario analizar un aspecto de un requisito con el fin de entender lo suficiente para permitir estimar el esfuerzo requerido. **Prototipado de Interfaces de Usuario:** Se debe considerar prototipar las interfaces de usuario (pantallas, reportes, y manuales), al menos las pantallas principales, ya que los usuarios quieren ver qué es lo que van a obtener antes de comprarlo.
ELABORACIÓN	**Probar la arquitectura:** Se realiza a través del desarrollo de la arquitectura del prototipo extremo a extremo para su sistema, y a la vez mitigando gran parte de los riegos técnicos en el proyecto.
CONSTRUCCIÓN	Se realizan las **primeras pruebas** mediante un acercamiento al Desarrollo Dirigido por Test (TDD) para todos los aspectos de la aplicación. Además se puede ir construyendo el sistema cada vez que el código fuente cambie, utilizando Cruise Control **(SourceForge.net),** el cual monitorea su sistema de control de versiones para cambios al código y regeneración a como las necesite. **Evolución de las interfaces de usuario:** Se debe tomar en cuenta que la interface de usuario es el sistema para la mayoría de usuarios, por lo que el software debe ser lo más usable posible siguiendo las estrategias de diseño de interfaces de usuario y usabilidad.
TRANSICIÓN	Esta fase debe centrarse en la corrección de defectos que se han encontrados como resultado de las pruebas.

2.5.2.2.3. PRUEBAS

El objetivo de ésta disciplina es realizar una evaluación objetiva para asegurar la calidad. Esto incluye encontrar defectos, validar que el sistema funcione como fue diseñado y verificar que los requerimientos están completos.

Tabla 2.5: Actividades por Fases de la Disciplina Pruebas

FASES	ACTIVIDADES
INICIACIÓN	**Planificación de pruebas:** El objetivo principal es identificar cuántas pruebas se necesita hacer, quien será el responsable de hacerlas, el nivel de participación requerido por los usuarios, los tipos de herramientas y los entornos necesarios. Debe estar claro y disponible el plan del proyecto, la visión, el modelo de requerimientos inicial, un modelo de arquitectura inicial y así poder comunicar el alcance y la arquitectura del sistema a las personas que están activamente envueltos en el desarrollo de los modelos.
ELABORACIÓN	**Validación de la Arquitectura:** Se debe tomar un enfoque del Desarrollo Dirigido por Pruebas (TDD) para construir el prototipo técnico el cual compruebe la arquitectura del sistema. **Modelo de pruebas:** Se desarrollará un conjunto de casos de prueba, compuesta por la unidad de pruebas del TDD en la implementación, la aceptación de pruebas del modelado, y las pruebas del sistema (funcionamiento, integración). En este punto el reporte de defectos será simplemente la salida del paquete de pruebas.
CONSTRUCCIÓN	**Pruebas de software:** Además de las unidades de prueba de los desarrolladores se deberá hacer pruebas de instalación, pruebas de esfuerzos como la carga, pruebas de función, y pruebas de aceptación de usuario. Todas dentro de un ambiente de pruebas de pre-producción.
TRANSICIÓN	**Validación del sistema:** Se realizarán grandes pruebas como las de sistema, las de integración y aceptación. Su objetivo es probar completamente el sistema dentro del ambiente de pruebas de pre-producción. **Análisis del modelo de pruebas:** Se tendrá que seguir ejecutando el conjunto de casos de prueba y actualizarlo hasta que el sistema esté listo para ser desplegado en producción. El reporte de defectos será registrado, junto con los detalles para que los desarrolladores puedan corregirlos.

2.5.2.2.4. DESPLIEGUE

Planifica la entrega del sistema y ejecutar el plan para que el sistema esté disponible para los usuarios finales.

Tabla 2.6: Actividades por Fases de la Disciplina Despliegue

FASES	ACTIVIDADES
INICIACIÓN	Se identifica el tiempo que se pondrá al sistema en producción, esto debe ser aplicado correctamente ya que deben tomar en cuenta todos los riesgos que pueden correr. Además se realiza un **plan de entregables**: El principal objetivo es determinar una estrategia general de implementación basada en entender el proyecto.
ELABORACIÓN	Aparte de definir la arquitectura también es importante definir la configuración de los entregables del sistema, como por ejemplo soportar una configuración cliente/servidor de tres capas, una interfaz basada en HTML para usarla en Internet, entre otros; Estas configuraciones pueden ser documentadas como parte del modelo de despliegue que se basa en la organización de los componentes tanto del software actual como del hardware, todo esto puede ayudar a identificar los diferentes tipos de instalación que se deben realizar.
CONSTRUCCIÓN	**Desarrollar el script de instalación:** Estos scripts deben estar escritos de manera que se puedan reconfigurar fácilmente para ponerlo en producción. **Actualizar el plan:** De manera cómo progrese el desarrollo del sistema, debe avanzar el plan de implementación. **Implementar el sistema en entornos de pre-producción:** Se debe entregar regularmente el sistema en un ambiente de pre-producción para efectuar pruebas y llevar a cabo el control de calidad necesario, así como realizar demostraciones a los involucrados.
TRANSICIÓN	En esta fase se realiza una "última" revisión de software, así como la conversión o migración de datos. Adicionalmente se debe capacitar a los clientes o usuarios del proyecto, así como a la administración, equipo de operaciones y equipo de soporte. Hay que tomar en cuenta que los usuarios pueden necesitar capacitación más allá que aprender a trabajar con su aplicación.

2.5.2.2.5. ADMINISTRACIÓN DE LA CONFIGURACIÓN

Administra el acceso a los entregables o productos del proyecto. Esto no solo incluye el seguimiento de las versiones de los productos, sino también controlar y administrar los cambios sobre ellos.

Tabla 2.7: Actividades por Fases de la Disciplina Administración de la Configuración

FASES	ACTIVIDADES
INICIACIÓN	Se establece la **configuración del ambiente** mediante: - La estructura de directorios. - Los miembros del equipo del proyecto necesitan tener acceso al folder o directorios del proyecto y cualquier otro software instalado en sus ordenadores, además de conocer los conceptos básicos de la administración de configuración y las herramientas necesarias. - También cada usuario debe poner su trabajo bajo el Control de Gestión de Configuración en una base regular, verificar las entradas y salidas en cada caso, resolver conflictos de actualización cuando se requiera, y presupuestar los productos del proyecto cuando las versiones más actualizadas estén aprobadas.
ELABORACIÓN	En esta fase se analiza la arquitectura del sistema y los entregables a través del Control de Gestión de Configuración que se centra en establecer y mantener la consistencia del sistema, los atributos funcionales, el diseño y el rendimiento del producto.
CONSTRUCCIÓN	Los scripts para la instalación, la actualización del plan del proyecto y en sí el código del sistema debe pasar por el Control de Gestión de Configuración para determinar las características de seguridad apropiadas y la seguridad de que se utilizan para medir el estado de configuración del sistema.
TRANSICIÓN	Todas las pruebas que se realizan a lo largo del ciclo de vida del sistema como las de aceptación y unitarias deben pasar por el Control de Gestión de Configuración para garantizar el control de los cambios realizados.

2.5.2.2.6. ADMINISTRACIÓN DEL PROYECTO

Dirige las actividades que se lleva a cabo en el proyecto. Esto incluye administración del riesgo, la dirección de personas y coordinar con los sistemas que están fuera del alcance del proyecto.

Tabla 2.8: Actividades por Fases de la Disciplina Administración del Proyecto

FASES	ACTIVIDADES
INICIACIÓN	En esta fase se **determina** si el proyecto es viable. Debe ser capaz de construirlo y mantenerlo en funcionamiento una vez que lo pongan en producción, y la organización debe ser capaz de apoyar el proyecto. También se desarrolla estratégias para enfrentar los diferentes riesgos. También demostrar que entienden el alcance, que pueda entender y abordar los riesgos, y que tiene un plan viable para proceder.
ELABORACIÓN	**Construir el equipo:** Durante esta fase se necesitará personas con habilidades de análisis, desarrollo e implementación. **Obtener recursos:** El equipo necesita financiación, instalaciones (por ejemplo salas y cubículos), hardware, software, y así sucesivamente para hacer su trabajo. Se tendrá que ejecutar la revisión del ciclo de vida de la arquitectura, con la finalidad de demostrar que la arquitectura funciona y que se está enfrentando correctamente a los principales riesgos del proyecto.
CONSTRUCCIÓN	**Manejo del riesgo:** Continuar con la gestión y control del riesgo. **Actualizar el plan del proyecto:** Se deberá considerar las necesidades de los equipos de operación y soporte, capacitación del usuario final, y el plan de pruebas al sistema piloto. Se tendrá que ejecutar la revisión de la estabilidad del sistema, aceptación del riesgo, plan del proyecto, cuya finalidad es demostrar que el equipo ha desarrollado un sistema que está potencialmente listo para implementarse en producción.
TRANSICIÓN	**Administrar el equipo:** Debe incluir en el equipo de desarrolladores, encargados de pruebas e implementadores. Se tendrá que ejecutar la revisión de los productos entregables, cuya principal finalidad es demostrar que su sistema ha pasado las pruebas y es aceptable para los involucrados.

2.5.2.2.7. ENTORNO

El objetivo de esta disciplina es apoyar el resto de los esfuerzos para garantizar que el proceso sea adecuado, como la orientación (normas y directrices) y herramientas (hardware, software, etc.) las cuales deben estar disponibles para cualquier equipo según sea necesario.

Tabla 2.9: Actividades por Fases de la Disciplina Entorno

FASES	ACTIVIDADES
INICIACIÓN	Se realiza una actualización del entorno de trabajo ya que se puede integrar gente nueva al equipo. Además se identifica la categoría del proyecto, ya que muchas organizaciones desarrollan varias versiones de sus procesos de software, lo cual en este momento es en donde se empieza el ajuste al AUP para cumplir con las necesidades de cada proyecto.
ELABORACIÓN	El proyecto progresa a medida que se entiende la evolución de los requisitos, la estrategia de la arquitectura, y el enfoque general, para esto se necesita evolucionar el entorno, instalando nuevas herramientas, o remover las herramientas que ya no se necesitan.
CONSTRUCCIÓN	En este momento de la fase se realiza el apoyo al equipo, es decir ayudar para utilizar o configurar herramientas, elegir correctamente las plantillas para una adecuada documentación. **Ambiente de capacitaciones:** A medida que progresa el plan de despliegue se necesita realizar capacitaciones tanto al usuario como personal de soporte y personal de operación.
TRANSICIÓN	**Configuración de las operaciones:** Se necesita una versión del sistema configurada para simular reportes de defectos en una forma segura. **Recopilar licencias de software:** A medida que el proyecto llega a la conclusión puede ser necesario desinstalar las licencias de software en los equipos que ya no necesitan el mismo para que las licencias puedan estar disponibles a los demás dentro de la organización.

2.5.3. FILOSOFÍA DE AUP

La filosofía de las metodologías ágiles dan mayor valor al individuo, a la colaboración con el cliente y al desarrollo incremental del software con iteraciones muy cortas.

AUP se basa en las siguientes filosofías:

- ✓ ***"El personal sabe lo que están haciendo:*** *La gente no va a leer la documentación detallada del proceso, pero desearán una guía de alto nivel y/o capacitación de vez en cuando.*
- ✓ ***Simplicidad:*** *Todo está descrito de manera concisa.*
- ✓ ***Agilidad:*** *AUP se ajusta a los valores y principios del desarrollo de software ágil y la Alianza Ágil.*
- ✓ ***Enfocarse en actividades de alto valor:*** *Se centra en las actividades que realmente cuentan, no en todas las cosas que podrían sucederle en un proyecto.*
- ✓ ***La independencia de herramientas:*** *Se puede utilizar cualquier conjunto de herramientas que desee en AUP. Pero es recomendable que use las herramientas que mejor se adapten al trabajo, que son con frecuencia herramientas simples o inclusive herramientas de código abierto."* ***(Inteco Instituto Nacional de Tecnologias de la Comunicación, España, 2010)***

2.6. HERRAMIENTAS DE DESARROLLO

2.6.1. JAVA ENTERPRISE EDITION (JAVA EE)

"Conocido como J2EE hasta la versión 1.4, es una plataforma de programación que permite desarrollar e implantar software de aplicaciones en un lenguaje de programación Java con arquitectura n niveles distribuida". ***(Cantero, 2004)***

Java EE define rigurosamente un conjunto de servicios que un servidor de aplicaciones debe tener, junto con una API estándar para acceder a estos servicios.

Java EE incluye varias especificaciones de API como:

JDBC, permite la ejecución de operaciones sobre base de datos desde el lenguaje Java.

SERVICIOS WEB, permite el intercambio de datos entre varias aplicaciones desarrolladas en diferentes lenguajes de programación.

XML, permite la definición de la gramática de lenguajes específicos de acuerdo a las diferentes necesidades de una manera estructurada.

Java EE también contiene algunas especificaciones únicas como:

Enterprise JavaBeans, proporcionan un modelo de componentes distribuidos, mediante el servidor de aplicaciones, proveen objetos desde el lado del servidor.

Servlets, permite la generación de páginas web de una manera dinámica a partir de los parámetros de la petición que envíe el navegador web.

Portlets, son componentes modulares de interfaz de usuario visualizadas en un portal web, que son agregados en una página.

Java Server Pages, simplifica el desarrollo de interfaces de usuarios en aplicaciones Java EE.

2.6.2. JAVASCRIPT

JavaScript es un lenguaje de programación interpretado, por lo que no es necesario compilar los programas para ejecutarlos. En otras palabras, los programas escritos con JavaScript se pueden probar directamente en cualquier navegador sin necesidad de procesos intermedios.

A pesar de su nombre, JavaScript no guarda ninguna relación directa con el lenguaje de programación Java. Legalmente, JavaScript es una marca registrada de la empresa Sun Microsystems, como se puede ver en http://www.sun.com/suntrademarks/ ***(Pérez, 2009)***

El lenguaje Javascript puede interactuar con el código HTML mediante la utilización de etiquetas, permitiendo a los programadores web utilizar contenido dinámico. Por ejemplo, hace fácil responder a los acontecimientos iniciados por usuarios (como introducción de datos en formularios) sin tener que utilizar CGI[7]. El lenguaje Javascript es ***opensource***[8], por lo que cualquier persona puede utilizarlo sin comprar una licencia.

[7] CGI.- 'Common Gateway Interface' es de las primeras formas de programación web dinámica.

2.6.3. JAVA SERVER PAGES (JSP)

2.6.3.1 DEFINICIÓN DE JSP

"JSP es un acrónimo de Java Server Pages, viene a ser como Paginas de Servidor Java, pues es una tecnología orientada a la creación y generación de páginas web dinámicas en servidor con programación Java." ***(Alvarez, 2002)***

Las JSP's permiten la utilización de código Java mediante scripts. Además, es posible utilizar algunas acciones JSP predefinidas mediante etiquetas. Estas etiquetas pueden ser enriquecidas mediante la utilización de Bibliotecas de Etiquetas externas e incluso personalizadas.

2.6.3.2 CARACTERÍSTICAS DE JSP

- ✓ Se ejecutan en una maquina virtual JAVA.
- ✓ Su persistencia le permite realizar peticiones de manera más eficiente como conexiones a bases de datos y el manejo de sesiones.
- ✓ Están compuestas de código HTML/XML mezclado con etiquetas esenciales para programar scripts de servidor en sintaxis Java.
- ✓ Permite generar archivos .jsp que incluyen dentro estructuras de etiquetas HTML.

[8] Open source.- Software que está licenciado de tal manera que los usuarios pueden estudiar, modificar y mejorar su diseño mediante la disponibilidad de su código fuente.

- ✓ Permite la extensión de librerías de etiquetas, mediante la utilización de JSTL (librerías core, XML, SQL, fmt), lo cual brinda funcionalidad de propósito general a muchas aplicaciones web.
- ✓ Tiene un buen desempeño y es más eficiente que otras tecnologías web que ejecutan el código de una manera puramente interpretada.

2.6.3.3 VENTAJAS DE JSP

JSP utiliza lenguaje Java, el mismo que apoya la creación de clases que manejen lógica del negocio y acceso a datos de una manera prolija.

De esta forma permite que se separe en niveles las aplicaciones web, dejando la parte encargada de la generación del documento HTML en el archivo JSP.

2.6.4. JAVA SERVER FACES (JSF)

2.6.4.1 DEFINICIÓN DE JSF

Es un framework que permite el desarrollo de aplicaciones Java las cuales se basan en aplicaciones web, de una manera que simplifican el desarrollo de interfaces de usuarios en aplicaciones Java EE.

"JSF utiliza JSP como la tecnología que permite realizar el despliegue de las páginas." ***(Anónimo, 2006)***

JSF ofrece un marco de trabajo que facilita el desarrollo de aplicaciones, separando las diferentes capas de una arquitectura: presentación, reglas y entidades de negocio.

2.6.4.2 CARACTERÍSTICAS DE JSF

- ✓ Un modelo de trabajo basado en componentes UI (user interface), definidos por medio de etiquetas y XML.
- ✓ Una arquitectura basada en el patrón MVC.
- ✓ Incluye la capa de control, definida de forma declarativa en archivos XML. Lo que implica control de eventos y errores.
- ✓ Validación en cliente y en servidor.
- ✓ Control de mensajes y roles.
- ✓ Es muy flexible, porque permite la creación de nuevos componentes.

2.6.5. JQUERY

2.6.5.1 DEFINICIÓN DE JQUERY

JQuery es una librería de **JavaScript** que permite trabajar con AJAX facilitando la manipulación de HTML o JSON, y ofrece una serie de utilidades para programar de forma más limpia y libre de errores. La forma básica de interactuar con la página es mediante la función $() (alias de JQuery ()), que recibe como parámetro una expresión CSS o el nombre de una etiqueta HTML. ***(Utreras, 2009)***

Ejemplo:

$(‘*’); Devuelve todos los elementos existentes en una página web

$(‘.boton’); Devuelve todos los elementos que contenga la clase ‘boton’.

2.6.5.2 CARACTERÍSTICAS DE JQUERY

- ✓ Manipular estilos CSS.
- ✓ Crear efectos visuales
- ✓ Modificar el contenido de la página y apariencia de la misma
- ✓ Manejar eventos de los elementos de la página
- ✓ Compatible con los navegadores Mozilla Firefox 2.0+, Internet Explorer 6+, Safari 3+, Opera 10.6+ y Google Chrome 8+.
- ✓ Utiliza componentes GUI.

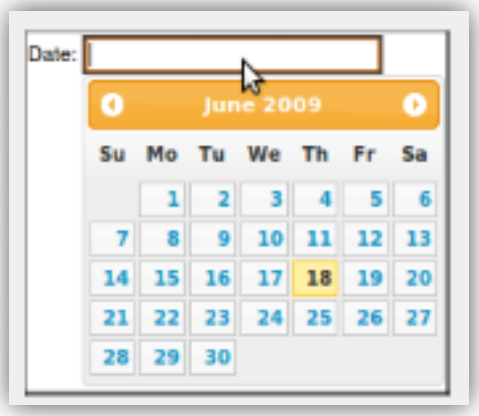


Figura 2.23: Calendario con JQuery *(Utreras, 2009)*

2.6.6. FRAMEWORK JUNIT

JUnit es un Framework Open Source para la automatización de las pruebas (tanto unitarias, como de integración) en los proyectos Software. El framework provee al usuario de herramientas, clases y métodos que le facilitan la tarea de realizar pruebas en su sistema y así asegurar su consistencia y funcionalidad.

JUnit trae consigo el propio framework, como *runners* para ejecutar directamente los test o *Assert* para comprobar la validez de los resultados. ***(Beck, Test Driven Development: by Example, 2003)***

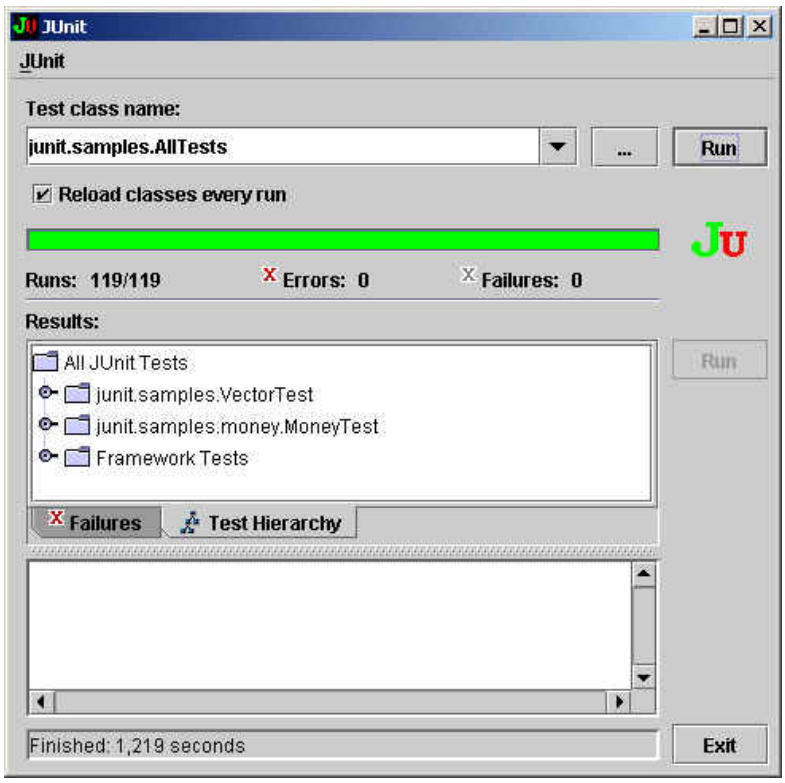


Figura 2.24: Framework JUnit

Métodos

Una clase de Test realizada para ser tratada por JUnit tiene una estructura con los siguientes métodos:

Método setUp: Se debe asignar valores iniciales a las variables antes de la ejecución de cada test. Si solo se quiere que se inicialicen al principio una vez, el método debe llamarse "setUpClass"

Método tearDown: Es llamado después de cada test y puede servir para liberar recursos o similar. Igual que antes, si se quiere que solo se llame al final de la ejecución de todos los test, se debe llamar "tearDownClass"

Métodos Test: Contienen las pruebas concretas que se va a realizar.

2.6.7. MYSQL

2.6.7.1 DEFINICIÓN DE MYSQL

MySQL es un sistema de gestión de bases de datos relacional, licenciado bajo la GPL de la GNU. Su diseño multihilo le permite soportar una gran carga de forma muy eficiente, además de que existen infinidad de librerías y otras herramientas que permiten su uso a través de gran cantidad de lenguajes de programación, además de su fácil instalación y configuración. ***(Oracle Corporation, 1997)***

2.6.7.2 CARACTERÍSTICAS DE MYSQL

- ✓ Aprovecha la potencia de sistemas multiprocesador, gracias a su implementación multihilo.
- ✓ Soporta gran cantidad de tipos de datos.
- ✓ Dispone de API's en gran cantidad de lenguajes (C, C++, Java, PHP, etc).
- ✓ Gran portabilidad entre sistemas.
- ✓ Soporta hasta 32 índices por tabla.

- ✓ Gestión de usuarios y contraseñas, manteniendo un muy buen nivel de seguridad en los datos.

2.6.8 PMD

2.6.8.1 DEFINICIÓN DE PMD

PMD es una herramienta de análisis que posee un conjunto de reglas las cuales permitirán comprobar que la aplicación cumpla con las mismas, para así obtener un código más elegante, sencillo y mantenible.

PMD se puede utilizar desde línea de comandos, o puede integrarse con varios IDEs y herramientas, como Eclipse, NetBeans, Maven o JEdit. Además pertenece al conjunto de proyectos de software libre de SourceForge.net. ***(Iso25000, 2009)***

2.6.8.2 CARACTERÍSTICAS DE PMD

PMD escanea el código fuente Java y busca problemas potenciales como:

- ✓ **Posibles defectos:** sentencias try/catch/finally/switch vacías.
- ✓ **Código muerto:** variables, parámetros y métodos no utilizados.
- ✓ **Código no óptimo:** uso ineficiente del StringBuffer, etc.
- ✓ **Expresiones innecesarias:** sentencias “if” innecesarias, bucles “for” que pueden ser de tipo “while”.

- ✓ **Código duplicado:** el código copiado y pegado significa copiar y pegar defectos.

PMD también permite a los usuarios crear sus propias reglas con las que analiza el código fuente. De manera que si la comprobación que se quiere realizar no corresponde a ninguna de las reglas existentes PMD facilita dos maneras para definir nuevas reglas:

- ✓ Mediante métodos Java.
- ✓ Mediante expresiones en XPath.

La herramienta PMD incluye además el módulo conocido como CPD “Copy Paste Detector”, el cual permite detectar el código duplicado existente en el programa. Al copiar y pegar código se debe tener en cuenta que también se copian y pegan los defectos de éste, y el módulo CPD permite detectar los archivos que contienen el fragmento repetido identificando donde comienza el código repetido dentro de cada uno de los archivos.

2.6.9 SOURCEFORGE

2.6.9.1 DEFINICIÓN DE SOURCEFORGE

SourceForge es un servidor web para desarrollos de software, que controla y gestiona varios proyectos de software libre y actúa como un repositorio de código fuente**. *(E2G Consultores, 2011)***

2.6.9.2 CARACTERÍSITICAS DE SOURCEFORGE

- ✓ Integración de forma segura (SSL) a través del web de múltiples servicios útiles para desarrolladores.
- ✓ Administración vía web.
- ✓ Entre sus métodos de distribución de proyectos destacan archivos/ficheros comprimidos en Zip y otros compatibles con Linux también paquetes de Windows o Mac y en algunos casos se distribuye únicamente el código fuente del proyecto.
- ✓ Uso y gestión de foros y wikis por cada proyecto.
- ✓ Gestión de la documentación.
- ✓ Hosting de páginas web sobre el proyecto. Se puede usar PHP y MySQL.
- ✓ Página con datos resumen sobre el estado del proyecto, estadísticas, etc.
- ✓ SourceForge Incluye proyectos compatibles con Linux, Mac, Windows.
- ✓ Gestión de releases del software.

CAPÍTULO III

ANÁLISIS, DISEÑO Y DESARROLLO DEL CASO PRÁCTICO

3.1 ESPECIFICACIÓN DE REQUISITOS DE SOFTWARE (ERS)

3.1.1 INTRODUCCIÓN

3.1.1.1 Propósito

El propósito de esta sección es especificar de manera clara y precisa las funcionalidades y restricciones del sistema a desarrollar, principalmente se centra en la funcionalidad requerida por los desarrolladores y los usuarios finales. Esta funcionalidad se basa principalmente en presentar un panorama general del sistema y además se presentan los requerimientos funcionales y no funcionales.

3.1.1.2 Alcance

El sistema se centra en la asignación de laboratorios generales de computación de la ESPE para las hora(s) requeridas por el usuario final, es decir el registro de datos a la base.

Los roles de usuarios permitirán el acceso al sistema, otorgando un nivel de seguridad y transaccionalidad sobre las entidades disponibles.

Cada laboratorio tiene sus respectivas características, por lo que el usuario podrá escoger el laboratorio que se acople a sus necesidades, siempre y cuando haya verificado la disponibilidad del mismo.

Las respectivas entidades que posee el sistema podrán ser administradas únicamente por el perfil administrador el cual podrá:

- Crear registros.
- Modificar registros.
- Eliminar registros.

3.1.1.3 Definiciones, Siglas y Abreviaturas

Tabla 3.1: Definición y Siglas de ERS

Definición / Siglas	Descripción
NRC	Código asignado a las materias impartidas por un determinado profesor, la cual es otorgada por el sistema Banner [9]de la Escuela Politécnica del Ejército
DISPONIBILIDAD	Laboratorios que se encuentran libres para su reserva.
IDBANNER	Código asignado a cada usuario de la ESPE, obtenido por el sistema BANNER.

3.1.1.4 Referencias

IEEE Recommended Practice for Software Requirements Specification IEEE Std 830-1998

[9] Banner.- Sistema de Gestión Académica de la Escuela Politécnica del Ejército.

3.1.2 DESCRIPCIÓN GENERAL

3.1.2.1 Perspectiva del Producto

El Sistema web SILVERLAB se compone de las siguientes entidades las cuales abarcan el proceso de reserva de laboratorios con la finalidad de satisfacer las necesidades de los usuarios.

Las entidades son:

- Persona.- Entidad que almacena las personas que pueden acceder al sistema cada una con su respectivo perfil de acceso sobre el mismo.
- Laboratorio.- Entidad creada con la finalidad de llevar el control de laboratorios disponibles en la Escuela Politécnica del Ejército.
- Materia.- Entidad que almacena el listado de materias disponibles para cada carrera, la cual es administrada por el código NRC otorgado por el sistema Banner.
- Carrera.- Listado de carreras disponibles en la Escuela Politécnica del Ejército.
- Período Académico.- Gestiona los periodos académicos configurados en el cual se detalla fechas correspondientes al inicio y fin de clases.
- Reservas.- Entidad principal de la gestión de laboratorios, ya que es la encargada de integrar información correspondiente a un específico laboratorio en un horario establecido.

Es importante tener claro que el sistema web utiliza cierta información del sistema BANNER como es el código correspondiente al NRC.

3.1.2.2 Funcionalidades del Sistema

El sistema SILVERLAB es un producto de software de ambiente web, el cual contiene una estricta relación de entidades con la finalidad de cubrir el proceso de reserva de laboratorios, de manera eficiente, ágil y confiable en cuanto al manejo de información, a través del cumplimiento de los siguientes objetivos:

- Supervisar el conjunto de actividades que se realizan al registrar una reserva.
- Garantizar la calidad de la información almacenada en el sistema.
- Evitar la duplicidad de reservas.

3.1.2.3 Características de los Usuarios

Para operar el Sistema SILVERLAB, se han establecido tres tipos de roles de usuario, los cuales se describen a continuación:

Tabla 3.2: Características de los Usuarios

No.	Nombre	Descripción
1	Administrador	Total acceso a todas las opciones del sistema.
2	Usuario Registrado	Consulta de información, creación de reservas y cambio de su contraseña.
3	Usuario No Registrado	Acceso limitado de solo lectura sobre la entidad (Laboratorio) y podrá verificar y visualizar la disponibilidad de los mismos.

Figura 3.1: Jerarquía de Usuarios

3.1.2.4 Restricciones

El sistema SILVERLAB, será desarrollado con base a la información suministrada por el jefe de laboratorios generales de computación de la ESPE, a través de esta Especificación de Requerimientos de Software.

El sistema debe ser implementado en el lenguaje de programación JAVA, y se debe usar como motor de base de datos MySQL.

3.1.2.5 Suposiciones y Dependencias

3.1.2.5.1 Suposiciones

- Se asume que los requisitos descritos en esta sección son estables una vez que sea aprobado dicho documento.
- Se requiere que el cliente cuente con manejador de base de datos MySQL versión 4.2.1 o superior y un sistema Windows 2003 o superior, así como con los equipos de cómputo necesarios para el montaje del sistema de información.
- Se asume que los equipos de los usuarios cumplen con los requisitos mínimos necesarios para su producción (acceso a Internet).

3.1.2.5.2 Dependencias

El sistema de información funciona autónomamente, sin necesidad de comunicarse con otros sistemas externos a la institución, por lo que no hay dependencias respecto de otros sistemas y recursos.

3.1.3 REQUISITOS ESPECÍFICOS

3.1.3.1 Requisitos de Interfaces Externas

3.1.3.1.1 Interfaces de Usuario

Tabla 3.3: Descripción de Interfaces de Usuario

ID	RNF-001
Tipo	Interfaces de usuario
Descripción	El sistema web debe por seguridad permitir al cliente tener una clave de acceso con ciertos privilegios, de acuerdo al usuario.

3.1.3.1.2 Interfaces de Hardware

Tabla 3.4: Descripción de Interfaces de Hardware

ID	RNF-002
Tipo	Interfaces de hardware
Descripción	La computadora debe permitir el acceso al servicio de Internet con cualquier Sistema Operativo: Windows, Unix o Mac Os.

3.1.3.1.3 Interfaces de Software

Tabla 3.5: Descripción de Interfaces de Software

ID	RNF-003
Tipo	Interfaces de software
Descripción	Se implementará sobre Internet como medio de acceso, por lo que se considera el uso de servidores y navegadores web.

3.1.3.1.4 Interfaces de Comunicación

Tabla 3.6: Descripción de Interfaces de Comunicación

ID	RNF-004
Tipo	Interfaces de comunicaciones
Descripción	El sistema será accedido a través de computadores conectados a Internet sobre el protocolo de comunicaciones TCP/IP.

3.1.3.2 Requisitos Funcionales

REQ01: Información de Usuario

El sistema deberá registrar la información correspondiente a los usuarios registrados.

REQ02: Estado de Usuario

El sistema deberá permitir la configuración del estado de usuario correspondiente a activo – inactivo.

REQ03: Cambio de Contraseña

El sistema deberá permitir la actualización de contraseñas para los usuarios registrados en el sistema.

REQ04: Disponibilidad de Laboratorios

El sistema permitirá tanto al usuario registrado como al usuario no registrado tener la opción de visualizar y verificar disponibilidad de laboratorios.

REQ05: Administración de Períodos Académicos

El administrador del sistema podrá controlar los diferentes períodos académicos en el que podrá visualizar, crear, actualizar y eliminar registros.

REQ06: Administración de Laboratorios

El administrador tendrá control sobre la administración de los laboratorios disponibles en el cual podrá visualizar, crear, actualizar y eliminar registros.

REQ07: Administración de Carreras

El administrador tendrá control sobre la administración de las carreras en el cual podrá visualizar, crear, actualizar y eliminar registros.

REQ08: Administración de Personas

El administrador tendrá control sobre la administración de las personas disponibles en el cual podrá visualizar, crear, actualizar y eliminar registros.

REQ09: Registro de Reservas

El usuario registrado será la persona con acceso a la creación de reservas en la cual se deberá registrar datos como: fecha, día de la reserva, hora desde, hora hasta, laboratorio disponible, período académico actual, materia que dicta, información correspondiente al NRC y carreras.

REQ10: Manejo de Sesiones

El sistema se deberá manejar mediante el concepto de sesiones en el cual se detalle el usuario y su perfil de acceso para proceder a utilizar los servicios del sistema correspondientes a su rol de usuario.

3.1.3.3 Requisitos No Funcionales

REQ11: Mantenibilidad

El sistema es susceptible de ser ampliado. Por tanto deberá asegurarse su fácil implementación, aplicando para su desarrollo las metodologías que para ello sean precisas.

REQ12: Disponibilidad

Estar disponible 100% o muy cercano a esta disponibilidad durante el horario hábil laboral de la ESPE (Ejemplo: de lunes a viernes de 7:15 a.m. a 9:30 p.m., con excepción de los días festivos).

REQ13: Escalabilidad

El sistema debe estar en capacidad de permitir en el futuro el desarrollo de nuevas funcionalidades, modificar o eliminar funcionalidades después de su construcción y puesta en marcha inicial.

REQ14: Facilidad de Uso

El sistema debe ser de fácil uso y entrenamiento por parte de los usuarios de la ESPE, así como de fácil adaptación de la institución con el mismo.

El sistema debe presentar mensajes de error que permitan al usuario identificar el tipo de error y comunicarse con el administrador del sistema.

REQ15: Flexibilidad

El sistema deberá tener la capacidad de poder adoptarse a nuevos necesidades o requerimientos de manera que se garantice la capacidad de reacción ante los cambios del entorno.

REQ16: Seguridad

El acceso al Sistema debe estar restringido por el uso de contraseñas asignadas a cada uno de los usuarios. Sólo podrán ingresar al Sistema las personas que estén registradas, estos usuarios serán clasificados en varios tipos de roles con acceso a las opciones de trabajo definidas para cada rol.

El control de acceso implementado debe permitir asignar los perfiles para cada uno de los roles identificados.

REQ17: Consumo de Recursos

La complejidad de la aplicación del usuario debe ser baja, y consumir pocos recursos.

REQ18: Ciclo de Vida

El ciclo de vida elegido para desarrollar el sistema será incremental, de manera que se puedan incorporar fácilmente cambios y nuevas funciones.

REQ19: Software Libre

El sistema será software libre y, por tanto, cualquier componente software que reutilice también deberá ser libre.

3.2 CASOS DE USO

Para explicar claramente la funcionalidad del sistema SILVERLAB, se utilizará diagramación de casos de uso.

Los casos de uso son una técnica para la especificación de requisitos funcionales que forma parte de la propuesta de UML [Boo99].

Un caso de uso es la descripción de una secuencia de interacciones entre el sistema y uno o más actores obteniendo como resultado una determinada acción.

Para su correcto entendimiento es necesario explicar dos características principales de la diagramación UML <<include>> y <<extends>>.

Include.- En una relación de dos casos de uso, cuando el caso de uso base incluye a un caso de uso B, quiere decir que el segundo es parte esencial del primero es decir sin el segundo, el primero no podría funcionar o cumplir su objetivo.

Extends.- En una relación de caso de uso principal hay ocasiones en la que el caso de uso de extensión no es necesario que se cumpla y cuando lo hace extiende o la da otro sentido al caso de uso principal.

A continuación se presenta los Diagramas Contextuales de cada usuario que interactúa con el sistema.

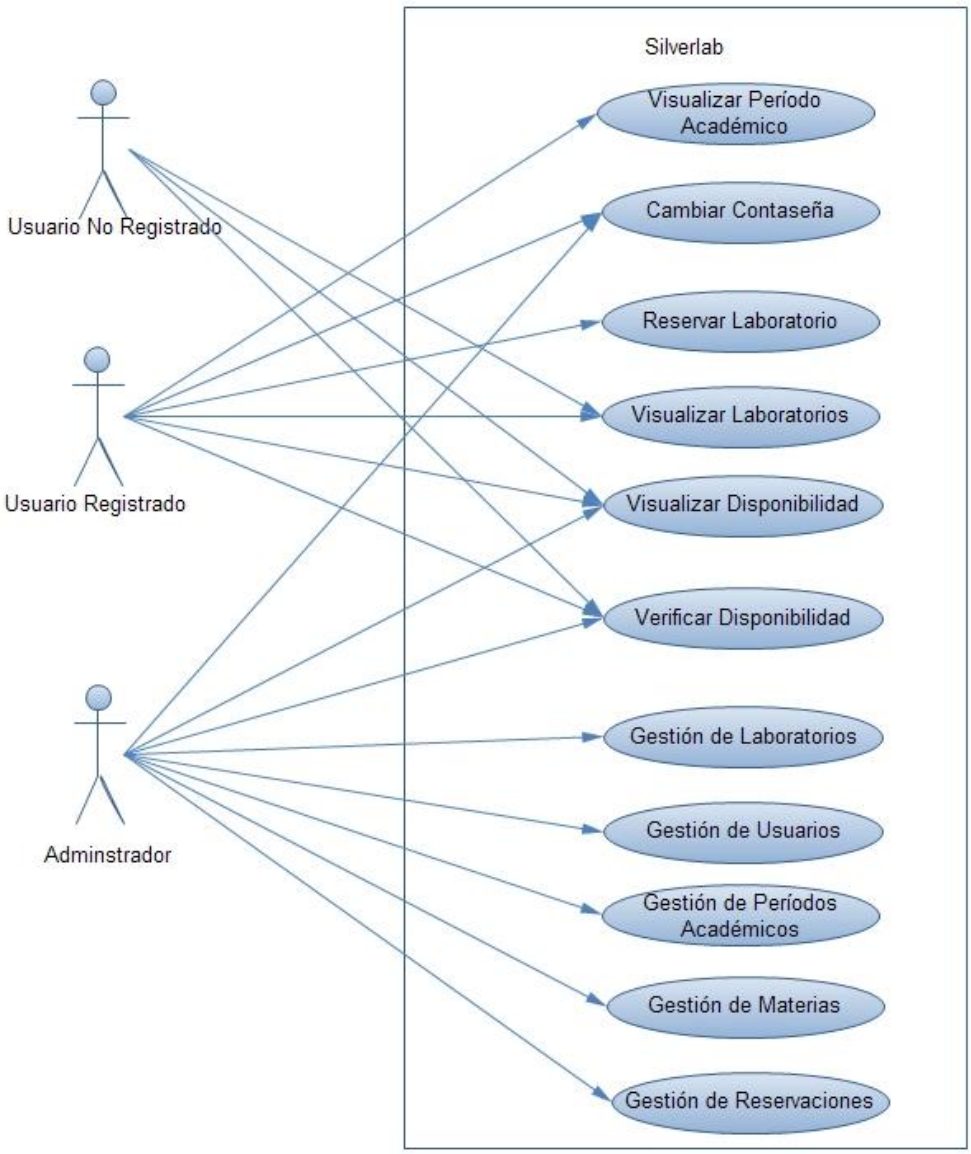


Figura 3.2: Diagrama Contextual del Usuario no registrado, Usuario registrado y Administrador

Para tener más claro el funcionamiento se mostrará a continuación cada uno de los casos de uso a un nivel más específico.

3.2.1 CASO DE USO CAMBIAR CONTRASEÑA

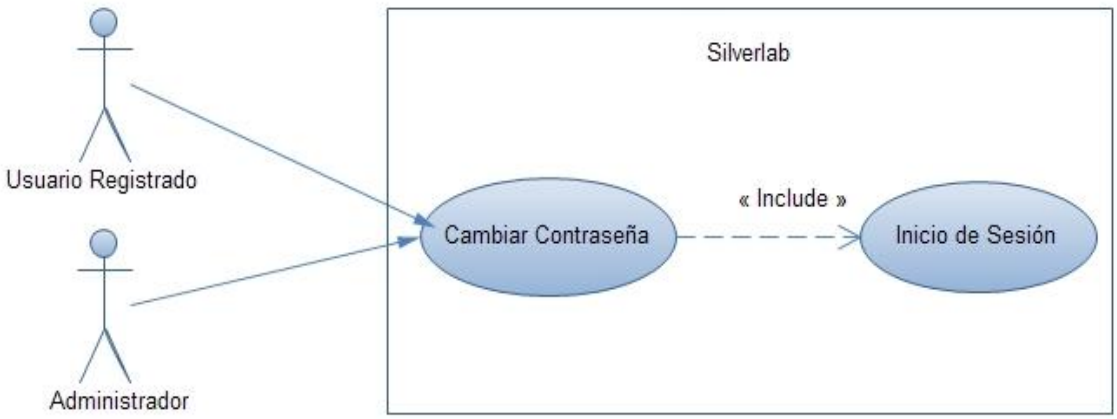


Figura 3.3: Diagrama de Caso de Uso Cambiar Contraseña

La siguiente tabla describe el caso de uso (Cambiar Contraseña), explicando la manera de interacción entre usuario y sistema.

Tabla 3.7: Descripción de Caso de Uso Cambiar Contraseña

Identificación:	1.1
Nombre:	Cambiar Contraseña
Descripción: Proceso mediante el cual se modificará la contraseña de acceso al sistema del usuario registrado o del administrador. Para la solicitud de la nueva contraseña, al usuario logueado se le pedirá una verificación de la contraseña.	
Actores: Administrador, Usuario Registrado.	

Precondiciones: Los usuarios deben estar registrados en el sistema. Los usuarios deben ingresar los datos correspondientes al nombre de usuario y contraseña. No deberán ingresar información en blanco.
Flujo Normal: • Ingresa el usuario y contraseña (Campos Obligatorios). • El sistema valida si el usuario y contraseña son correctos verificándolos en la base de datos. • El sistema despliega el menú de acuerdo al perfil asignado. • El sistema solicitará la nueva contraseña. • Al presionar el botón Actualizar se guardará la información ingresada.
Flujo Alternativo: **Validación:** Si el usuario y/o contraseña son incorrectos, o si el usuario se encuentra en estado inactivo, se presentará un mensaje de error “Usuario no registrado”.

3.2.2 CASO DE USO VERIFICAR DISPONIBILIDAD

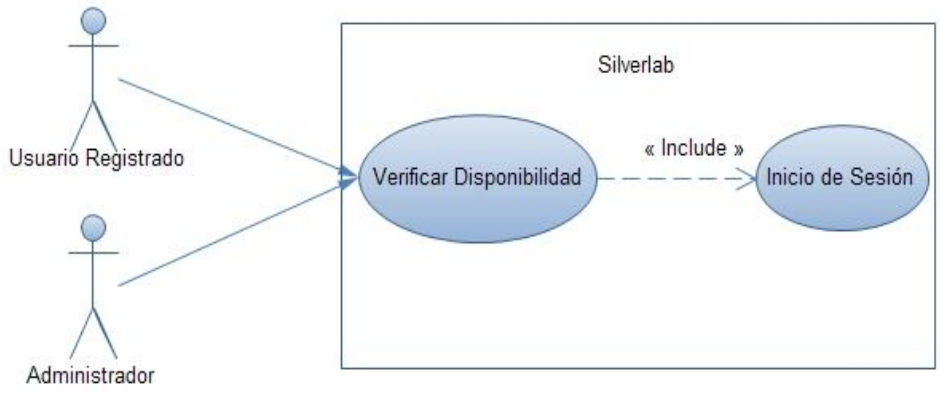


Figura 3.4: Diagrama de Caso de Uso Verificar Disponibilidad

La siguiente tabla describe el caso de uso (Verificar Disponibilidad), en la que se muestra en detalle la funcionalidad del mismo.

Tabla 3.8: Descripción de Caso de Uso Verificar Disponibilidad

Identificación:	1.2
Nombre:	Verificar Disponibilidad
Descripción: Funcionalidad en la que se debe poder verificar la disponibilidad de laboratorios en una fecha y hora específica. Dando como resultado la visualización en una tabla de los laboratorios disponibles con su respectivo identificador (ID).	
Actores: Administrador, Usuario Registrado, Usuario No Registrado.	
Precondiciones: Los usuarios deben estar registrados en el sistema. Los usuarios deben ingresar los datos correspondientes al nombre de usuario y contraseña. No deberán ingresar información en blanco.	
Flujo Normal: • Ingreso de datos correspondientes a la fecha, hora desde, hora hasta. • Validación de datos ingresados. • Se verificará en la base de datos la disponibilidad de laboratorios, dependiendo de los parámetros ingresados anteriormente. • El sistema desplegará la tabla con los resultados encontrados.	
Flujo Alternativo: **Validación:** Si los datos ingresados son en blanco, o son inconsistentes, se presentará un mensaje de error “ERROR, Verifique datos ingresados”.	

3.2.3 CASO DE USO VISUALIZAR DISPONIBILIDAD

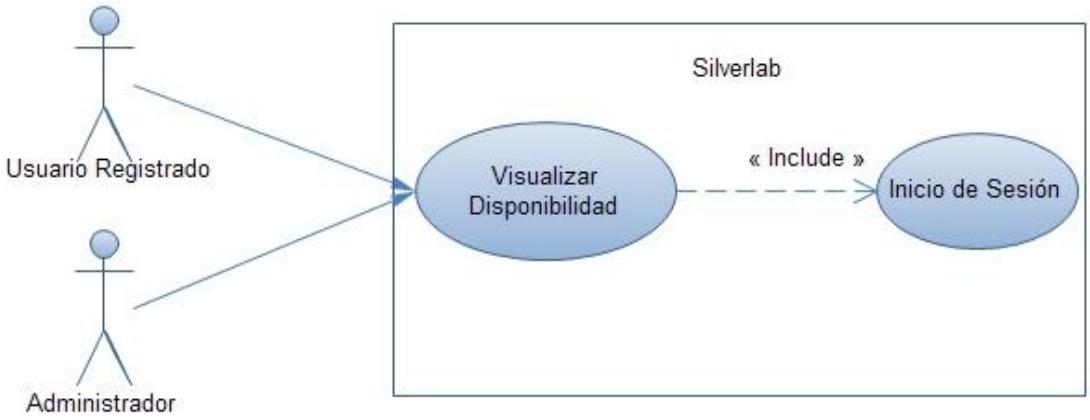


Figura 3.5: Diagrama de Caso de Uso Visualizar Disponibilidad

La siguiente tabla describe el caso de uso (Visualizar Disponibilidad), en la que se muestra en detalle la funcionalidad del mismo.

Tabla 3.9: Descripción de Caso de Uso Visualizar Disponibilidad

Identificación:	1.3
Nombre:	Visualizar Disponibilidad
Descripción: Funcionalidad en la que se debe poder visualizar la disponibilidad de determinado laboratorio en una fecha específica. Dando como resultado la visualización en una tabla mostrando datos correspondientes a la reserva en el orden (hora desde, hora hasta, NRC, Materia, Nombre de usuario).	
Actores: Administrador, Usuario Registrado, Usuario No Registrado.	
Precondiciones: Los usuarios deben estar registrados en el sistema. Los usuarios deben ingresar los datos correspondientes a la fecha y el nombre	

del laboratorio para poder consultar la disponibilidad y eventos disponibles en el día especificado.
Flujo Normal: • Ingreso de datos correspondientes a la fecha, escoger el nombre del laboratorio a buscar. • Validación de datos ingresados. • Se verificará en la base de datos la disponibilidad de laboratorios, dependiendo de los parámetros ingresados anteriormente. • El sistema desplegará la tabla con los resultados encontrados.
Flujo Alternativo: **Validación:** Si los datos ingresados son en blanco, o inconsistentes, se presentará un mensaje de error “ERROR, Verifique datos ingresados”.

3.2.4 CASO DE USO RESERVAR LABORATORIO

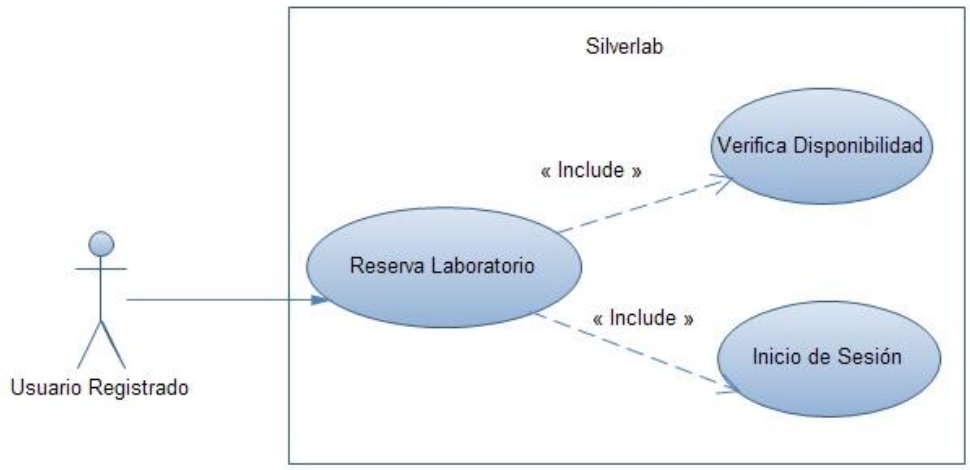


Figura 3.6: Diagrama de Caso de Uso Reservar Laboratorio

La siguiente tabla describe el caso de uso (Reservar Laboratorio), a continuación se explica la forma en que interactúa el usuario con este caso de uso.

Tabla 3.10: Descripción de Caso de Uso Reservar Laboratorio

Identificación:	1.4
Nombre:	Reservar Laboratorio
Descripción: El usuario que ingresa al sistema con perfil de acceso correspondiente a "Usuario Registrado" tendrá la opción de reservar un laboratorio en una fecha y hora específica. Para lo cual se verificará la disponibilidad del mismo para su respectiva reservación.	
Actores: Usuario Registrado	
Precondiciones: El usuario debe estar autentificado en el sistema Silverlab con el perfil de "Usuario Registrado".	
Flujo Normal: • El usuario deberá autentificarse en el sistema Silverlab. • El sistema presenta los calendarios y sus asignaciones disponibles. • El usuario registrado ingresará datos correspondientes a la reserva (fecha, hora desde, hora hasta, laboratorio, NRC). • El sistema valida la información ingresada. • El sistema verifica la disponibilidad del laboratorio con los datos ingresados. • El sistema despliega un mensaje dependiendo el caso de reservación exitosa o laboratorio no disponible.	
Flujo Alternativo: **Validación:** Los datos necesarios para la reserva son obligatorios y si la reserva se realizó con éxito despliega el mensaje "Laboratorio reservado exitosamente", y si no se encuentra disponible el laboratorio despliega el mensaje "No se han encontrado laboratorio disponible, Favor, verifique los datos ingresados".	

3.2.5 CASO DE USO GESTIÓN DE LABORATORIOS

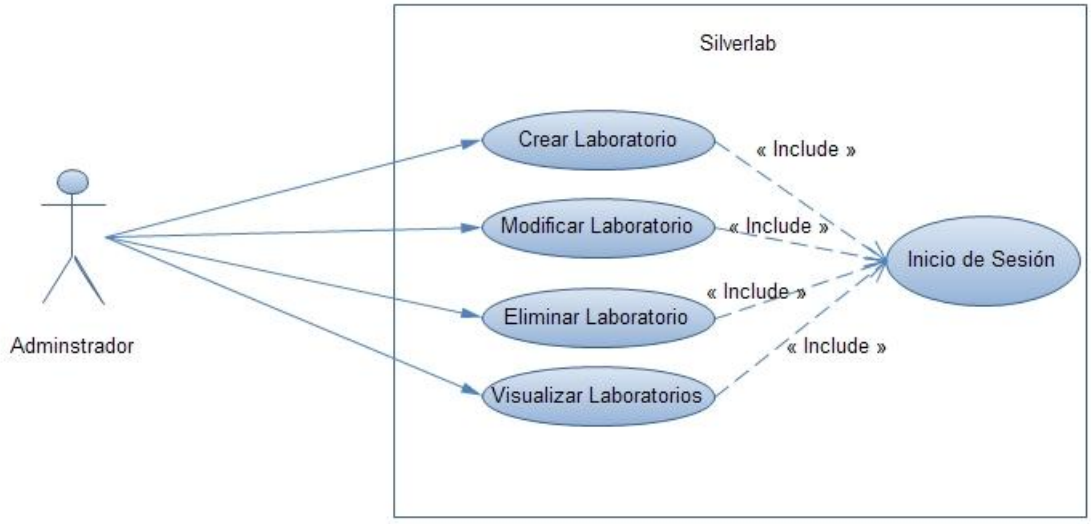


Figura 3.7: Diagrama de Caso de Uso Gestión de Laboratorios

A continuación se presentan varias tablas donde describen cada uno de los casos de uso que conforman la Gestión de Laboratorios.

Tabla 3.11: Descripción de Caso de Uso Crear Laboratorio

Identificación:	1.5
Nombre:	Crear Laboratorio
Descripción: El proceso permite ingresar datos para la creación de un nuevo laboratorio.	
Actores: Administrador	
Precondiciones: El usuario debe estar autentificado en el sistema Silverlab con el perfil de "Administrador".	

Flujo Normal:

- El usuario deberá autentificarse en el sistema Silverlab.
- El usuario con perfil de administrador será el único actor apto para poder ingresar datos correspondientes al laboratorio (nombre del laboratorio, capacidad de alumnos, número de equipos disponibles, ubicación y características).
- Se verifican los datos ingresados.
- Se guarda los datos correspondientes a un nuevo laboratorio.
- El sistema emite un mensaje de confirmación al guardar los datos.

Flujo Alternativo:

Autenticación de usuario: Al existir un problema en la autenticación de usuario el sistema mostrará el siguiente mensaje “ERROR, usuario no registrado”, si un usuario no tuviere el perfil administrador no podrá acceder a la opción de creación de laboratorios.

Ingreso de datos: Los datos tienen su respectiva validación, si no se los ingresa correctamente el sistema mostrará un mensaje de información del dato ingresado erróneamente.

Campos Requeridos: Si uno o más campos requeridos no son ingresados, el sistema mostrará un mensaje de error y le pedirá ingresar nuevamente la información.

Cambios no guardados: Si el usuario cierra la pantalla sin guardar previamente, los datos digitados se perderán.

Tabla 3.12: Descripción de Caso de Uso Modificar Laboratorio

Identificación:	1.6
Nombre:	Modificar Laboratorio
Descripción: El proceso permite modificar los datos de los diferentes laboratorios.	
Actores: Administrador	
Precondiciones: El usuario debe estar autentificado en el sistema Silverlab con el perfil de "Administrador".	
Flujo Normal: • El usuario deberá autentificarse en el sistema Silverlab. • El usuario con perfil de administrador actualizará los datos correspondientes al laboratorio (nombre del laboratorio, capacidad de alumnos, número de equipos disponibles, ubicación, características). • Se verifican los datos ingresados. • Se guarda los cambios realizados. • El sistema emite un mensaje de confirmación al grabar los datos.	
Flujo Alternativo: **Autenticación de usuario:** Al existir un problema en la autenticación de usuario el sistema mostrará el siguiente mensaje "ERROR, usuario no registrado", si un usuario no tuviere el perfil administrador no podrá acceder a la opción de modificar laboratorios **Ingreso de datos**: Los datos tienen su respectiva validación, si no se los ingresa correctamente el sistema mostrará un mensaje de información del dato ingresado erróneamente. **Campos Requeridos:** Si uno o más campos requeridos no son ingresados, el sistema mostrará un mensaje de error y le pedirá ingresar nuevamente la información. **Cambios no guardados:** Si el usuario cierra la pantalla sin guardar previamente, los datos digitados se perderán.	

Tabla 3.13: Descripción de Caso de Uso Eliminar Laboratorio

Identificación:	1.7
Nombre:	Eliminar Laboratorio
Descripción: El proceso permite la eliminación definitiva del laboratorio seleccionado.	
Actores: Administrador	
Precondiciones: El usuario debe estar autentificado en el sistema Silverlab con el perfil de "Administrador".	
Flujo Normal: • El usuario deberá autentificarse en el sistema Silverlab. • El usuario con perfil de administrador podrá eliminar el laboratorio que el desee. • Se guarda los cambios realizados. • El sistema emite un mensaje de confirmación.	
Flujo Alternativo: **Autenticación de usuario:** Al existir un problema en la autenticación de usuario el sistema mostrará el siguiente mensaje "ERROR, usuario no registrado", si un usuario no tuviere el perfil administrador no podrá acceder a la opción de eliminar laboratorios **Ingreso de datos**: Los datos tienen su respectiva validación, si no se los ingresa correctamente el sistema mostrará un mensaje de información del dato ingresado erróneamente. **Campos Requeridos:** Si uno o más campos requeridos no son ingresados el sistema mostrará un mensaje de error y le pedirá ingresar nuevamente la información. **Cambios no guardados:** Si el usuario cierra la pantalla sin guardar previamente, los datos digitados se perderán.	

Tabla 3.14: Descripción de Caso de Uso Visualizar Laboratorio

Identificación:	1.8
Nombre:	Visualizar Laboratorio
Descripción: Esta funcionalidad está abierta para cualquier tipo de usuario (Usuario no registrado, Administrador, Usuario registrado), ya que se muestra las características de los laboratorios disponibles con sus respectivos atributos (Nombre, Capacidad, Número de Equipos, Ubicación, Características).	
Actores: Administrador, Usuario Registrado, Usuario No Registrado.	
Precondiciones: Ingresar al sistema Silverlab.	
Flujo Normal: • El usuario podrá acceder al menú listar laboratorios disponibles.	
Flujo Alternativo: **Validación:** En caso de no existir laboratorios se debe comunicar con el administrador.	

3.2.6 CASO DE USO GESTIÓN DE USUARIOS

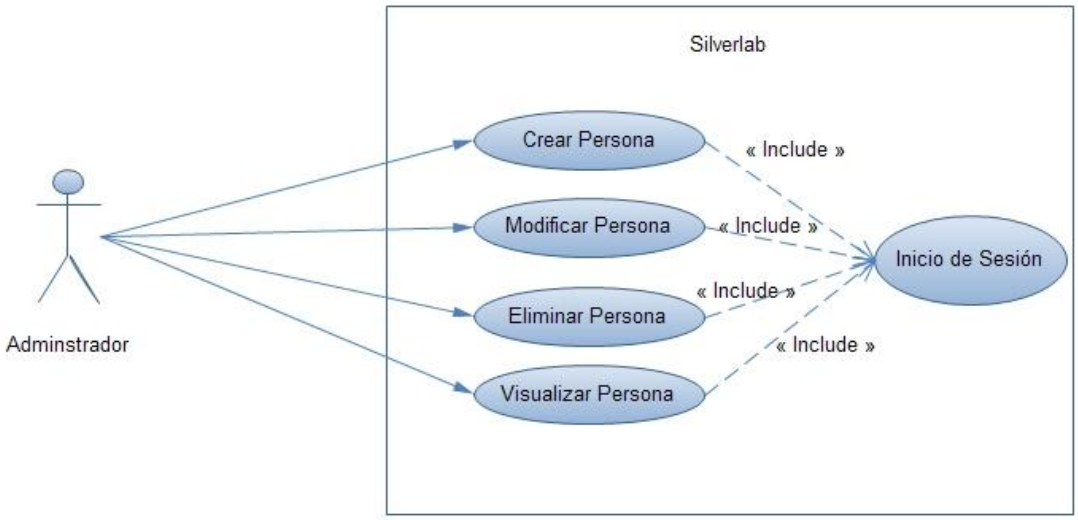


Figura 3.8: Diagrama de Caso de Uso Gestión de Usuarios

A continuación se presentan varias tablas donde describen cada uno de los casos de uso que conforman la Gestión de Usuarios.

Tabla 3.15: Descripción de Caso de Uso Crear Persona

Identificación:	1.9
Nombre:	Crear Persona
Descripción: El proceso permite ingresar datos para la creación de una nueva persona.	
Actores: Administrador	
Precondiciones: El usuario debe estar autentificado en el sistema Silverlab con el perfil de "Administrador".	

Flujo Normal:

- El usuario deberá autentificarse en el sistema Silverlab.
- El usuario con perfil de administrador será el único actor apto para poder ingresar datos correspondientes a la persona (id_banner, cedula, nombres, apellidos, teléfono, email, usuario, clave, perfil, estado, periodo académico, carreraid).
- Se verifican los datos ingresados.
- Se guarda los datos correspondientes a una nueva persona.
- El sistema emite un mensaje de confirmación al guardar los datos.

Flujo Alternativo:

Autenticación de usuario: Al existir un problema en la autenticación de usuario el sistema mostrará el siguiente mensaje “ERROR, usuario no registrado”, si un usuario no tuviere el perfil administrador no podrá acceder a la opción de creación de persona.

Ingreso de datos: Los datos tienen su respectiva validación, si no se los ingresa correctamente el sistema mostrará un mensaje de información del dato ingresado erróneamente.

Campos Requeridos: Si uno o más campos requeridos no son ingresados, el sistema mostrará un mensaje de error y le pedirá ingresar nuevamente la información.

Cambios no guardados: Si el usuario cierra la pantalla sin guardar previamente, los datos digitados se perderán.

Tabla 3.16: Descripción de Caso de Uso Modificar Persona

Identificación:	1.10
Nombre:	Modificar Persona
Descripción: El proceso permite modificar los datos de las diferentes personas registradas en la base de datos.	
Actores: Administrador	
Precondiciones: El usuario debe estar autentificado en el sistema Silverlab con el perfil de "Administrador".	
Flujo Normal: • El usuario deberá autentificarse en el sistema Silverlab. • El usuario con perfil de administrador actualizará los datos correspondientes a la persona (id_banner, cédula, nombres, teléfono, email, usuario, clave, perfil, estado, periodo académico, carreraid). • Se verifican los datos ingresados. • Se guarda los cambios realizados. • El sistema emite un mensaje de confirmación al grabar los datos.	
Flujo Alternativo: **Autenticación de usuario:** Al existir un problema en la autenticación de usuario el sistema mostrará el siguiente mensaje "ERROR, usuario no registrado", si un usuario no tuviere el perfil administrador no podrá acceder a la opción de modificar persona **Ingreso de datos**: Los datos tienen su respectiva validación, si no se los ingresa correctamente el sistema mostrará un mensaje de información del dato ingresado erróneamente. **Campos Requeridos:** Si uno o más campos requeridos no son ingresados, el sistema mostrará un mensaje de error y le pedirá ingresar nuevamente la información. **Cambios no guardados:** Si el usuario cierra la pantalla sin guardar previamente, los datos digitados se perderán.	

Tabla 3.17: Descripción de Caso de Uso Eliminar Persona

Identificación:	1.11
Nombre:	Eliminar Persona
Descripción: El proceso permite la eliminación definitiva de la persona seleccionada.	
Actores: Administrador	
Precondiciones: El usuario debe estar autentificado en el sistema Silverlab con el perfil de "Administrador".	
Flujo Normal: • El usuario deberá autentificarse en el sistema Silverlab. • El usuario con perfil de administrador podrá eliminar a la persona que el desee. • Se guarda los cambios realizados. • El sistema emite un mensaje de confirmación.	
Flujo Alternativo: **Autenticación de usuario:** Al existir un problema en la autenticación de usuario el sistema mostrará el siguiente mensaje "ERROR, usuario no registrado", si un usuario no tuviere el perfil administrador no podrá acceder a la opción de eliminar persona **Ingreso de datos**: Los datos tienen su respectiva validación, si no se los ingresa correctamente el sistema mostrará mensaje de información del dato ingresado erróneamente. **Campos Requeridos:** Si uno o más campos requeridos no son ingresados el sistema mostrará un mensaje de error y le pedirá ingresar nuevamente la información. **Cambios no guardados:** Si el usuario cierra la pantalla sin guardar previamente, los datos digitados se perderán.	

Tabla 3.18: Descripción de Caso de Uso Visualizar Persona

Identificación:	1.12
Nombre:	Visualizar Persona
Descripción: Esta funcionalidad está abierta para cualquier tipo de usuario (Usuario no registrado, Administrador, Usuario registrado), ya que se muestra las características de las personas que están registradas en el sistema con sus respectivos atributos (id_banner, cedula, nombres, apellidos, teléfono, email, usuario, clave, perfil, estado, periodo académico, carreraid).	
Actores: Administrador.	
Precondiciones: Ingresar al sistema Silverlab.	
Flujo Normal: • El usuario podrá acceder al menú de visualización de persona registrada.	
Flujo Alternativo: **Validación:** En caso de no existir ninguna persona se debe comunicar con el administrador.	

3.2.7 CASO DE USO GESTIÓN DE PERÍODOS ACADÉMICOS

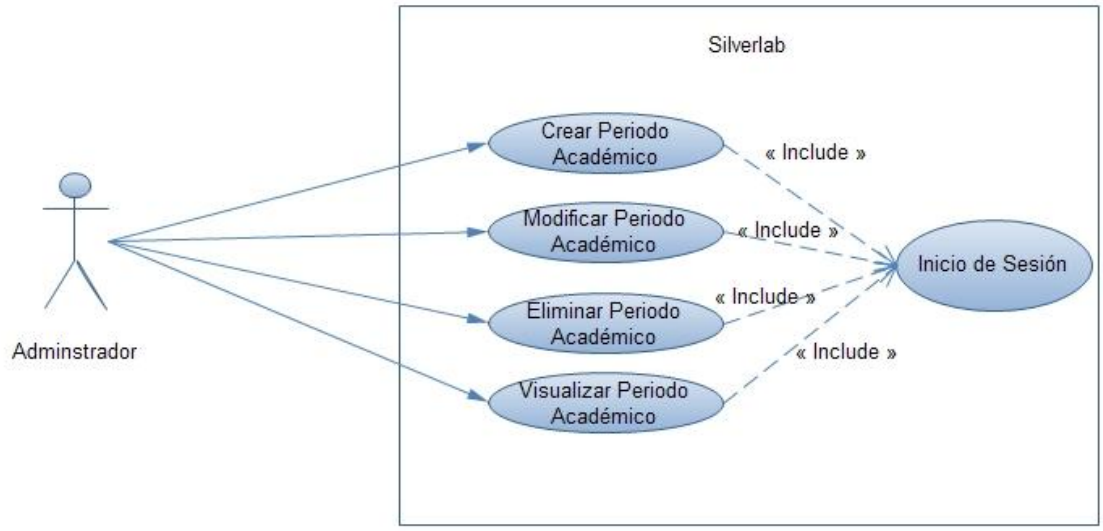


Figura 3.9: Diagrama de Caso de Uso Gestión de Períodos Académicos

A continuación se presentan varias tablas donde describen cada uno de los casos de uso que conforman la Gestión de Períodos Académicos.

Tabla 3.19: Descripción de Caso de Uso Crear Período Académico

Identificación:	1.13
Nombre:	Crear Período Académico
Descripción: El proceso permite ingresar datos para la creación de un nuevo período académico.	
Actores: Administrador	
Precondiciones: El usuario debe estar autentificado en el sistema Silverlab con el perfil de "Administrador".	

Flujo Normal:

- El usuario deberá autentificarse en el sistema Silverlab.
- El usuario con perfil de administrador será el único actor apto para poder ingresar datos correspondientes al período académico (fecha inicio, fecha final, estado, descripción).
- Se verifican los datos ingresados.
- Se guarda los datos correspondientes al nuevo período académico.
- El sistema emite un mensaje de confirmación al guardar los datos.

Flujo Alternativo:

Autenticación de usuario: Al existir un problema en la autenticación de usuario el sistema mostrará el siguiente mensaje “ERROR, usuario no registrado”, si un usuario no tuviere el perfil administrador no podrá acceder a la opción de creación de período académico.

Ingreso de datos: Los datos tienen su respectiva validación, si no se los ingresa correctamente el sistema mostrará un mensaje de información del dato ingresado erróneamente.

Campos Requeridos: Si uno o más campos requeridos no son ingresados, el sistema mostrará un mensaje de error y le pedirá ingresar nuevamente la información.

Cambios no guardados: Si el usuario cierra la pantalla sin guardar previamente, los datos digitados se perderán.

Tabla 3.20: Descripción de Caso de Uso Modificar Período Académico

Identificación:	1.14
Nombre:	Modificar Período Académico
Descripción: El proceso permite modificar los datos de los diferentes períodos académicos.	
Actores: Administrador	
Precondiciones: El usuario debe estar autentificado en el sistema Silverlab con el perfil de "Administrador".	
Flujo Normal: • El usuario deberá autentificarse en el sistema Silverlab. • El usuario con perfil de administrador actualizará los datos correspondientes al período académico (fecha inicio, fecha final, estado, descripción). • Se verifican los datos ingresados. • Se guarda los cambios realizados. • El sistema emite un mensaje de confirmación al grabar los datos.	
Flujo Alternativo: **Autenticación de usuario:** Al existir un problema en la autenticación de usuario el sistema mostrará el siguiente mensaje "ERROR, usuario no registrado", si un usuario no tuviere el perfil administrador no podrá acceder a la opción de modificar período académico. **Ingreso de datos**: Los datos tienen su respectiva validación, si no se los ingresa correctamente el sistema mostrará mensaje de información del dato ingresado erróneamente. **Campos Requeridos:** Si uno o más campos requeridos no son ingresados, el sistema mostrará un mensaje de error y le pedirá ingresar nuevamente la información. **Cambios no guardados:** Si el usuario cierra la pantalla sin guardar previamente, los datos digitados se perderán.	

Tabla 3.21: Descripción de Caso de Uso Eliminar Período Académico

Identificación:	1.15
Nombre:	Eliminar Período Académico
Descripción: El proceso permite la eliminación definitiva del período académico.	
Actores: Administrador	
Precondiciones: El usuario debe estar autentificado en el sistema Silverlab con el perfil de "Administrador".	
Flujo Normal: • El usuario deberá autentificarse en el sistema Silverlab. • El usuario con perfil de administrador podrá eliminar el período académico que desee. • Se guarda los cambios realizados. • El sistema emite un mensaje de confirmación.	
Flujo Alternativo: **Autenticación de usuario:** Al existir un problema en la autenticación de usuario el sistema mostrará el siguiente mensaje "ERROR, usuario no registrado", si un usuario no tuviere el perfil administrador no podrá acceder a la opción de eliminar período académico. **Ingreso de datos**: Los datos tienen su respectiva validación, si no se los ingresa correctamente el sistema mostrará un mensaje de información del dato ingresado erróneamente. **Campos Requeridos:** Si uno o más campos requeridos no son ingresados el sistema mostrará un mensaje de error y le pedirá ingresar nuevamente la información. **Cambios no guardados:** Si el usuario cierra la pantalla sin guardar previamente, los datos digitados se perderán.	

Tabla 3.22: Descripción de Caso de Uso Visualizar Período Académico

Identificación:	1.16
Nombre:	Visualizar Período Académico
Descripción: Esta funcionalidad está abierta para cualquier tipo de usuario (Administrador, Usuario registrado), ya que se muestra los diferentes períodos académicos registrados en la base de datos.	
Actores: Administrador, Usuario Registrado.	
Precondiciones: Ingresar al sistema Silverlab.	
Flujo Normal: • El usuario podrá acceder al menú de visualización de período académico.	
Flujo Alternativo: **Validación:** En caso de no existir ningún período académico se debe comunicar con el administrador.	

3.2.8 CASO DE USO GESTIÓN DE MATERIAS

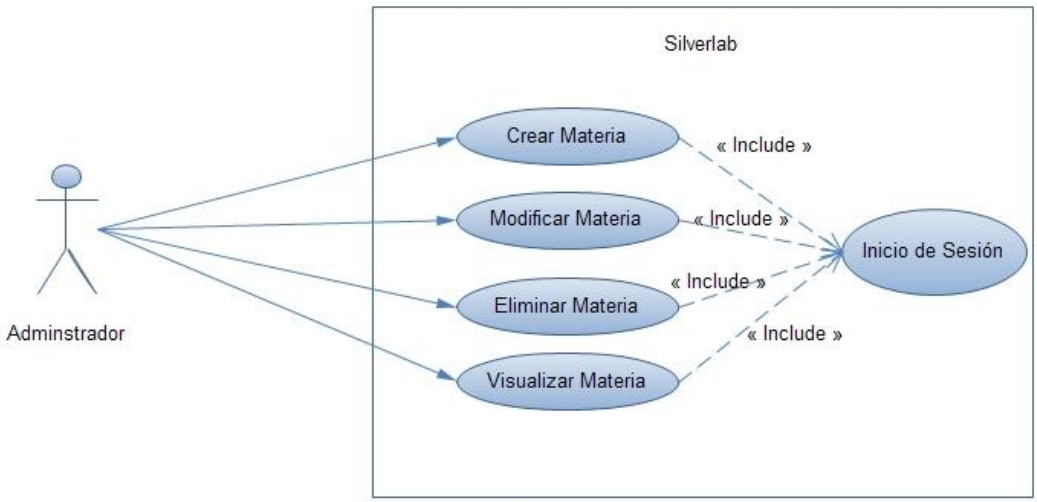


Figura 3.10: Diagrama de Caso de Uso Gestión de Materias

A continuación se presentan varias tablas donde describen cada uno de los casos de uso que conforman la Gestión de Materias.

Tabla 3.23: Descripción de Caso de Uso Crear Materia

Identificación:	1.17
Nombre:	Crear Materia
Descripción: El proceso permite ingresar datos para la creación de una nueva materia.	
Actores: Administrador	
Precondiciones: El usuario debe estar autentificado en el sistema Silverlab con el perfil de "Administrador".	

Flujo Normal:

- El usuario deberá autentificarse en el sistema Silverlab.
- El usuario con perfil de administrador será el único actor apto para poder ingresar datos correspondientes a la materia (NRC, detalle, persona id).
- Se verifican los datos ingresados.
- Se guarda los datos correspondientes a la nueva materia.
- El sistema emite un mensaje de confirmación al guardar los datos.

Flujo Alternativo:

Autenticación de usuario: Al existir un problema en la autenticación de usuario el sistema mostrará el siguiente mensaje “ERROR, usuario no registrado”, si un usuario no tuviere el perfil administrador no podrá acceder a la opción de creación de materia.

Ingreso de datos: Los datos tienen su respectiva validación, si no se los ingresa correctamente el sistema mostrará un mensaje de información del dato ingresado erróneamente.

Campos Requeridos: Si uno o más campos requeridos no son ingresados, el sistema mostrará un mensaje de error y le pedirá ingresar nuevamente la información.

Cambios no guardados: Si el usuario cierra la pantalla sin guardar previamente, los datos digitados se perderán.

Tabla 3.24: Descripción de Caso de Uso Modificar Materia

Identificación:	1.18
Nombre:	Modificar Materia
Descripción: El proceso permite modificar los datos de las diferentes materias.	
Actores: Administrador	
Precondiciones: El usuario debe estar autentificado en el sistema Silverlab con el perfil de "Administrador".	
Flujo Normal: • El usuario deberá autentificarse en el sistema Silverlab. • El usuario con perfil de administrador actualizará los datos correspondientes a la materia (NRC, detalle, persona id). • Se verifican los datos ingresados. • Se guarda los cambios realizados. • El sistema emite un mensaje de confirmación al grabar los datos.	
Flujo Alternativo: **Autenticación de usuario:** Al existir un problema en la autenticación de usuario el sistema mostrará el siguiente mensaje "ERROR, usuario no registrado", si un usuario no tuviere el perfil administrador no podrá acceder a la opción de modificar materia. **Ingreso de datos**: Los datos tienen su respectiva validación, si no se los ingresa correctamente el sistema mostrará un mensaje de información del dato ingresado erróneamente. **Campos Requeridos:** Si uno o más campos requeridos no son ingresados, el sistema mostrará un mensaje de error y le pedirá ingresar nuevamente la información. **Cambios no guardados:** Si el usuario cierra la pantalla sin guardar previamente, los datos digitados se perderán.	

Tabla 3.25: Descripción de Caso de Uso Eliminar Materia

Identificación:	1.19
Nombre:	Eliminar Materia
Descripción: El proceso permite la eliminación definitiva de la materia.	
Actores: Administrador	
Precondiciones: El usuario debe estar autentificado en el sistema Silverlab con el perfil de "Administrador".	
Flujo Normal: • El usuario deberá autentificarse en el sistema Silverlab. • El usuario con perfil de administrador podrá eliminar la materia que el desee. • Se guarda los cambios realizados. • El sistema emite un mensaje de confirmación.	
Flujo Alternativo: **Autenticación de usuario:** Al existir un problema en la autenticación de usuario el sistema mostrará el siguiente mensaje "ERROR, usuario no registrado", si un usuario no tuviere el perfil administrador no podrá acceder a la opción de eliminar materia. **Ingreso de datos**: Los datos tienen su respectiva validación, si no se los ingresa correctamente el sistema mostrará un mensaje de información del dato ingresado erróneamente. **Campos Requeridos:** Si uno o más campos requeridos no son ingresados el sistema mostrará un mensaje de error y le pedirá ingresar nuevamente la información. **Cambios no guardados:** Si el usuario cierra la pantalla sin guardar previamente, los datos digitados se perderán.	

Tabla 3.26: Descripción de Caso de Uso Visualizar Materia

Identificación:	1.20
Nombre:	Visualizar Materia
Descripción: Se muestra las diferentes materias registradas en la base de datos.	
Actores: Administrador.	
Precondiciones: Ingresar al sistema Silverlab.	
Flujo Normal: • El usuario podrá acceder al menú de visualización de materia.	
Flujo Alternativo: **Validación:** En caso de no existir ninguna materia se debe comunicar con el administrador.	

3.2.9 CASO DE USO GESTIÓN DE RESERVACIONES

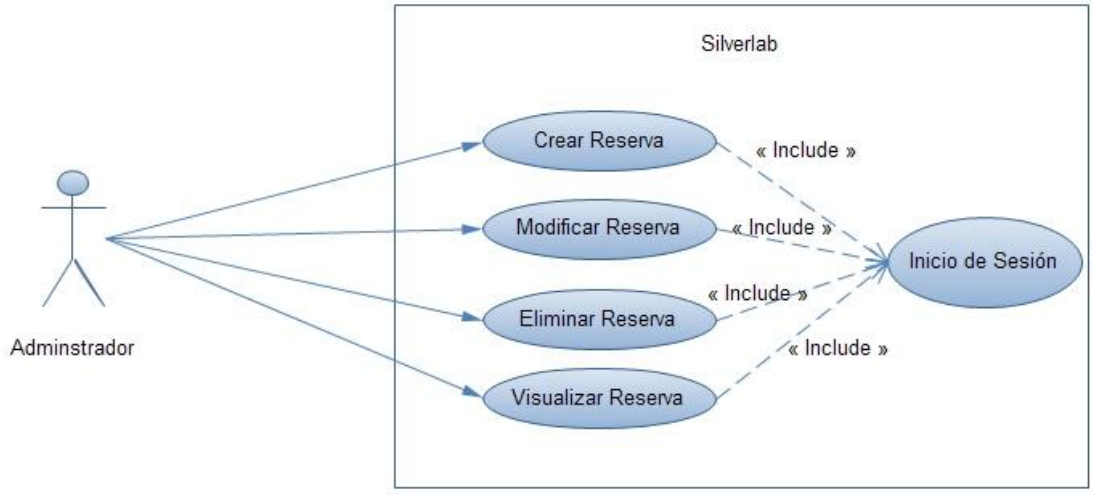


Figura 3.11: Diagrama de Caso de Uso Gestión de Reservas

A continuación se presentan varias tablas donde describen cada uno de los casos de uso que conforman la Gestión de Reservas.

Tabla 3.27: Descripción de Caso de Uso Crear Reserva

Identificación:	1.21
Nombre:	Crear Reserva
Descripción: El proceso permite ingresar datos para la creación de una nueva reserva.	
Actores: Usuario Registrado.	
Precondiciones: El usuario debe estar autentificado en el sistema Silverlab con el perfil de "Administrador".	
Flujo Normal: • El usuario deberá autentificarse en el sistema Silverlab. • El usuario con perfil de administrador será el único actor apto para poder ingresar datos correspondientes a la reserva (fecha, día, hora desde, hora hasta, laboratorio id, persona id, periodo académico id, materia id, estado, observación). • Se verifican los datos ingresados. • Se guarda los datos correspondientes a la nueva reserva. • El sistema emite un mensaje de confirmación al guardar los datos.	

Flujo Alternativo: **Autenticación de usuario:** Al existir un problema en la autenticación de usuario el sistema mostrará el siguiente mensaje “ERROR, usuario no registrado”, si un usuario no tuviere el perfil administrador no podrá acceder a la opción de creación de reserva. **Ingreso de datos**: Los datos tienen su respectiva validación, si no se los ingresa correctamente el sistema mostrará un mensaje de información del dato ingresado erróneamente. **Campos Requeridos:** Si uno o más campos requeridos no son ingresados, el sistema mostrará un mensaje de error y le pedirá ingresar nuevamente la información. **Cambios no guardados:** Si el usuario cierra la pantalla sin guardar previamente, los datos digitados se perderán.

Tabla 3.28: Descripción de Caso de Uso Modificar Reserva

Identificación:	1.22
Nombre:	Modificar Reserva
Descripción: El proceso permite modificar los datos de las diferentes reservas.	
Actores: Administrador.	
Precondiciones: El usuario debe estar autentificado en el sistema Silverlab con el perfil de “Administrador”.	

Flujo Normal:

- El usuario deberá autentificarse en el sistema Silverlab.
- El usuario con perfil de administrador actualizará los datos correspondientes a la reserva (fecha, día, hora desde, hora hasta, laboratorio id, persona id, período académico id, materia id, estado, observación).
- Se verifican los datos ingresados.
- Se guarda los cambios realizados.
- El sistema emite un mensaje de confirmación al grabar los datos.

Flujo Alternativo:

Autenticación de usuario: Al existir un problema en la autenticación de usuario el sistema mostrará el siguiente mensaje “ERROR, usuario no registrado”, si un usuario no tuviere el perfil administrador no podrá acceder a la opción de modificar reserva.

Ingreso de datos: Los datos tienen su respectiva validación, si no se los ingresa correctamente el sistema mostrará mensaje de información del dato ingresado erróneamente.

Campos Requeridos: Si uno o más campos requeridos no son ingresados, el sistema mostrará un mensaje de error y le pedirá ingresar nuevamente la información.

Cambios no guardados: Si el usuario cierra la pantalla sin guardar previamente, los datos digitados se perderán.

Tabla 3.29: Descripción de Caso de Uso Eliminar Reserva

Identificación:	1.23
Nombre:	Eliminar Reserva
Descripción: El proceso permite la eliminación definitiva de la reserva.	
Actores: Administrador	
Precondiciones: El usuario debe estar autentificado en el sistema Silverlab con el perfil de "Administrador".	
Flujo Normal: • El usuario deberá autentificarse en el sistema Silverlab. • El usuario con perfil de administrador podrá eliminar la reserva que el desee. • Se guarda los cambios realizados. • El sistema emite un mensaje de confirmación.	
Flujo Alternativo: **Autenticación de usuario:** Al existir un problema en la autenticación de usuario el sistema mostrará el siguiente mensaje "ERROR, usuario no registrado", si un usuario no tuviere el perfil administrador no podrá acceder a la opción de eliminar reserva. **Ingreso de datos**: Los datos tienen su respectiva validación, si no se los ingresa correctamente el sistema mostrará un mensaje de información del dato ingresado erróneamente. **Campos Requeridos:** Si uno o más campos requeridos no son ingresados el sistema mostrará un mensaje de error y le pedirá ingresar nuevamente la información. **Cambios no guardados:** Si el usuario cierra la pantalla sin guardar previamente, los datos digitados se perderán.	

Tabla 3.30: Descripción de Caso de Uso Visualizar Reserva

Identificación:	1.24
Nombre:	Visualizar Reserva
Descripción: Esta funcionalidad está abierta para cualquier tipo de usuario (Usuario no registrado, Administrador, Usuario registrado), ya que se muestran las diferentes reservas.	
Actores: Administrador, Usuario Registrado, Usuario No Registrado	
Precondiciones: Ingresar al sistema Silverlab.	
Flujo Normal: • El usuario podrá acceder al menú de visualización de reserva.	
Flujo Alternativo: **Validación:** En caso de no existir ninguna reserva se debe comunicar con el administrador.	

3.2.10 CASO DE USO GESTIÓN DE CARRERAS

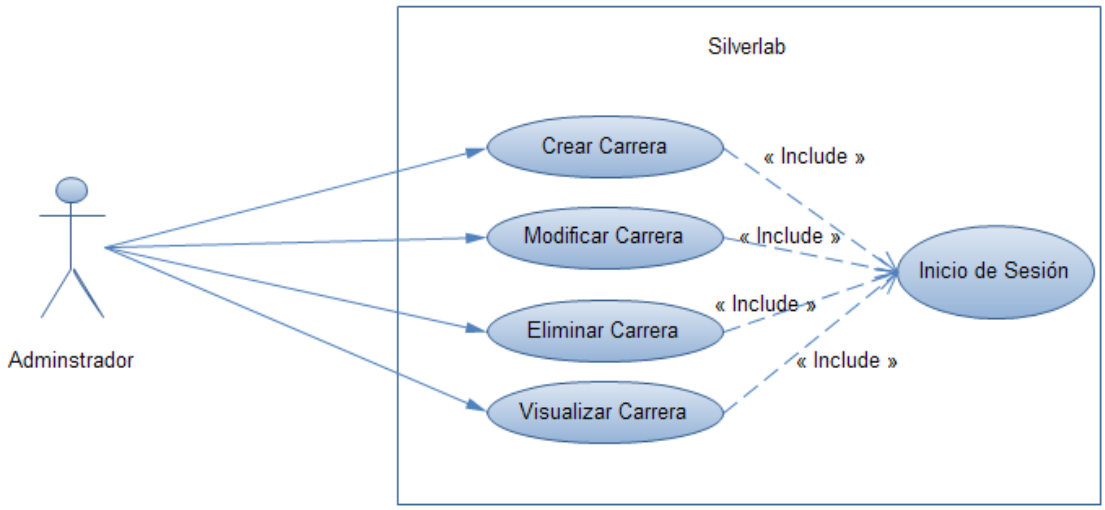


Figura 3.12: Diagrama de Caso de Uso Gestión de Carreras

A continuación se presentan varias tablas donde describen cada uno de los casos de uso que conforman la Gestión de Carreras.

Tabla 3.31: Descripción de Caso de Uso Crear Carrera

Identificación:	1.25
Nombre:	Crear Carrera
Descripción: El proceso permite ingresar datos para la creación de una nueva carrera.	
Actores: Administrador	
Precondiciones: El usuario debe estar autentificado en el sistema Silverlab con el perfil de "Administrador".	
Flujo Normal: • El usuario deberá autentificarse en el sistema Silverlab. • El usuario con perfil de administrador será el único actor apto para poder ingresar datos correspondientes a la carrera (nombre, sigla, detalle). • Se verifican los datos ingresados. • Se guarda los datos correspondientes a la nueva carrera. • El sistema emite un mensaje de confirmación al guardar los datos.	
Flujo Alternativo: **Autenticación de usuario:** Al existir un problema en la autenticación de usuario el sistema mostrará el siguiente mensaje "ERROR, usuario no registrado", si un usuario no tuviere el perfil administrador no podrá acceder a la opción de creación de carrera. **Ingreso de datos**: Los datos tienen su respectiva validación, si no se los ingresa correctamente el sistema mostrará mensaje de información del dato ingresado erróneamente. **Campos Requeridos:** Si uno o más campos requeridos no son ingresados, el sistema mostrará un mensaje de error y le pedirá ingresar nuevamente la información. **Cambios no guardados:** Si el usuario cierra la pantalla sin guardar previamente, los datos digitados se perderán.	

Tabla 3.32: Descripción de Caso de Uso Modificar Carrera

Identificación:	1.26
Nombre:	Modificar Carrera
Descripción: El proceso permite modificar los datos de las diferentes carreras.	
Actores: Administrador	
Precondiciones: El usuario debe estar autentificado en el sistema Silverlab con el perfil de "Administrador".	
Flujo Normal: • El usuario deberá autentificarse en el sistema Silverlab. • El usuario con perfil de administrador actualizará los datos correspondientes a la carrera (nombre, sigla, detalle). • Se verifican los datos ingresados. • Se guarda los cambios realizados. • El sistema emite un mensaje de confirmación al grabar los datos.	
Flujo Alternativo: **Autenticación de usuario:** Al existir un problema en la autenticación de usuario el sistema mostrará el siguiente mensaje "ERROR, usuario no registrado", si un usuario no tuviere el perfil administrador no podrá acceder a la opción de modificar carrera. **Ingreso de datos**: Los datos tienen su respectiva validación, si no se los ingresa correctamente el sistema mostrará mensaje de información del dato ingresado erróneamente. **Campos Requeridos:** Si uno o más campos requeridos no son ingresados, el sistema mostrará un mensaje de error y le pedirá ingresar nuevamente la información. **Cambios no guardados:** Si el usuario cierra la pantalla sin guardar previamente, los datos digitados se perderán.	

Tabla 3.33: Descripción de Caso de Uso Eliminar Carrera

Identificación:	1.27
Nombre:	Eliminar Carrera
Descripción: El proceso permite la eliminación definitiva de la carrera.	
Actores: Administrador	
Precondiciones: El usuario debe estar autentificado en el sistema Silverlab con el perfil de "Administrador".	
Flujo Normal: • El usuario deberá autentificarse en el sistema Silverlab. • El usuario con perfil de administrador podrá eliminar la carrera que el desee. • Se guarda los cambios realizados. • El sistema emite un mensaje de confirmación.	
Flujo Alternativo: **Autenticación de usuario:** Al existir un problema en la autenticación de usuario el sistema mostrará el siguiente mensaje "ERROR, usuario no registrado", si un usuario no tuviere el perfil administrador no podrá acceder a la opción de eliminar carrera. **Ingreso de datos**: Los datos tienen su respectiva validación, si no se los ingresa correctamente el sistema mostrará un mensaje de información del dato ingresado erróneamente. **Campos Requeridos:** Si uno o más campos requeridos no son ingresados el sistema mostrará un mensaje de error y le pedirá ingresar nuevamente la información. **Cambios no guardados:** Si el usuario cierra la pantalla sin guardar previamente, los datos digitados se perderán.	

Tabla 3.34: Descripción de Caso de Uso Visualizar Carrera

Identificación:	1.28
Nombre:	Visualizar Carrera
Descripción: Se muestran las diferentes carreras registradas en la base de datos.	
Actores: Administrador.	
Precondiciones: Ingresar al sistema Silverlab.	
Flujo Normal: • El usuario podrá acceder al menú de visualización de carrera.	
Flujo Alternativo: **Validación:** En caso de no existir ninguna reserva se debe comunicar con el administrador.	

3.3 DIAGRAMA DE CLASES

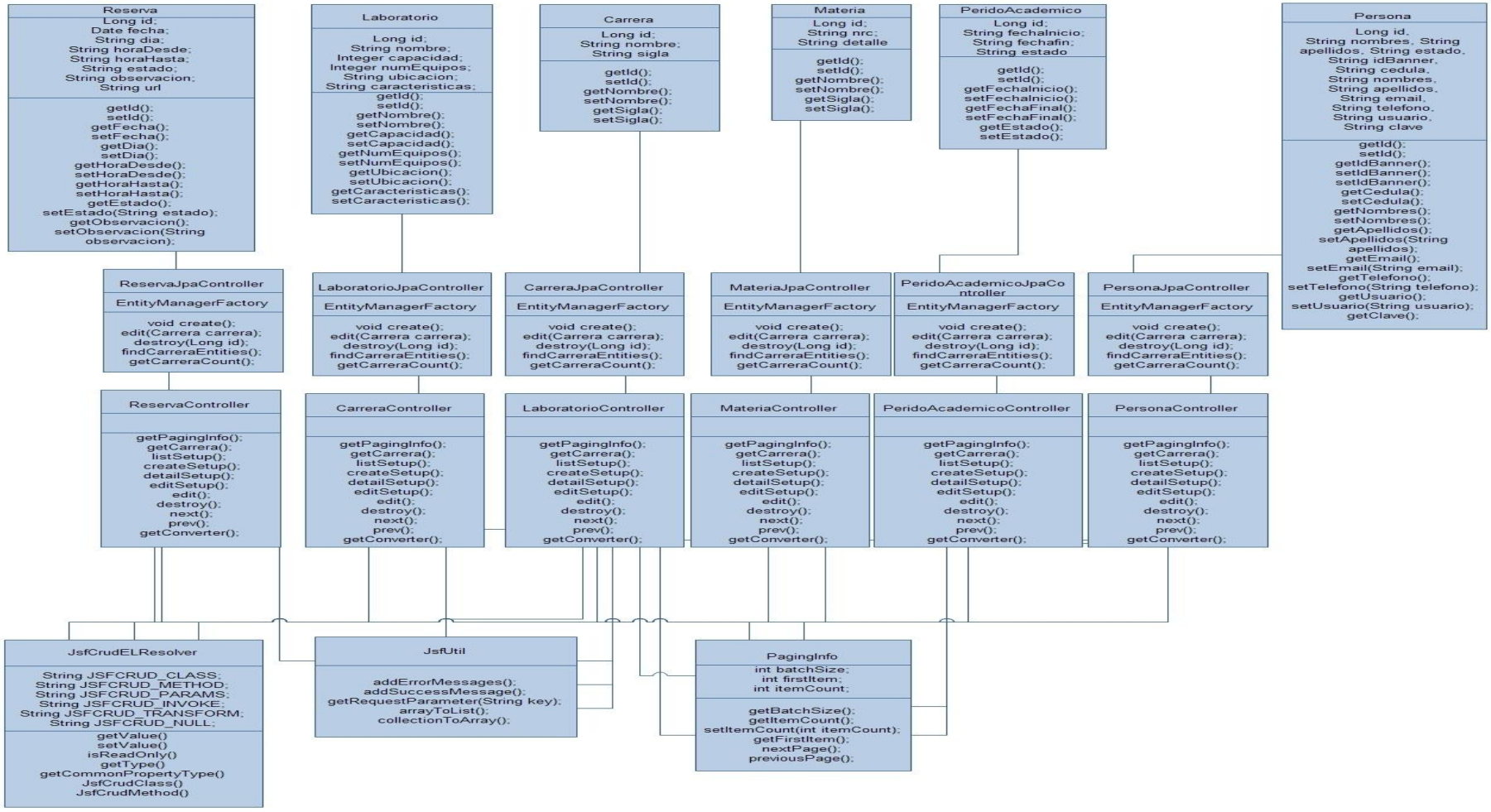


Figura 3.13: Diagrama de Clase

3.4 ESTÁNDARES DE DISEÑO

Un estándar es un conjunto de reglas normalizadas que describen los requisitos que deben ser cumplidos por un sistema, con el objetivo de establecer un mecanismo base para permitir que distintos elementos hardware o software que lo utilicen, sean compatibles entre sí.

3.4.1 ESTÁNDARES DE PROGRAMACIÓN

Tabla 3.35: Estándares de Programación

IDENTFICADOR	ESTÁNDAR	EJEMPLO
Clases	Para los nombres de clases por lo general serán sustantivos, identificados con la primera letra mayúscula en el caso de ser una sola palabra, si el nombre de la clase contiene dos o más palabras, la primera letra de estas palabras empezarán igualmente con letra mayúscula.	Class Laboratorio; Class LaboratorioController;
Interfaces	Se aplicará el mismo estándar de las clases teniendo en cuenta que los nombres asignados por lo general son adjetivos.	interface Runnable interface Serializable;

Métodos	Se aplicará un verbo en minúsculas que describa la acción a realizar, se unirá una descripción adicional al método el cual empezará con letra mayúscula.	ingresar();
		buscarTodos();
		setCustomerName();

Variables	Los nombres de variables deben ser cortos pero significativos, expresando su futura funcionalidad, se deberán escribir con letras minúsculas la primera palabra y las siguientes con la primera mayúscula según sea el caso. Para el caso de variables temporales se utilizaran las letras i,j,k,m.	int contador; String nombreCompleto; float myAltura;
Constantes	Los nombres de las variables declaradas como constantes deberán ser escritas con letras mayúsculas con las palabras separadas por guiones.	int MIN_NUMERO = 4; int GET_THE_CPU = 1;

Comentario de inicio	Todos los archivos de código fuente deberán tener un comentario en el inicio, donde se detallará el nombre del archivo, información de la versión, derechos de autor.	/* * testLaboratorio * * *versión 2.1* * * *Juan Perez* */
Bloque de Comentario	Los bloques de comentarios se utilizarán para realizar una breve descripción de métodos, estructuras de datos, algoritmos, etc. Estos deberán ir antes de cada método.	/* * Bloque de comentario. */
Paquete	Dominio + nombre proyecto + nombre del identificador de contenedor de clases.	edu.espe.ec.silverlab.nombre_paquete

3.4.2 ESTÁNDARES DE BASE DE DATOS

Tabla 3.36: Estándares de la Base de Datos

ESTÁNDAR	DESCRIPCIÓN	EJEMPLO
Prefijo de la tabla	Los nombres asignados a las tablas se deberán escribir con letras mayúsculas. Para la descripción de dos palabras se utilizará el separador "_".	LABORATORIO PERIODO_ACADEMICO
Campos de cada tabla	Los campos se escribirán con letras mayúsculas.	CEDULA
Primary key	Los atributos correspondientes a la clave primaria empezarán con la palabra "ID".	ID(Clave primaria)
Foreing key	Los atributos correspondientes a la clave foránea empezarán con el nombre de la entidad seguido del separador "_" y la palabra "ID".	PERSONA_ID

3.5 MODELO DE DATOS

3.5.1 MODELO LÓGICO

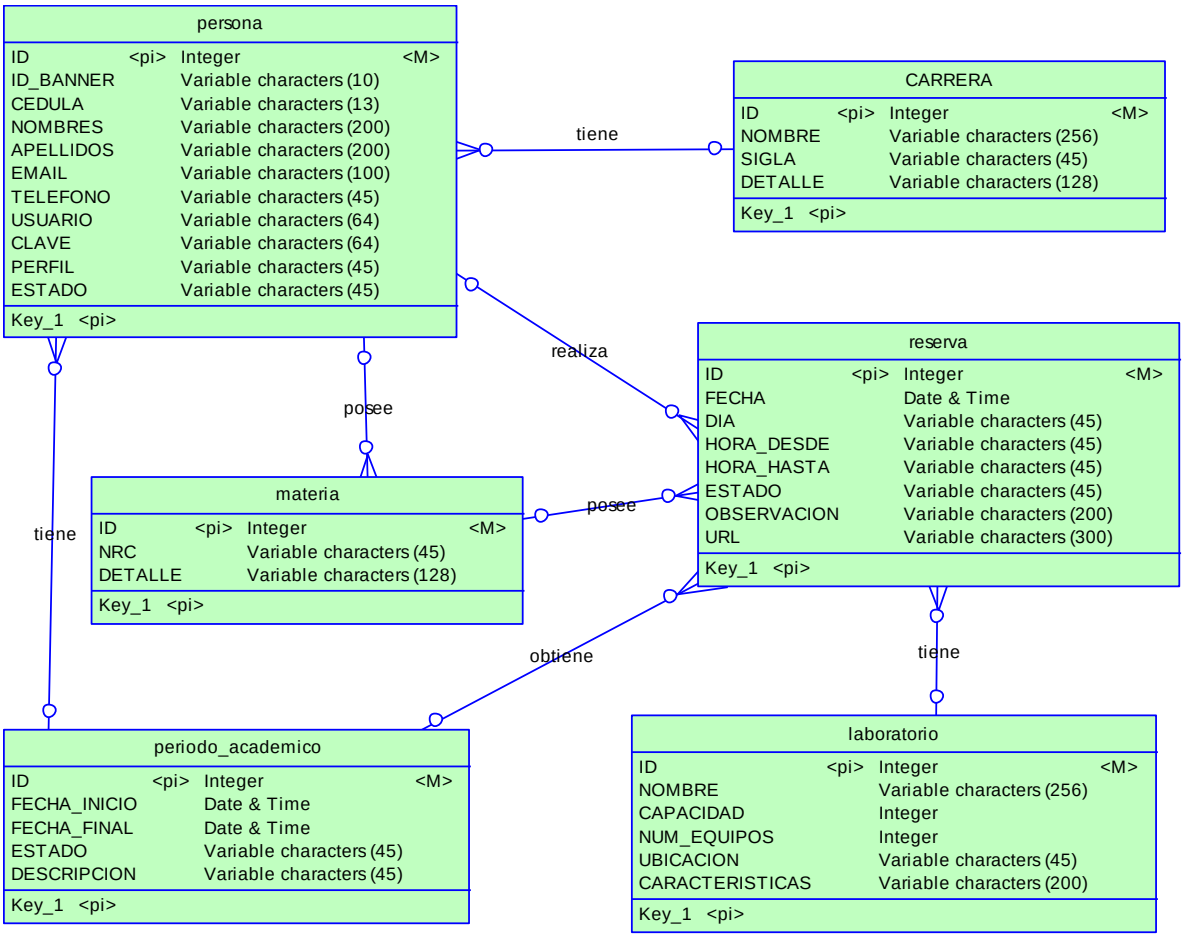


Figura 3.14: Modelo Lógico

3.5.2 MODELO FÍSICO

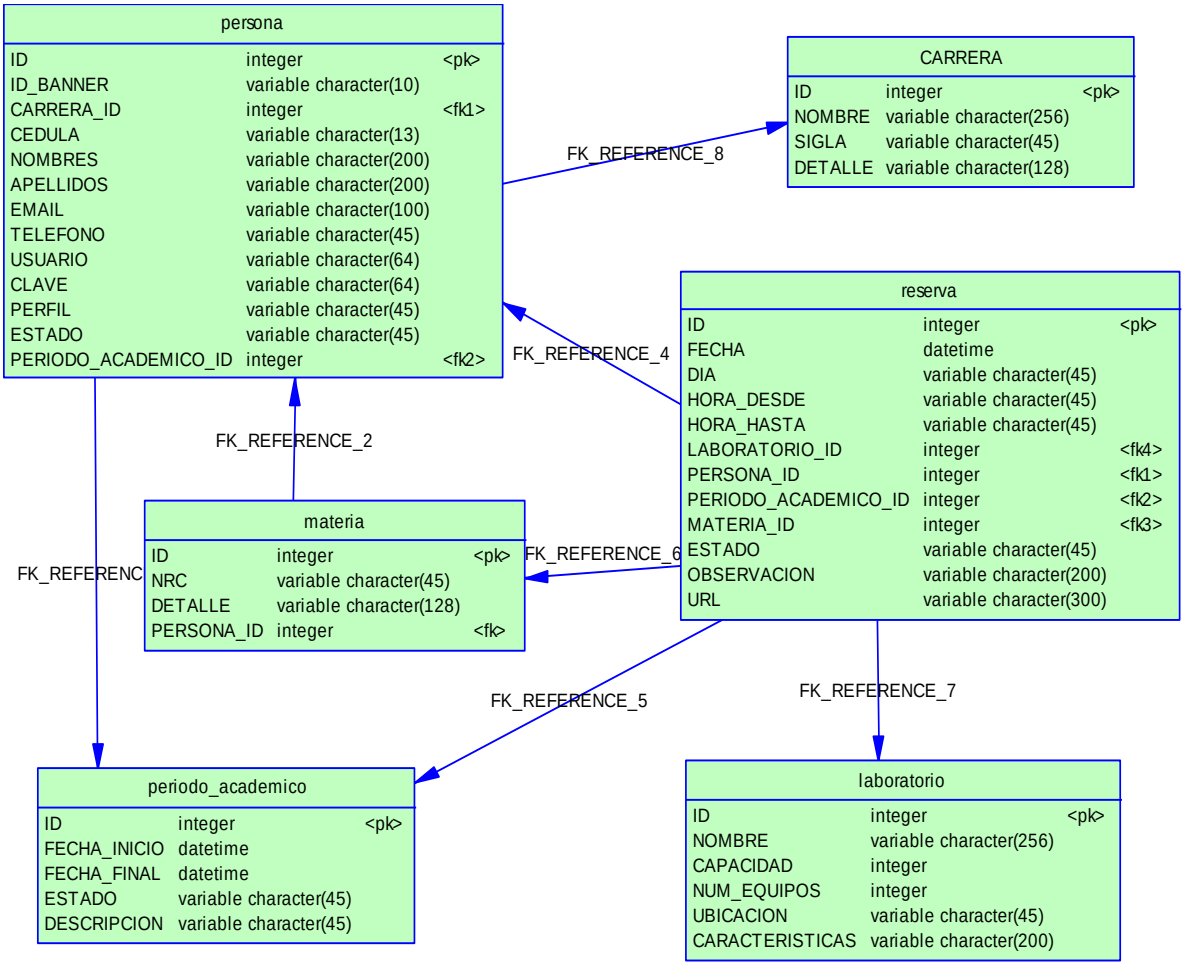


Figura 3.15: Modelo Físico

3.6 DISEÑO ARQUITECTÓNICO

Uno de los patrones más conocidos en el desarrollo web es el patrón MVC (Modelo Vista Controlador). Este patrón es el que permite/obliga a separar la lógica de control, lógica de negocio y la lógica de presentación.

3.6.1 ARQUITECTURA LÓGICA

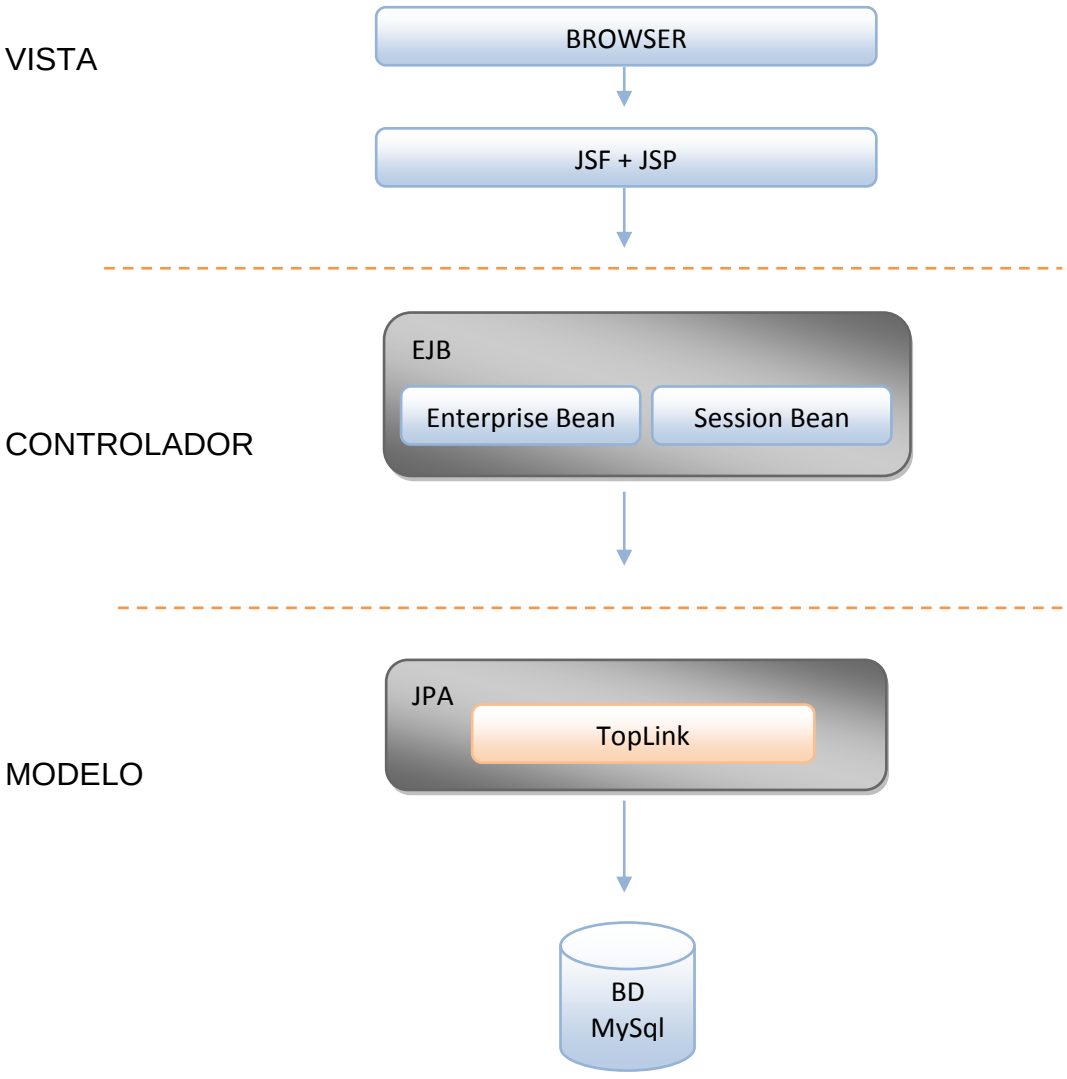


Figura 3.16: Arquitectura Lógica del Sistema Asignación de Laboratorios

3.6.2 ARQUITECTURA FÍSICA

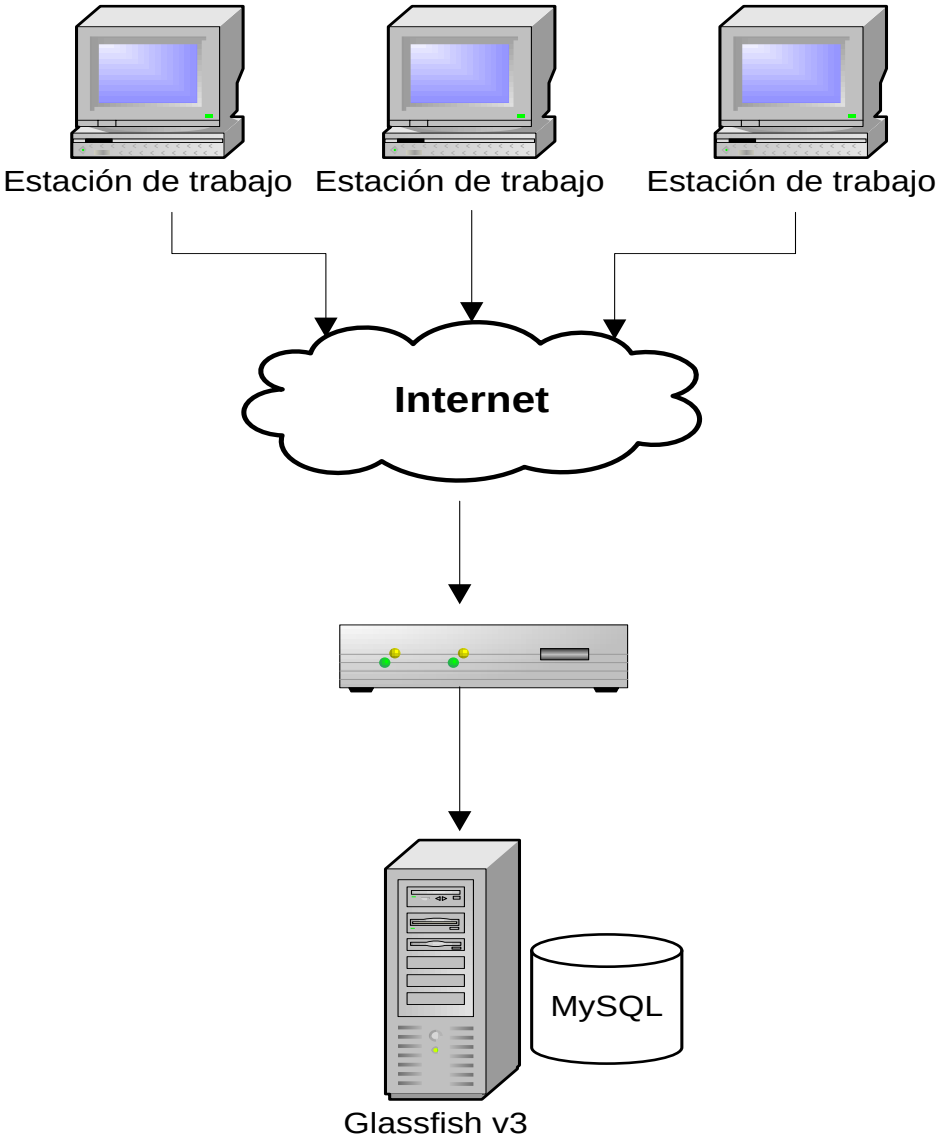


Figura 3.17: Arquitectura Física del Sistema Asignación de Laboratorios

3.7 PRUEBAS

3.7.1 PRUEBAS DE ACEPTACIÓN

RESERVA DE LABORATORIOS

¿Qué pasaría si el usuario intenta reservar un laboratorio que ya ha sido asignado?

El sistema no permitirá el registro de la reserva cuando los datos correspondientes a la fecha, hora y laboratorio se encuentran asignados a otro usuario, mostrando el siguiente mensaje “Laboratorio no disponible, Favor, verifique los datos ingresados”. Por lo que es recomendable verificar la disponibilidad de laboratorios.

¿Qué pasaría si el usuario intenta ingresar datos en blanco al realizar una reserva?

El usuario deberá ingresar datos en los campos obligatorios marcados con *, caso contrario desplegará un mensaje de “Error, Verificar campos obligatorios”.

¿Cómo puedo realizar reservas de laboratorio que aplique para todo el período académico?

Existe un control (checkbox) en el cual se debe seleccionar cuando se trate de reservas para todo el período académico. Si son reservas para una fecha específica es omitirá la selección del mismo.

CAMBIAR CONTRASEÑA

¿Qué pasaría si se ingresa datos en blanco en el campo de la nueva contraseña?

El sistema mostrará el siguiente mensaje "ERROR, Verifique contraseña ingresada, mínimo 6 caractéres".

¿Cuál es el mínimo de caractéres que el sistema acepta en el campo contraseña?

El sistema está validado para que se ingrese un mínimo de seis caracteres, si el usuario intenta ingresar un número de caractéres menor al establecido mostrará el mensaje "ERROR, Verifique contraseña ingresada, mínimo 6 caractéres".

3.7.2 PRUEBAS DEL SISTEMA

Tabla 3.37: Pruebas del Sistema

ESTADO	DESCRIPCIÓN DEL TEST	RESULTADO
Pass	El administrador de entidad no puede incluirse en una aplicación que utilice un modelo multisubproceso porque no ofrece integridad de subprocesos.	Para [Silverlab] No se ha informado de ningún error.
Pass	Un contexto de persistencia ampliado sólo se puede iniciar en el ámbito de un Stateful Session Bean.	Para [Silverlab] No se ha informado de ningún error.
Pass	El tipo de icono contiene elementos de icono pequeño e icono grande que especifican los nombres de archivo para las imágenes de iconos GIF, JPEG, o PNG grandes y pequeñas utilizadas para representar el elemento principal en la herramienta GUI. GIF, JPEG se admiten hasta J2EE 1.4; el tipo PNG se ha introducido en JAVA	Para [Silverlab] No se ha detectado ningún error.

	EE 5. Los archivos especificados deberían incluirse en el paquete que contiene este descriptor de implementación.	
Pass	El subelemento de nombre del elemento de etiqueta define un nombre de acción exclusivo.	Para [Silverlab] No se ha especificado ningún archivo de biblioteca de etiquetas
Pass	La clase de etiqueta implementa javax.servlet.jsp.tagext.JspTag para JSP versión 2.0, javax.servlet.jsp.tagext.Tag para versiones anteriores de la especificación JSP.	Para [Silverlab] No se ha especificado ningún archivo de biblioteca de etiquetas
Pass	La asignación de filtro debe ser una URL correcta o un nombre de Servlet-name en la aplicación.	Para [Silverlab] Todas las asignaciones de filtro son correctas.
Pass	Las clases de aplicación contienen métodos nativos.	Para [Silverlab] No se ha informado de ningún error.
Pass	Todas las clases del archivo de almacenamiento web se pueden cargar, a excepción de las clases utilizadas en JSP. El elemento AllJSPsMustBeCompilable de prueba se encarga de la elaboración de informes de errores no portátiles de las clases JSP.	Para [Silverlab] Todas las clases se pueden cargar en [C:\Users\User\AppData\Local\Temp\exploded20110727030245\Silverlab].
Pass	Autorización de referencia de recursos. Consulte Sección #SRV.13.4 de la especificación de Java Servlet 2.5 para obtener más información.	Para [Silverlab] No se ha informado de ningún error.
Pass	El nombre de ServletParam de la aplicación Web existe.	Para [Silverlab] El nombre de ServletParam existe en la aplicación Web.
Pass	El valor de ServletParam de la aplicación Web existe.	Para [Silverlab] No se ha informado de ningún error.
Pass	El valor de parámetro de la aplicación Web existe. Consulte 3D para obtener más información.	Para [Silverlab] El valor de parámetro existe en la aplicación Web.
Pass	El nombre de parámetro de la aplicación Web existe.	Para [Silverlab] El nombre de parámetro existe en la aplicación Web.
Pass	Todos los servlet-mappings contienen url-patterns exclusivos.	Para [Silverlab] Todos los servlet-mappings contienen url-patterns exclusivos en el archivo Web [C:\Users\User\AppData\Local\Temp\exploded20110727030245\Silverlab].
Pass	url-pattern contiene un retorno de carro (CR) o salto de línea (LF). Consulte Sección #SRV.11.2 de la especificación de Java Servlet 2.5 para obtener más información.	Para [Silverlab] url-pattern [/faces/*] en [C:\Users\User\AppData\Local\Temp\exploded20110727030245\Silverlab] no contiene

		ningún retorno de carro ni salto de línea.
Pass	Un url-pattern utilizado para una coincidencia exacta no debe contener ningún asterisco (*). Consulte Sección #SRV.11.2 de la especificación de Java Servlet 2.5 para obtener más información.	Para [Silverlab] url-pattern [/faces/*] en [C:\Users\User\AppData\Local\Temp\exploded20110727030245\Silverlab] sigue las reglas establecidas en las especificaciones del Servlet.
Pass	El contenido del elemento url-pattern debe seguir las reglas establecidas en las especificaciones del Servlet. Consulte Sección #SRV.11.2 de la especificación de Java Servlet 2.5 para obtener más información.	Para [Silverlab] url-pattern [/faces/*] en [C:\Users\User\AppData\Local\Temp\exploded20110727030245\Silverlab] sigue las reglas establecidas en las especificaciones del Servlet.
Pass	El elemento Servlet session-timeout define el intervalo predeterminado de tiempo de espera de sesión expresado en minutos. Consulte Secciones #SRV.7.5 y #SRV.13.3 de la especificación de Java Servlet 2.5 para obtener más información.	Para [Silverlab] El elemento Servlet session-timeout[30] define el intervalo predeterminado de tiempo de espera de sesión expresado en minutos.
Pass	El elemento welcome-file contiene el nombre del archivo que se usa como archivo de bienvenida predeterminado. Consulte Secciones #9.10 y #SRV.13.4 de la especificación de Java Servlet 2.4 para obtener más información.	Para [Silverlab] No se ha informado de ningún error.
Pass	Servlet implementa la interfaz javax.servlet.Servlet directa o indirectamente mediante GenericServlet o HttpServlet. Consulte Sección #SRV.2.3 de la especificación de Java Servlet 2.5 para obtener más información.	Para [Silverlab] La clase de Servlet [javax.faces.webapp.FacesServlet] implementa directa o indirectamente javax.servlet.Servlet.
Pass	El descriptor de implementación de la biblioteca de etiquetas debe tener los valores esperados para PubidLiteral "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.1/1.2 //EN".	Para [Silverlab] La prueba se ha superado satisfactoriamente.
Pass	Todos los JSP que están incluidos en la aplicación Web deben ser compilables mediante un compilador JSP compatible con J2EE que no tenga ninguna función opcional o de propiedad en él.	Para [Silverlab] All JSPs are compilable.
Pass	Todos los archivos de asignación especificados mediante el elemento en persistece.xml deben presentar el formato de asignación XML estándar, tener un nombre exclusivo y permitir la carga de recursos desde la ruta de clase de la aplicación.	Para [Silverlab#WEB-INF/classes/#SilverlabPU] No se ha informado de ningún error.

Pass	Todos los nombres de archivos jar especificados mediante el elemento en persistence.xml deberían estar disponibles en la aplicación.	Para [Silverlab#WEB-INF/classes/#SilverlabPU] No se ha informado de ningún error.
Pass	Todas las clases especificadas mediante el elemento en persistence.xml deben poderse cargar.	Para [Silverlab#WEB-INF/classes/#SilverlabPU] No se ha informado de ningún error.
Pass	Una unidad de persistencia debe tener un nombre. Sólo puede definirse una unidad de persistencia con un determinado nombre en un único archivo EJB-JAR, un único archivo WAR, un único jar de cliente de aplicaciones o mediante persistence.xml en un archivo EAR.	Para [Silverlab#WEB-INF/classes/#SilverlabPU] No se ha informado de ningún error.

3.7.3 PRUEBAS UNITARIAS

El Desarrollo Dirigido por Test al basarse en la ejecución de pruebas, usa herramientas específicas para realizar pruebas unitarias, entre ellas se encuentra el framework JUnit específico para pruebas de código java. Este tipo de herramientas hacen que el esfuerzo y el trabajo en el Desarrollo Dirigido por Test se reduzcan, permitiendo que el desarrollador pueda centrarse en fragmentos de código que cumplen determinado requerimiento.

Una prueba puede estar conformada por una serie de datos, utilización y resultados, este último se compara con los datos que en realidad debería de mostrar el software para tener un conocimiento si la aplicación está cumpliendo el requerimiento específico.

Los tests de aceptación iniciales para verificación son:

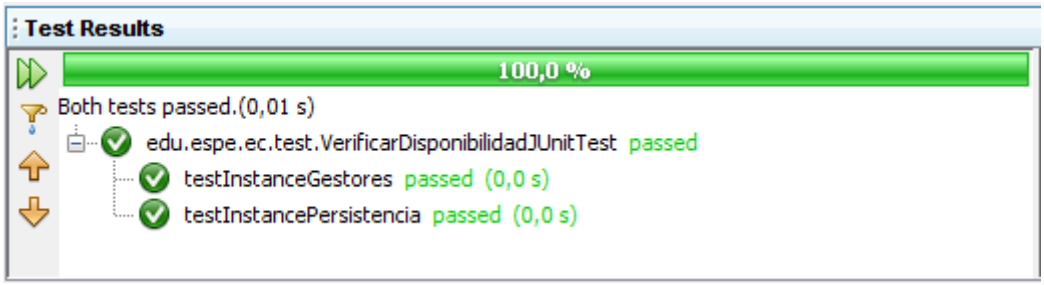
- ✓ *Al verificar la disponibilidad de un laboratorio el usuario no podrá ingresar datos en blanco.*
- ✓ *El usuario deberá ingresar información consistente. Caso contrario desplegará mensaje de error.*

3.7.3.1 VERIFICAR DISPONIBILIDAD

Test para la verificación de la correcta relación con las instancias Gestores, Persistencia.

```
@Test
   public void testInstanceGestores()
  {
       edu.espe.ec.silverlab.session.Gestores  instanciaGestores = new
Gestores();
       assertTrue(instanciaGestores instanceof Gestores):
```

```
  @Test
   public void testInstancePersistencia()
  {
       edu.espe.ec.silverlab.entities.Persistencia  instanciaPersistencia =
new Persistencia();
       assertTrue(instanciaPersistencia instanceof Persistencia);
  }
```



Verificación y validación de datos ingresados.

Test 1

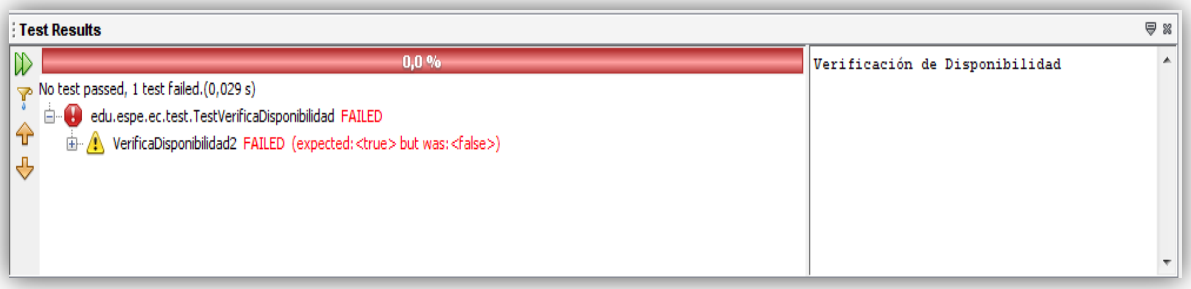
```
public void VerificaDisponibilidad2() {
System.out.println("Verificación de Disponibilidad");
edu.espe.ec.silverlab.session.Gestores  Controlador = new Gestores();
boolean expResult = true;
final String fecha = "23/12/2011";
final String hora_desde = null;
final String hora_hasta = "";
boolean result =
Controlador.verificaDisponibilidadLaboratorio(fecha,hora_desde,hora_ha
sta);
assertEquals(expResult, result);
//fail("The test case is a prototype.");
}
```

```
public boolean verificaDisponibilidadLaboratorio(String fecha, String
hdesde, String hhasta)
{
    boolean salida;
    if( isDate(fecha)  && hdesde!=null && hhasta!=null && hdesde!="" &&
hhasta!="" )
        {
            salida=true;
        }else
            salida=false;

    return salida;
}
     //metodo para validar si la fecha es correcta
    public boolean isDate(String fecha) {
        try {
            SimpleDateFormat formatoFecha = new
SimpleDateFormat("dd/MM/yyyy");
            Date fecha1 = formatoFecha.parse(fecha);
        } catch (Exception e) {
            return false;
        }
        return true;
    }
```

Resultado de la prueba

Al realizar la prueba se verifica que la misma no pasa debido a que los parámetros correspondientes a la hora_desde y hora_hasta son datos Null y en blanco respectivamente.

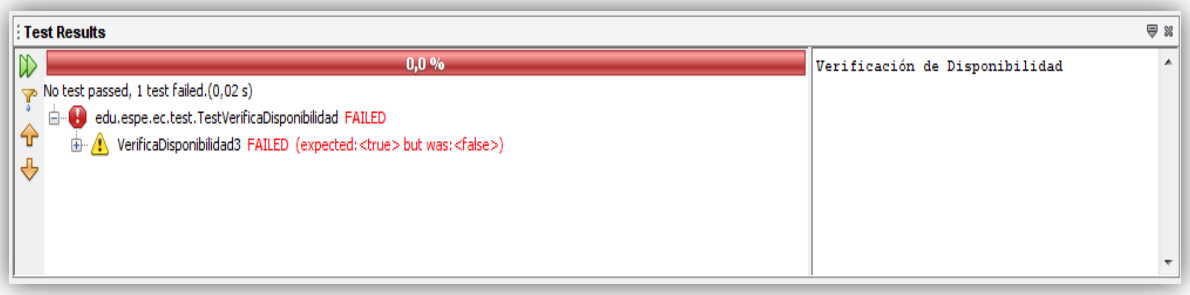


Test 2

```
   @Test
    public void VerificaDisponibilidad() {
        System.out.println("Verificación de Disponibilidad");
         edu.espe.ec.silverlab.session.Gestores  Controlador = new
Gestores();
        boolean expResult = true;
        String fecha="23/12/2011";
        String hora_desde="07:15";
        String hora_hasta= "09:15";
        boolean result =
Controlador.verificaDisponibilidadLaboratorio(fecha,hora_desde
,hora_hasta);
        assertEquals(expResult, result);
           }
```

Resultado de la prueba

Al realizar la prueba se verifica que la misma no pasa debido a que el formato del campo fecha no cumple con lo establecido en la método verificaDisponibilidadLaboratorio() .

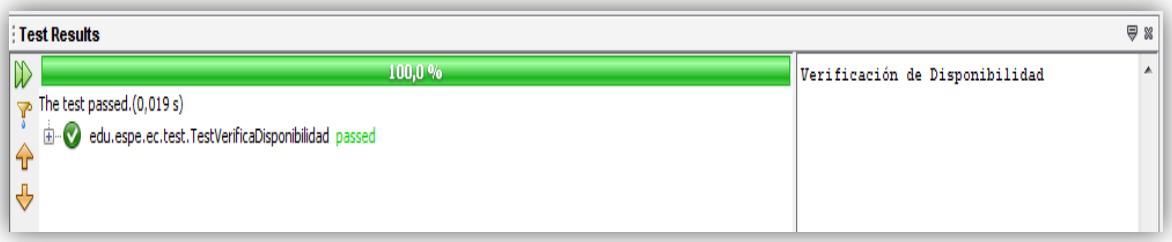


Test 3

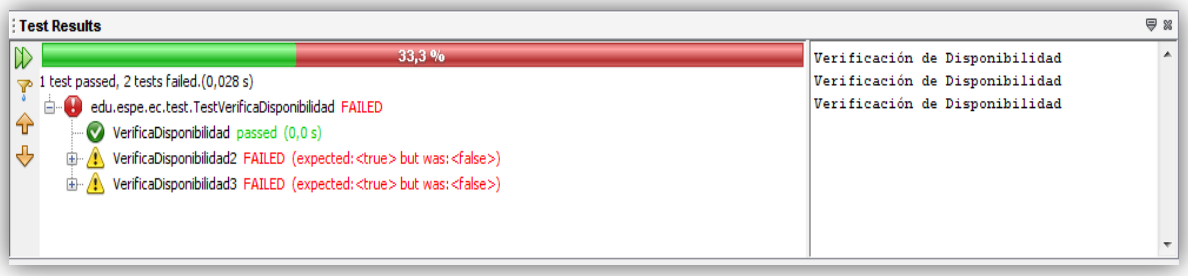
```
   @Test
    public void VerificaDisponibilidad() {
        System.out.println("Verificación de Disponibilidad");
         edu.espe.ec.silverlab.session.Gestores  Controlador = new
Gestores();
        boolean expResult = true;
        String fecha="23/12/2011";
        String hora_desde="07:15";
        String hora_hasta= "09:15";
        boolean result =
Controlador.verificaDisponibilidadLaboratorio(fecha,hora_desde
,hora_hasta);
        assertEquals(expResult, result);
    }
```

Resultado de la prueba

Al enviar datos correctos al método escrito, se puede garantizar que el caso de prueba ha pasado con éxito.



Ejecución del conjunto de pruebas



3.7.3.2 VISUALIZACIÓN DE LABORATORIOS DISPONIBLES

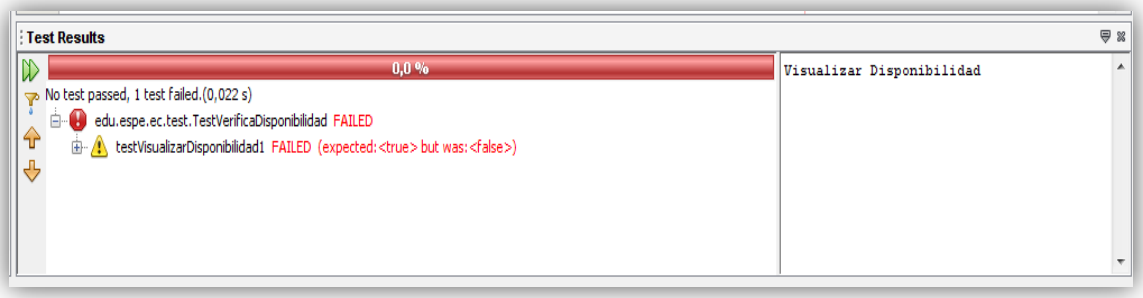
Test para validar datos para la visualización de laboratorios disponibles.

Test 1

```
@Test
    public void testVisualizarDisponibilidad1() {
        System.out.println("Visualizar Disponibilidad");
        edu.espe.ec.silverlab.session.Gestores  Controlador = new
Gestores();
        boolean expResult = true;
        assertEquals(expResult,
Controlador.imprimeReservas("23/12/2011",""));
        //fail("The test case is a prototype.");
    }
```

Resultado de la prueba

Al enviar datos correctos al método escrito, se puede garantizar que el caso de prueba ha pasado con éxito.

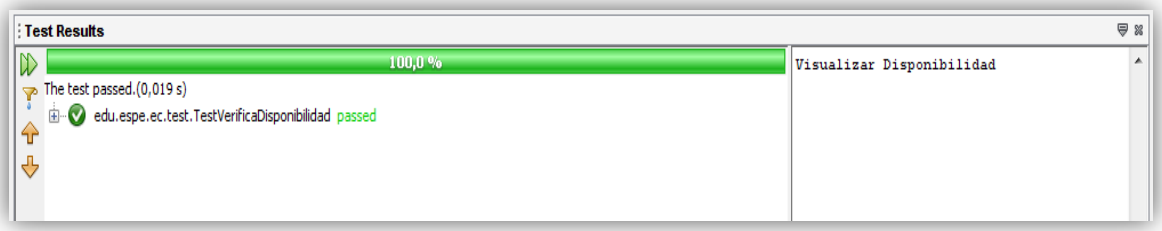


Test 2

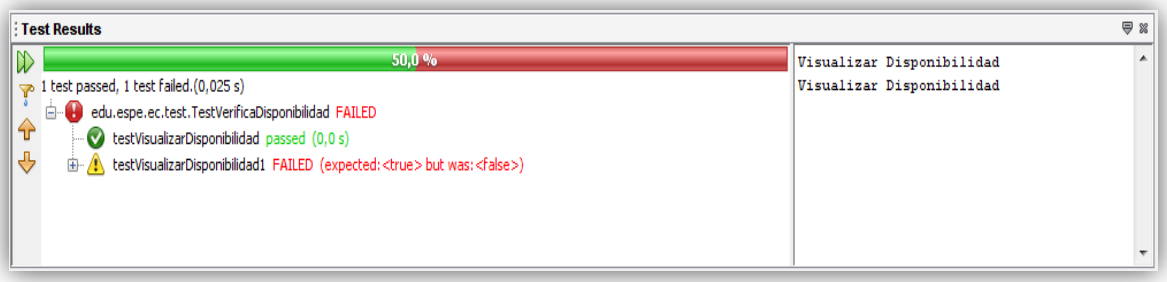
```
@Test
public void testVisualizarDisponibilidad() {
   System.out.println("Visualizar Disponibilidad");
   edu.espe.ec.silverlab.session.Gestores  Controlador = new Gestores();
   boolean expResult = true;
   assertEquals(expResult, Controlador.imprimeReservas("23/12/2011","1"));
   //fail("The test case is a prototype.");
}
```

Resultado de la prueba

Al enviar datos correctos al método escrito, se puede garantizar que el caso de prueba ha pasado con éxito.



Ejecución del conjunto de pruebas



3.7.3.3 VERIFICACIÓN DE USUARIO

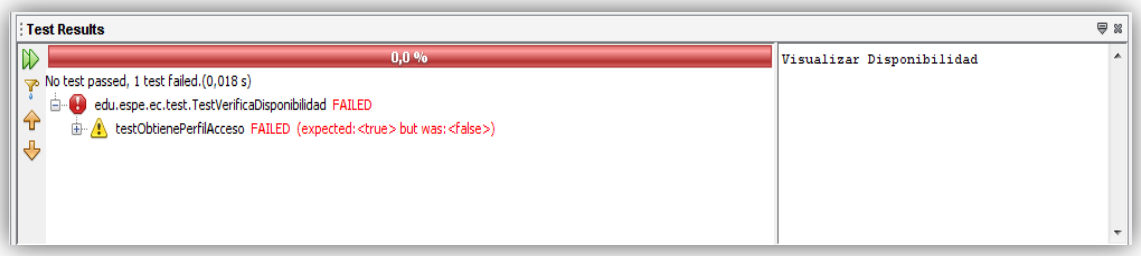
Test para validar datos para la verificación de un usuario en el sistema

Test 1

```
@Test
    public void testObtienePerfilAcceso() {
        System.out.println("Visualizar Disponibilidad");
        edu.espe.ec.silverlab.session.Gestores  Controlador = new
Gestores();
        boolean expResult = true;
        final String user = "admin";
        final String password = "";
        Boolean perfil;
         perfil = Controlador.VerificaPerfilPersona(user,password);
        assertEquals(expResult, perfil);
    }
```

Resultado de la prueba

Al enviar el parámetro password en blanco, la herramienta JUnit no permitirá pasar el test.

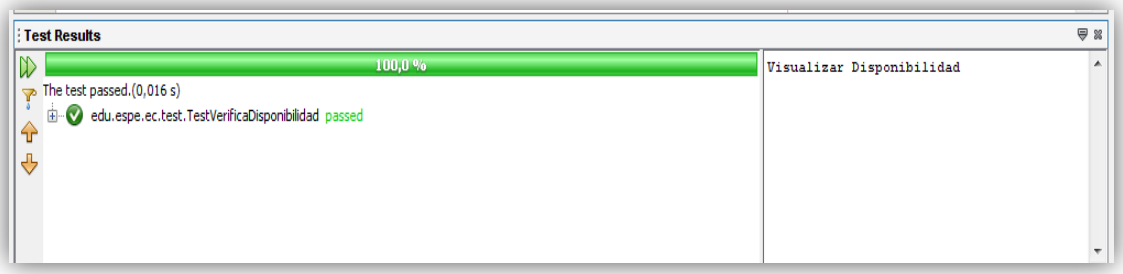


Test 2

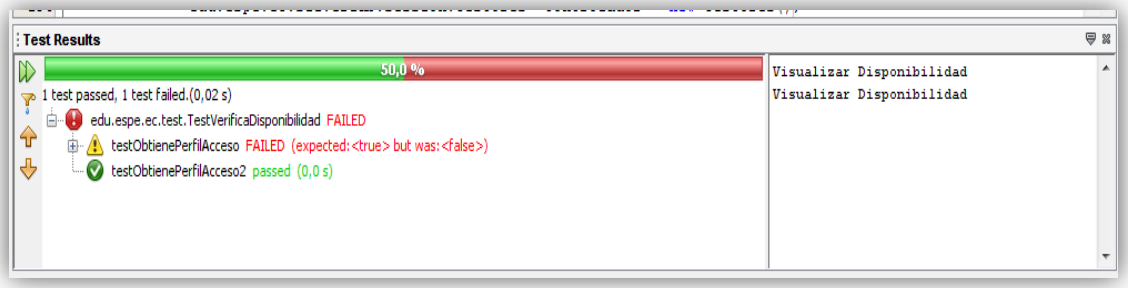
```
@Test
    public void testObtienePerfilAcceso() {
        System.out.println("Visualizar Disponibilidad");
        edu.espe.ec.silverlab.session.Gestores  Controlador = new
Gestores();
        boolean expResult = true;
        final String user = "admin";
        final String password = "";
        Boolean perfil;
         perfil = Controlador.VerificaPerfilPersona(user,password);
        assertEquals(expResult, perfil);
    }
```

Resultado de la prueba

Al enviar datos correctos al método escrito, se puede garantizar que el caso de prueba ha pasado con éxito.



Ejecución del conjunto de pruebas



CAPÍTULO IV

CONCLUSIONES Y RECOMENDACIONES

4.1 CONCLUSIONES

Tras el estudio de la técnica del Desarrollo Dirigido por Test, se ha podido constatar que la misma no es solamente una técnica que abarca el testing de la aplicación, sino que es una técnica de diseño, ya que intenta mejorar el enfoque de desarrollo obteniendo código de calidad.

Se realizó una comparación entre algunas técnicas que también han llevado la filosofía de TDD, como son Acceptance Test Driven Development (ATDD) y Behaviour Driven Development (BDD), ambas son muy similares en cuanto a que están orientados a la automatización de pruebas y generación de código, pero la diferencia radica en que TDD busca que el programador vaya más allá que implementar código, sino que se convierta en un diseñador de software.

El acoplarse al TDD no es una tarea sencilla, requiere de un largo proceso de adaptabilidad a buenas prácticas de desarrollo ágil, en principio el uso del mismo se torna complejo necesitando de mayor esfuerzo y tiempo para poder lograr resultados eficientes.

El Desarrollo Dirigido por Test (TDD) se acopla a diferentes lenguajes de programación como son Ruby, PHP, Java, Pearl, entre otros. Para la aplicación desarrollada se utilizó la plataforma de programación Java Enterprise Edition,

facilitando el desarrollo de una aplicación distribuida en niveles de programación, permitiendo ejecutar funcionalidades sencillas que se acoplan al requerimiento del usuario.

Hacer las pruebas antes de la etapa de desarrollo permitió tener mayor seguridad en la implementación de código, ya que éste trabaja en paralelo con la ejecución de pruebas, eliminando el riesgo de generar bugs que afecten a otras clases al momento de modificar código y no poder resolverlo con facilidad.

De igual manera la utilización del framework JUnit facilitó la ejecución de pruebas unitarias en la aplicación desarrollada, permitiendo tener un detalle en cuanto al fallo o éxito de las mismas, ya que si la prueba no fue exitosa se podía saber a tiempo donde estuvo el error.

Tras el proceso de refactorización y utilización de una buena semántica se logró generar un código limpio y escalable, obteniendo como resultado calidad en el producto software.

La metodología AUP aplica técnicas ágiles como el Desarrollo Dirigido por Pruebas (Test Driven Development - TDD), Modelado Ágil y Gestión de Cambios Ágil, por lo que se acopló de manera significativa en el desarrollo del caso práctico, permitiendo tener una correcta distribución de las actividades de trabajo y a la vez centrarse en actividades de alto valor, logrando una aplicación distribuida y sencilla.

Adicionalmente se utilizó la arquitectura MVC, logrando una clara separación entre interfaz, lógica de negocio y de presentación, facilitando la realización de pruebas unitarias de sus clases, obteniendo un código flexible para cambios.

La aplicación desarrollada fue publicada en el repositorio web sourceforge.net, el cual se encuentra a disposición de la comunidad con la finalidad de aportar, comentar, y continuar con el desarrollo de funcionalidades sobre la aplicación.

Cabe mencionar que cuando existen proyectos realizados sin TDD, se puede realizar test de regresión que consiste en la aplicación de test en el front-end (primera capa), con la finalidad de encontrar divergencias funcionales respecto al comportamiento esperado del software.

4.2 RECOMENDACIONES

Para obtener un mejor resultado en cuanto al uso de la técnica TDD, se recomienda:

- ✓ Trabajar conjuntamente con buenas prácticas del desarrollo ágil como por ejemplo la utilización de patrones de diseño, buen uso de semántica, integración continua, entre otros.

- ✓ Intentar crear los tests antes de la implementación, ya que si se realizan los mismos después de la implementación se está cayendo en el desarrollo tradicional, por lo que se pierde todas las ventajas que aporta usar TDD.

- ✓ Crear un test por iteración y solo implementar el mínimo código necesario para resolver ese caso. No es bueno "emocionarse" implementando y desarrollar más de lo necesario para resolver el caso de prueba, ya que si se desarrolla más casos se pierde una gran parte de la eficacia de esta técnica.

- ✓ No intentar automatizar todo el proceso de prueba, puede no ser viable ni práctico. La prueba debe ser estratégica en la búsqueda de defectos.

- ✓ Realizar un desarrollo de parejas durante la aplicación del TDD, donde el equipo debe tener un nivel promedio de experiencia en el desarrollo de software y haber aprendido de errores pasados para de esta manera obtener mejores resultados.

- ✓ Cuando se realice un desarrollo de gran dimensión es importante dividir por módulos de funcionalidad, ya que el proceso de testing se torna complejo y es necesaria la aplicación de herramientas adicionales de pruebas como Jmocks y Stubs.

En cuanto a la aplicación desarrollada, el administrador encargado deberá tener un amplio conocimiento en cuanto a su funcionalidad, para ayudar de soporte a otros usuarios.

El código estará disponible en el repositorio web de SourceForge con la finalidad de poder acceder a realizar mejoras a la aplicación, por lo que se recomienda que cualquier cambio deba ser realizado de forma transparente para el usuario final.

GLOSARIO

AJAX: Siglas de Asynchronous JavaScript and XML, es una forma de desarrollo web para crear aplicaciones interactivas. Estas aplicaciones se ejecutan en el cliente (en este caso el navegador de los usuarios), y mantiene comunicación asíncrona con el servidor en segundo plano, permitiendo realizar cambios sobre la misma página sin necesidad de recargarla.

API: Siglas de Application Development Interface, es un conjunto de recursos para la implementación de características específicas dentro de un paquete de software.

Artefactos: Es un producto tangible resultante del proceso de desarrollo de software como son los casos de uso, diagrama de clases u otros modelos, así como también el código fuente compilado ya que el ejecutable es necesario para el plan de testeo.

Entregables: Productos que se intercambian entre los clientes y los desarrolladores a lo largo de la ejecución del proyecto como la descripción detallada del sistema, diagramas, documentos de diseño, resultado de las pruebas entre otros.

Framework: Son diseñados para facilitar el desarrollo de software, permitiendo a los diseñadores y programadores pasar más tiempo identificando

requerimientos de software que tratando con esos pequeños detalles de bajo nivel de proveer un sistema funcional.

HTML: (Hyper Text Mark-up Language o Lenguaje de Marcas de Hipertexto) sirve para modelar texto y agregarle funciones especiales (por ej. Hipervínculos). Es la base para la creación de páginas web tradicionales.
El texto se modela a partir del uso de etiquetas o tags.

JUnit: Es una librería Java que permite ejecutar pruebas unitarias, con el fin de informar directamente si pasan o no satisfactoriamente las pruebas.

Just In Time: Literalmente quiere decir "Justo a tiempo". Es una filosofía que define la forma en que debería optimizarse un sistema.

NUnit: Es la alternativa a JUnit para .Net. Permite realizar pruebas unitarias para cualquier lenguaje de .Net. (C#, VB.Net, C++.Net, J#...)

PHP: (Hypertext Pre-processor) Lenguaje de programación usado generalmente en la creación de contenidos para sitios web y aplicaciones para servidores.

Prototipo: Es una representación de aquellos aspectos del software que serán visibles para el cliente o el usuario final, con el objetivo de satisfacer las necesidades del cliente. Esto permite que al mismo tiempo el desarrollador entienda mejor lo que se debe hacer y el cliente vea resultados a corto plazo.

Ruby: Es un lenguaje de programación interpretado, orientado a objetos, soporta herencia con enlace dinámico y métodos singleton (pertenecientes y definidos por una sola instancia más que definidos por la clase).

RUP: Siglas de Rational Unified Process (Proceso Racional Unificado), es un proceso de desarrollo de software, que constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos.

Script: Son un conjunto de instrucciones generalmente almacenadas en un archivo de texto que deben ser interpretados línea a línea en tiempo real para su ejecución.

XP: Siglas de Extreme Programming (Programación Extrema) es una metodología de desarrollo ligera (o ágil) basada en una serie de metodologías de desarrollo de software en la que se da prioridad a los trabajos que dan un resultado directo y que reducen la burocracia que hay alrededor de la programación.

BIBLIOGRAFÍA

Alvarez, M. A. (08 de Julio de 2002). *desarrolloweb.com Que es JSP*. Recuperado el 02 de Abril de 2011, de desarrolloweb.com Que es JSP: http://www.desarrolloweb.com/articulos/831.php

Ambler, S. W. (2005). *Disciplinas de Agile UP*. Recuperado el 15 de Mayo de 2011, de Disciplinas de Agile UP: http://cgi.una.ac.cr/AUP/html/disciplines.html

Ambler, S. W. (2005). *El Proceso Unificado Ágil*. Recuperado el 10 de Abril de 2011, de El Proceso Unificado Ágil: http://cgi.una.ac.cr/AUP/index.html

Anónimo. (s.f.). *Agile Alliance*. Recuperado el 17 de Marzo de 2011, de Agile Alliance: www.agilealliance.com

Anónimo. (s.f.). *SeleniumHQ*. Recuperado el 23 de Marzo de 2011, de SeleniumHQ: http://seleniumhq.org/

Anónimo. (s.f.). *TDD: Test Driven Development*. Recuperado el 25 de 06 de 2011, de www.chuidiang.com: http://www.chuidiang.com/java/herramientas/test-automaticos/tdd-test-driven-development.php

Anónimo. (18 de Mayo de 2006). *Wikipedia JavaServer Faces* . Recuperado el 07 de Abril de 2011, de Wikipedia JavaServer Faces : http://es.wikipedia.org/wiki/JavaServer_Faces

Barros, A. (05 de Enero de 2010). *Comportamiento de Proyectos TI*. Recuperado el 21 de Febrero de 2011, de http://www.alejandrobarros.com/content/view/691759/Comportamiento-de-proyectos-TI-Estan-en-deuda.html

Beck, K. (2002). *NUnit.org*. Recuperado el 23 de Marzo de 2011, de NUnit.org: http://www.nunit.org

Beck, K. (2003). *Test Driven Development: by Example.*

Blé, C. (Enero 2010). *DiseñoAgilConTDD.* Madrid: Creative Commons.

Canós, J. H. (2006). *Metodologías Ágiles en el Desarrollo de Software.* Recuperado el 20 de Marzo de 2011, de Metodologías Ágiles en el Desarrollo de Software: http://www.willydev.net/descargas/prev/TodoAgil.Pdf

Cantero, J. (2004). *Wikipedia Java EE*. Recuperado el 02 de Abril de 2011, de Wikipedia Java EE: http://es.wikipedia.org/wiki/Java_EE

Carlos Iglesias, J. J. (07 de 11 de 2005). *Agile Spain Principios Ágiles*. Recuperado el 017 de Marzo de 2011, de Agile Spain Principios Ágiles: http://www.agile-spain.com/principios_agiles

DesaSoft Desarrollo de Software. (09 de Mayo de 2009). *Proceso Unificado.* Recuperado el 28 de 04 de 2011, de Proceso Unificado: http://iie.fing.edu.uy/ense/asign/desasoft/Teorico2/ProcesoUnificado.pdf

E2G Consultores. (21 de Febrero de 2011). *¿Qué es SourceForge.net?* Recuperado el 15 de Junio de 2011, de ¿Qué es SourceForge.net?: http://www.e2gconsultores.com/E2Globe/2011/02/21/%C2%BFque-es-sourceforge-net/

Inteco Instituto Nacional de Tecnologias de la Comunicación, España. (28 de Septiembre de 2010). *Agile Unified Process (AUP).* Recuperado el 20 de Marzo de 2011, de Agile Unified Process (AUP): http://es.scribd.com/doc/38351703/24/Agile-Unified-Process-AUP

Iso25000. (19 de Agosto de 2009). *PMD*. Recuperado el 15 de Junio de 2011, de PMD: http://iso25000.com/index.php/pmd.html

JUnit Group, J. Y. (s.f.). *JUnit.org*. Recuperado el 23 de Marzo de 2011, de JUnit.org: http://www.junit.org/

Kent Beck, M. B. (07 de Noviembre de 2005). *Agile Spain Manifiesto Ágil*. Recuperado el 17 de Marzo de 2011, de Agile Spain Manifiesto Ágil: http://www.agile-spain.com/manifiesto_agil

Oracle Corporation. (1997). *MySQL The world's most popular open source database*. Recuperado el 30 de Mayo de 2011, de MySQL The world's most popular open source database: http://dev.mysql.com/doc/refman/5.0/es/features.html

Oracle Corporation. (s.f.). *The Java Community Process*. Recuperado el 20 de Marzo de 2011, de The Java Community Process: http://www.jcp.org/en/jsr/overview

Pérez, J. E. (2009). *Javier Eguíluz Pérez.* Creative Commons.

SourceForge.net. (s.f.). *CruiseControl*. Recuperado el 10 de Abril de 2011, de CruiseControl: http://cruisecontrol.sourceforge.net/

Standish Group. (s.f.). *The Standish Group*. Recuperado el 18 de Febrero de 2011, de The Standish Group: http://www.standishgroup.com

Utreras, V. (2009). *Introducción a JQuery.* Recuperado el 30 de Mayo de 2011, de Introducción a JQuery: http://www.slideshare.net/continuumslides/introduccin-a-jquery

Viklund, A. (2010). *MbUnit*. Recuperado el 23 de Marzo de 2011, de MbUnit: http://www.mbunit.com

SISTEMA DE RESERVACIÓN DE LOS LABORATORIOS GENERALES DE COMPUTACIÓN DE LA ESCUELA POLITÉCNICA DEL EJÉRCITO

MANUAL DE INSTALACIÓN

2011

Título	Manual de Instalación y Configuración		
Entregable	Manual de Instalación y Configuración		
Nombre del Fichero	Manual de Instalación Silverlab v.1.0		
Autor	Tatiana Carolina Pozo Molina Carlos Javier Aucancela Maguana		
Versión/Edición	Manual de Instalación y Configuración	**Fecha Versión**	20/07/2011
Aprobado por		**Fecha Aprobación**	DD/MM/AAAA
		N Total de Paginas	

CONTENIDO

1. INTRODUCCIÓN

El presente manual está dirigido a la instalación y configuración técnica del sistema Silverlab, en el cual se analizará las características y requerimientos técnicos para una correcta instalación.

Silverlab es un sistema web desarrollado en java, teniendo como front-end Java Server Pages (JSP) y Java Server Faces (JSF) y características adicionales de Lenguaje de Marcado de Hipertexto (HTML), como back-end, tiene un sistema de gestión de base de datos MySql 5.1.

2. ENTORNO DE DESARROLLO

2.1 NETBEANS

Es un Entorno de Desarrollo Integrado (IDE), hecho principalmente para el lenguaje de programación Java mediante el cual puedes crear aplicaciones de escritorio, web, mobile, enterprise, etc. Su última versión estable es la 7.0.

Network + Java Beans = NetBeans

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 de Netbeans puede ser instalado tanto Windows, Linux, Mac OS, para su respectiva descarga se lo puede realizar desde www.netbeans.org . Ver Gráfico 1.

Gráfico 1. Inicialización del IDE Netbeans

Entre las principales características de Netbeans tenemos:

- ✓ Soporte MySQL en Exploración de Bases de Datos
  - Registro de servidores MySQL
  - Ver, crear y borrar bases de datos
  - Fácil lanzamiento de la herramienta de administración para MySQL
- ✓ Soporte Java Beans
  - Modelos Bean en el Navegador
  - Generador de Propiedades Bean
  - Editor BeanInfo
- ✓ Generador JSF CRUD
  - Generador de aplicaciones JavaServer Faces CRUD a partir de clases de entidades.

- Soporta todo tipo de relaciones de entidades (uno-a-uno, uno-a-varios, varios-a-uno y varios-a-varios).
- Soporta todo tipo de claves principales (columna simple, compuesta y generada).

✓ Generación de Código Javadoc

- Soporte de etiquetas (tags) estándares: @param, etc.
- Completación de Código para parámetros, excepciones, etc.

✓ Soporte RESTful Web Service

2.2 GLASSFISH

Es un servidor de aplicaciones de software libre desarrollado por Sun Microsystems, compañía adquirida por Oracle Corporation, que implementa las tecnologías definidas en la plataforma Java EE y permite ejecutar aplicaciones que siguen esta especificación. La versión comercial es denominada Oracle GlassFish Enterprise Server (antes Sun GlassFish Enterprise Server). Es gratuito y de código libre, se distribuye bajo un licenciamiento dual a través de la licencia CDDL y la GNU GPL.

Netbeans por defecto posee el servidor de aplicaciones GlassFish. De esta forma la depuración y otras tareas asociadas al desarrollo de aplicaciones pueden ser realizadas desde el IDE de Netbeans.

NOTA IMPORTANTE: Para la correcta instalación del sistema Silvelab, necesitamos tener instalado en nuestro sistema operativo la última versión del

JDK de JAVA (http://www.oracle.com/technetwork/java/javase/downloads/jdk-6u26-download-400750.html).

3. INSTALACIÓN Y CONFIGURACIÓN DEL SISTEMA

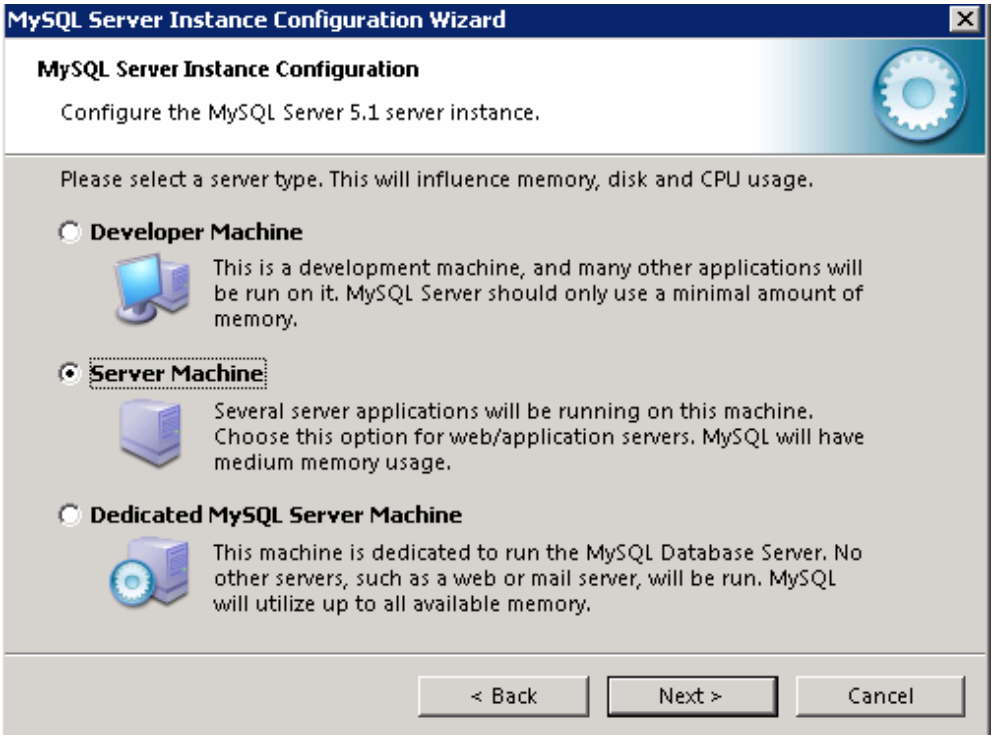
3.1 Descripción

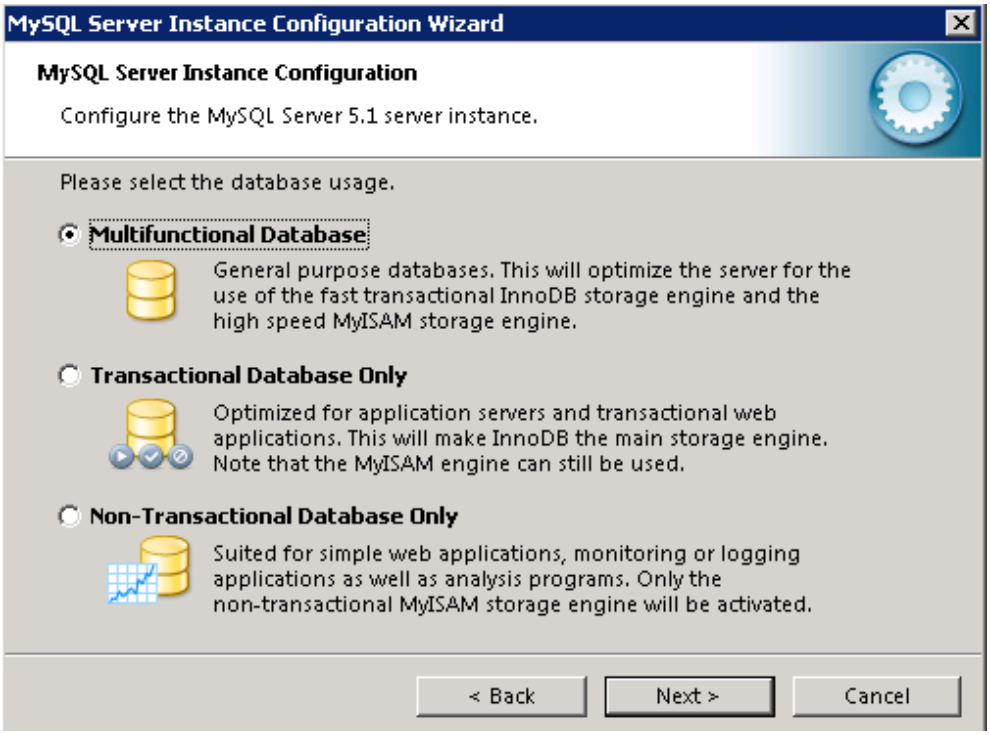
La entrega del sistema se la realiza en formato electrónico adjuntado en el CD de Silverlab. Adicionalmente el sistema se encuentra disponible en el repositorio web de SourceForge, en el cual se podrá acceder a su código fuente, documentación, control de versiones, etc. Desde el siguiente link (http://silverlab.svn.sourceforge.net/).

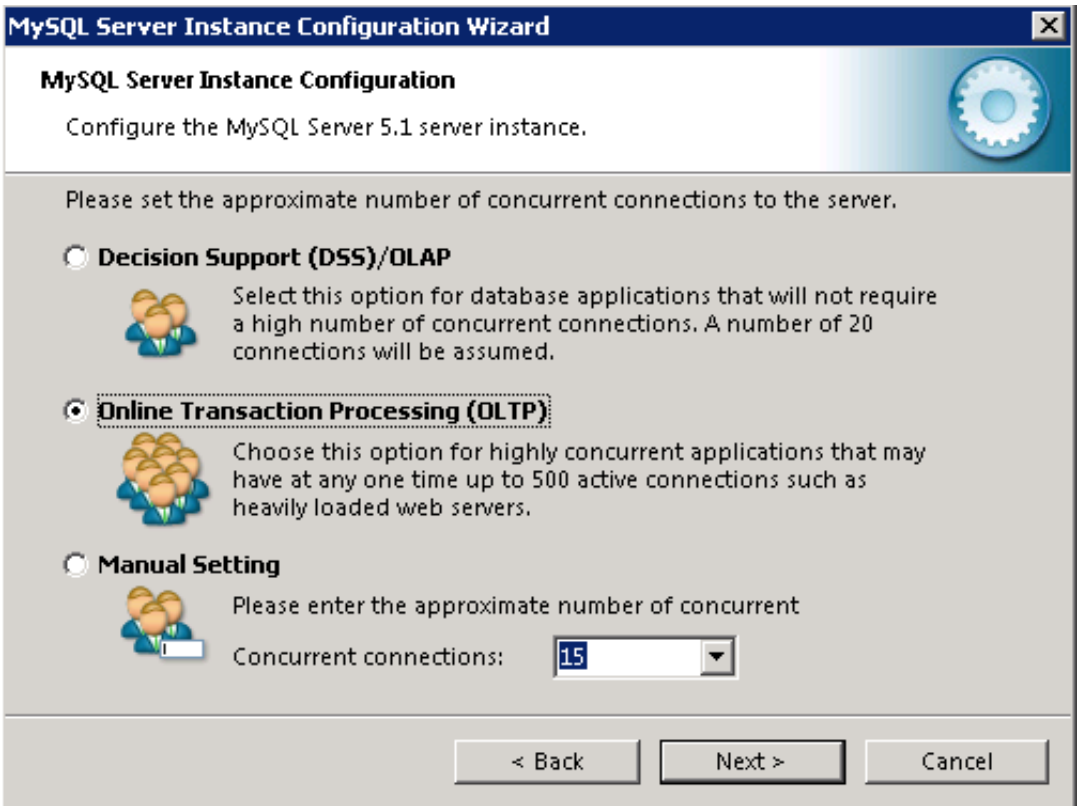
3.2 Instalación

La secuencia de tareas para realizar la instalación de Silverlab es la siguiente:

- ✓ Instalación de la máquina virtual de Java (JDK).
- ✓ Instalación del Sistema Gestor de base de datos MySql.
- ✓ Los parámetros de instalación de la base de datos adicionales a la instalación estándar se los muestra en los siguientes gráficos.


MySQL Server Instance Configuration Wizard
MySQL Server Instance Configuration
Configure the MySQL Server 5.1 server instance.
Please select a server type. This will influence memory, disk and CPU usage.
Developer Machine
This is a development machine, and many other applications will be run on it. MySQL Server should only use a minimal amount of memory.
Server Machine
Several server applications will be running on this machine. Choose this option for web/application servers. MySQL will have medium memory usage.
Dedicated MySQL Server Machine
This machine is dedicated to run the MySQL Database Server. No other servers, such as a web or mail server, will be run. MySQL will utilize up to all available memory.
< Back
Next >
Cancel


MySQL Server Instance Configuration Wizard
MySQL Server Instance Configuration
Configure the MySQL Server 5.1 server instance.
Please select the database usage.
Multifunctional Database
General purpose databases. This will optimize the server for the use of the fast transactional InnoDB storage engine and the high speed MyISAM storage engine.
Transactional Database Only
Optimized for application servers and transactional web applications. This will make InnoDB the main storage engine. Note that the MyISAM engine can still be used.
Non-Transactional Database Only
Suited for simple web applications, monitoring or logging applications as well as analysis programs. Only the non-transactional MyISAM storage engine will be activated.
< Back
Next >
Cancel



- ✓ Abrimos el editor Mysql Browser con la finalidad de ejecutar los scripts para la ejecución de tablas e inserts básicos para que el sistema funciones (los scripts se los encuentra en el CD en la carpeta SCRIPTS) entre estos tenemos:

    - ✓ Esquema de base de datos (EspeSilverlab).
    - ✓ Tablas y procedimientos almacenados.
        - Creates.sql
        - StoreProcedures.sql

- ✓ Instalación de código fuente del proyecto en el Servidor. Con la estructura del mismo previamente establecida Ver Gráfico 2.

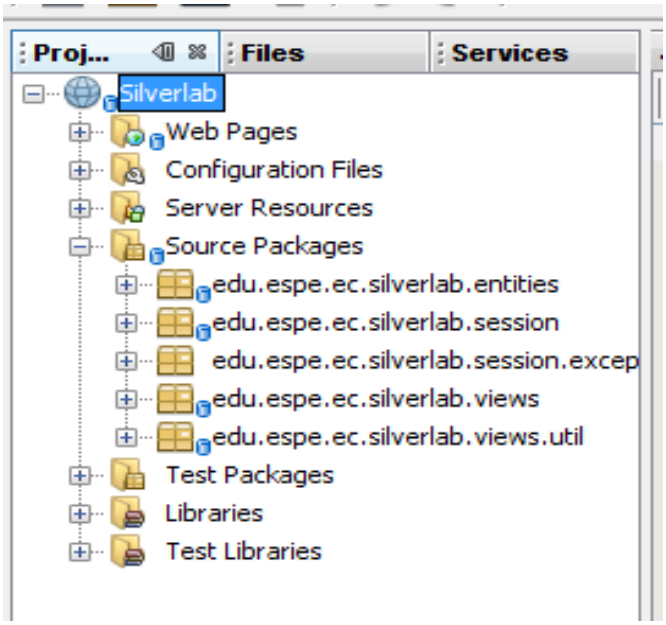


Gráfico 2. Estructura del Proyecto Silverlab.

- ✓ Creación de la conexión jdbc en Netbeans la cual realiza es necesaria para la comunicación con la base de datos Ver Gráfico 3.

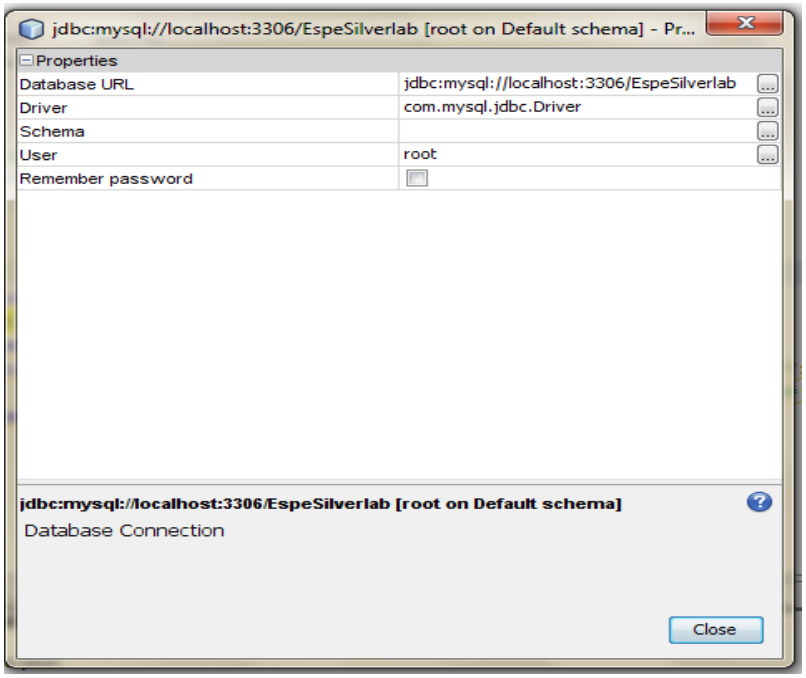


Gráfico 3. jdbc a la Base de Datos.

✓ Subir servicios de Glassfish versión 2.1. como muestra el Gráfico 4.

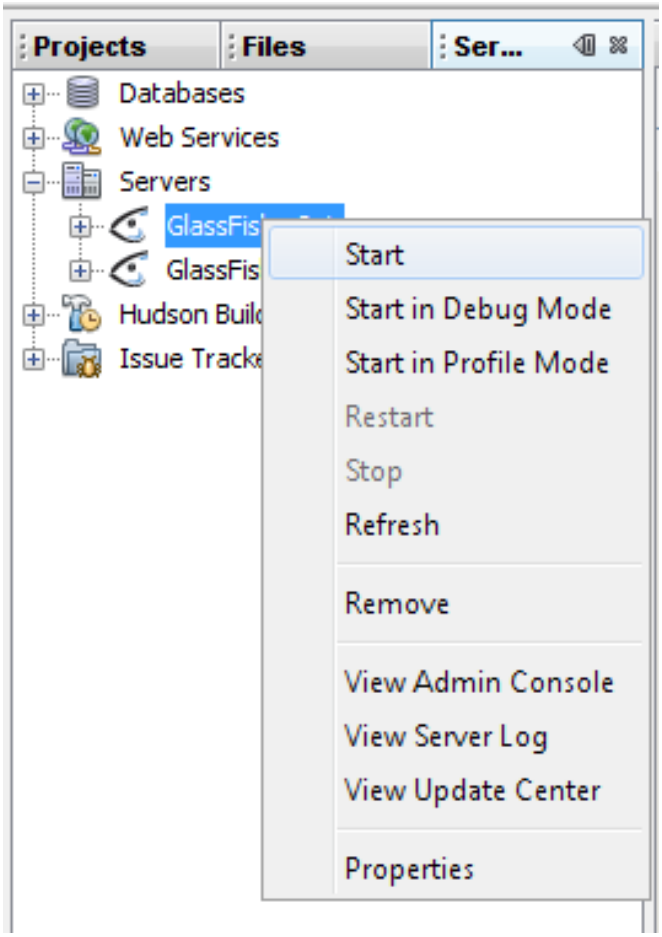


Gráfico 4. Levantamiento de Servicios de Glassfish v2.1.

✓ Ahora estaremos en capacidad de compilar nuestro proyecto. Para lo cual damos clic derecho sobre la raíz del proyecto y escogemos la opción "**Clean and Build**". Obteniendo como resultado el siguiente mensaje Ver Gráfico 5.

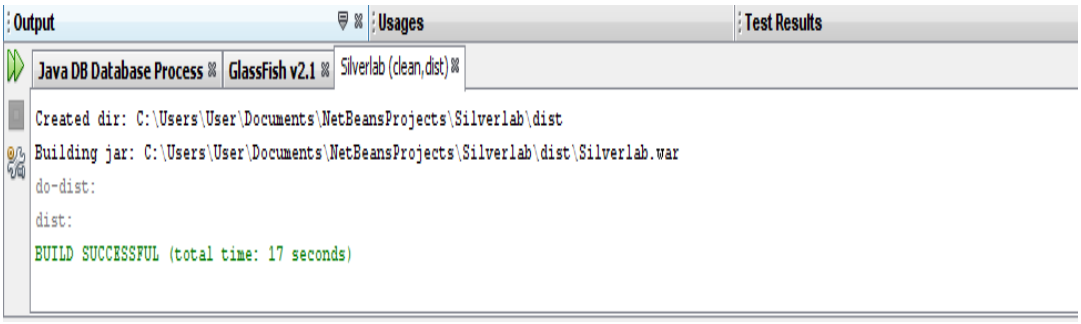


Gráfico 5. Compilación del Proyecto

4. NAVEGADORES

El sistema Silverlab soporta los siguientes navegadores.

- Internet Explorer 6 y 7 (sólo para PC)
- Firefox 1.5 + (PC, Mac y Linux)
- Safari 2 + (Mac).
- Google Chrome.

5. SOLUCIÓN A PROBLEMAS SURGIDOS

5.1 El navegador no presenta el sistema correctamente

- ✓ **Solución 1:** Elimine los archivos temporales de Internet.
- ✓ **Solución 2:** Reinicie al servidor de aplicaciones GlassFish v2.1.
- ✓ **Solución 3:** Verifique que el navegador que utiliza, tiene implementadas las tecnologías actuales de presentación de datos de Internet: Mozilla Firefox 2.0 o posterior y Microsoft Internet Explorer 7.0 o posterior.

6. GLOSARIO DE TÉRMINOS

Frontend.- Es la parte del software que interactúa con el usuario, la manera como lo visualiza.

Backend.- Comprende los componentes que procesan la salida del front-end. Gestiona la conexión a la base de datos.

HTML.- Siglas de HyperText Markup Language, es el lenguaje de marcado predominante para la elaboración de páginas web. Es usado para describir la estructura y el contenido en forma de texto

JSP.- Java Server Pages (JSP) es una tecnología Java que permite generar contenido dinámico para web, en forma de documentos HTML, XML o de otro tipo.

JSF.- Java Server Faces (*JSF*) es una tecnología y framework para aplicaciones Java basadas en web que simplifica el desarrollo de interfaces de usuario.

HOJA DE LEGALIZACIÓN DE FIRMAS

ELABORADA(O) POR

_____

Carlos Javier Aucancela Maguana

_____

Tatiana Carolina Pozo Molina

COORDINADOR DE LA CARRERA

_____

Ing. Mauricio Campaña

Lugar y fecha: _____