UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

SISTEMA INFORMÁTICO PARA LA GESTIÓN DE LOS TESTS FÍSICOS EN LOS ENTRENAMIENTOS DEPORTIVOS

Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas

Autores:

José Antonio Castillo González.

Luis Felipe Matos Marinas.

Tutores:

Dr.C. Armando Pérez Fuente.

MsC. Vladir A. Parrado Cruz.

Ciudad de La Habana, junio 2010

"Año 53 de la Revolución"

" Muchos me dirán aventurero, y lo soy, sólo que de un tipo diferente y de los que ponen el pellejo para demostrar sus verdades."

Ernesto Guevara de la Serna

Declaración de autoría

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo. Para que así conste firmo la presente a los

____ días del mes de _____ del año _____.

Autores: José Antonio Castillo González. Luis Felipe Matos Marinas.

_____ _____

Tutores: Dr.C. Armando Pérez Fuente. MsC. Vladir A. Parrado Cruz.

_____ _____

Tutor: MsC. Vladir Antonio Parrado Cruz.

Especialidad de graduación: Ingeniero Informático.

Categoría Docente: Asistente.

Categoría Científica: Máster.

Correo electrónico: vparrado@uci.cu

Tutor: Dr.C. Armando Pérez Fuente.

Especialidad de graduación: Licenciado en Cultura Física.

Categoría Docente: Titular.

Categoría Científica: Doctor.

Correo electrónico: armandopf@uci.cu

A mi madre por darlo todo por mi incondicionalmente, a mi abuelo Ibrahim González Valentín por ser mi ejemplo en la vida y por ser el hombre más correcto que existe en el mundo, a mi abuela Gabina Ramos Martí, por ser mi inspiración, mi guía, mi conciencia en tiempo buenos y malos, por serlo todo para mí.

José

A mi mamá por todo el amor, apoyo y confianza que en mí depositó, por siempre estar pendiente y orgullosa de mis pasos.

A mi papá por siempre señalarme el camino correcto y prepararme para la vida.

A mi hermana por su cariño infinito, por ser mi inspiración y la alegría de mi vida.

A mis abuelos por siempre creer en mí, por cuidarme, y guiarme siempre en la dirección correcta.

A mis tíos Adis y Sixto por ser mis segundos padres, por apoyarme siempre y brindarme sus consejos.

A Elisa que es mi otra hermana, por escucharme siempre y darme ánimos para seguir.

Luis Felipe

A todos aquellos que de una forma u otra tuvieron algo que ver en nuestro desarrollo como profesional en esta universidad. A nuestros tutores por confiar en nosotros a pesar del poco tiempo que teníamos para la realización de este trabajo.

José y Luis Felipe.

A mis amigos de toda la vida, por ser los hermanos que no tengo, Yuli, Yuniel, Noylen, Renier, Andres y Ken.

A mi buena amiga Mayi, que por estar lejos no le impidió darme fuerzas y ánimos para no rendirme nunca.

A los nuevos amigos que hice en la universidad, por compartir conmigo buenos y malos momentos y demostrarme que puedo contar con ellos en cualquier situación, principalmente a Bryan por soportarnos mutuamente estos cinco años siempre en el mismo grupo, los cuales fueron muchos.

A mi compañero de tesis, Felipe, que más que compañero de tesis es amigo y hermano, porque gracias a él y a su capacidad de cogerlo todo como un vicio terminamos la tesis en tiempo.

A Dino, Orea, Amauri y Leo, por ser más que amigos y comportarse como hermanos en estos cortos años.

A Rubier, Alberto, Raidel, Kresnik por estar siempre cuando se les necesita, a mi hijo Minato por hacerme sentir padre.

A Pablo e Isael por la ayuda prestada en la tesis.

A Laidy, Amaia, Selita, Yudelkis, Viviana, Lianet por ser muy buenas amigas.

A Vania e Ylli por soportarme en su equipo de seminario aun sabiendo lo irresponsable que era.

A Susel por sacar lo mejor de mí y convertirme en mejor persona y mejor estudiante, te lo agradeceré siempre.

A Yoandris Lacoste, por su paciencia y dedicación al enseñarme Física, por ser el mejor profesor que haya tenido.

A Oksana, por comprenderme en estos tiempos difíciles de tesis y darme su cariño y apoyo incondicional siempre.

A mi padre, que la distancia no le ha impedido su preocupación por mí.

A Lupe por ser la madre más preocupada del mundo, por todo su apoyo incondicional y por estar siempre pendiente de mis cosas.

A mi abuelo por darme su ejemplo, porque por querer ser como el, soy mejor persona, por ser mi ídolo, seguro estoy que si llego a ser la mitad de humano que es él, me sentiría realizado.

A mi abuela por ser mi mejor amiga, por todo el amor que me ha dado, por malcriarme y ser recta a la vez, por toda la dedicación que ha tenido en mí, toda una vida no me bastaría para agradecerle todo lo que ha hecho por mí.

José.

Después de 5 años de sacrificio agradezco;

A mis padres y hermana por confiar en mí y estar a mi lado en todo momento.

A Elisa, Adis y Sixto por todo el sacrificio realizado durante todo este tiempo.

A mi familia en general por darme ánimos para seguir adelante.

A todos aquellos que han compartido conmigo todos estos años en especial a aquellos que han demostrado que la amistad no es compartir solo los buenos momentos sino de estar ahí cuando más los necesitas. Ellos son: Jeyser(mi hijo), Jose mi compañero de tesis(el mejor que se puede tener), Dino, Bryan, Leo, Georvis, Yoesner.

A la chica más pilla de la UCI Laidys por tanto amor y cariño desde mi primer día en esta universidad.

A las VIP que han demostrado que puedes contar con ellas para todo: Susel, Viviana, Amaia, Eisel, Lipsy, Ana Li, Sailin

A Isael y Pablo por brindarnos su ayuda incondicional en todo momento.

Al piquete del vicio Rubiel, Yasmany, Alberto, Adrian, Nidelso, Minato.

Luis Felipe.

Resumen

Estudios realizados demuestran la importancia de los tests pedagógicos aplicados al deporte y al entrenamiento deportivo como elemento esencial para el diagnóstico, control y evaluación del proceso de entrenamiento. Con frecuencia las normativas establecidas en una región no se ajustan a otras regiones por el nivel de exigencia de las mismas, casi siempre o se quedan por debajo o van por encima de las poblaciones evaluadas. Muchos entrenadores realizan esta labor de forma manual, ocasionando lentitud e imprecisión en los resultados. En este trabajo se presenta una herramienta que permite elaborar normativas por el propio entrenador y que las mismas cumplan con el requisito de que se ajusten a las características de sus atletas. Resuelve el problema de almacenamiento y centralización de la información que generan los procesos de entrenamientos deportivos. Realiza un sistema de *ranking* con los resultados obtenidos por los atletas que los permiten valorar en su proceso de entrenamiento. Para la creación y almacenamiento de las normativas, baterías de pruebas y *ranking*, se implementó una aplicación web utilizando herramientas informáticas libres.

Palabras Claves: normativa, percentil, test, batería de pruebas, *ranking.*

Índice.

Índice.

INTRODUCCIÓN

Desde el triunfo de la revolución cubana en el año 1959, el país ha dedicado múltiples esfuerzos a las esferas sociales, una de estas es el deporte. El 23 de Febrero de 1961 es fundado el Instituto Nacional de Deportes Educación Física y Recreación (INDER) como organismo principal encargado de la dirección, planificación y ejecución de los programas deportivos en la nación. Bajo la consigna "El deporte derecho del pueblo" se aplica el principio martiano de la no comercialización del movimiento deportivo nacional, quedando eliminado el profesionalismo en Cuba en 1962.

Actualmente el deporte cubano goza de muy buena salud, los resultados alcanzados en los últimos años a nivel internacional en disímiles deportes dan muestra de ello. Los buenos resultados se mantienen, pero actualmente el nivel y rendimiento de los atletas se encuentra en aumento internacionalmente, debido a esto los entrenadores cubanos tienen que incrementar el rendimiento y nivel de los atletas. Los entrenadores deportivos utilizan varios estándares de control de rendimiento sobre los atletas, basados en pruebas. Los resultados arrojados en las pruebas realizadas a los atletas son evaluados según las normativas creadas por el INDER, tomando una muestra en un momento dado. Una normativa es un conjunto de normas aplicables a una determinada materia o actividad (1).

Las normativas existentes internacionalmente para cualquier deporte son muy generales y pueden no servir para algún tipo de población o centro de entrenamiento específico. El INDER tiene estipulado normativas que por lo general no se adecuan a las características de un determinado centro de entrenamiento. En este sentido los entrenadores perciben que algunas normativas establecidas nacionalmente se quedan por debajo o están por encima de las expectativas de sus atletas. Por lo que con frecuencia acuden a otras bibliografías que contemplan variadas pruebas.

Los datos obtenidos de otras bibliografías son basadas en el estudio de la literatura científica actual. Con este tipo de información muchas veces pasa lo mismo que con las normativas del INDER, que no se ajustan a las características de sus atletas o simplemente no existen normativas para medirlas o evaluarlas. En Cuba, las pruebas seleccionadas y sus normativas son productos de investigaciones que se realizan en la nación, como lo constituyen las *Pruebas Nacionales de Eficiencia Física* elaboradas por el INDER y diferentes baterías de pruebas, las cuales contienen un conjunto de resultados que pueden servir para análisis detallados, cada una de estas baterías están divididas en varios deportes, definidos por categorías de edades, todo esto contemplado en el *Programa de la Preparación del Deportista*. Estas baterías de pruebas y sus normativas a pesar de que tienen un gran valor como referencia, en algunos casos no se ajustan a las características de los atletas de una región del país, o a las peculiaridades de un centro de entrenamiento en particular. En Canadá se han hecho

pruebas físicas a personas de la tercera edad, han creado normativas en distintas disciplinas y ejercicios. Estos ejercicios son muy específicos de donde se aplican. En Cuba se pretenden realizar estudios similares al anterior pero no pueden medirse por las mismas normativas, por presentar características diferentes en la población.

Con la ausencia de un medio para que los entrenadores creen sus propias normativas basadas en las capacidades de sus atletas, se crea el inconveniente de no aprovechar al máximo el potencial del entrenamiento o de la actividad física en general.

Con el desarrollo de la ciencia y la técnica ha aumentado el conocimiento, apareciendo nuevos conceptos y términos en las ramas del saber. Una de estas ramas es la Informática, que cada día, con el desarrollo avanza extraordinariamente.

Surge aquí un concepto importante, el de las Tecnologías de la Información y las Comunicaciones (TIC). Las mismas no significan realizar las cosas más rápidas o fáciles, sino que implican nuevas y distintas formas de vincular las tecnologías, la información y las personas. Son potencialidades significativas para el desarrollo personal y colectivo, con posibilidades y limitaciones siempre dependientes de las intencionalidades y de las condiciones de uso. Las TIC no son sólo aparatos o soportes físicos sofisticados, sino que constituyen poderosos sistemas que implica la forma de hacer, de producir, de reproducir y de transmitir información (2).

En Cuba están creándose las condiciones tecnológicas necesarias para el desarrollo de las TIC en varias esferas. Una de las áreas que está siendo beneficiada es la del deporte. Actualmente los entrenadores de todo el país realizan el diagnóstico, control sistemático y evaluación de pruebas de manera manual, guardando esa información en hojas de papel, documentos de textos u hojas de cálculos, pudiendo perderse y haciendo más engorroso el sistema de comparación entre atletas en las mismas pruebas.

Mantener toda esta información de manera consistente por largos períodos de tiempo, permite al entrenador realizar análisis posteriores e ir aplicando controles sistemáticos que vayan informando de los avances que se producen con el entrenamiento deportivo, lo cual permite una retroalimentación de la efectividad que está teniendo el entrenamiento, modificando y ajustando el mismo cuando resulte necesario.

El almacenamiento digital centralizado y seguro de los datos de pruebas por parte de los entrenadores, permitirá mejorar el control sobre las pruebas realizadas a los atletas. Campus Deportivo (3) e Informática y deportes (4) son dos aplicaciones que abordan soluciones que contemplan las dificultades que se presentan en los entrenamientos. Estas herramientas son confiables, dinámicas y

sobre todo facilitan el trabajo del entrenador. El inconveniente que presentan estas herramientas es que son propietarias o están implementadas con aplicaciones propietarias. Lo cual trae como consecuencia que no sea posible su uso sin comprar la licencia.

En Cuba, se realizó una aplicación en la Universidad de las Ciencias Informáticas (UCI) llamada Norma-Test. Esta herramienta se centra en el proceso de la asignatura Educación Física y está centrada únicamente en este contexto, lo que constituye un desarrollo a la medida sin posibilidades de ser extendidas a otras áreas.

Por ello surge el siguiente **problema científico**: ¿Cómo contribuir a la elaboración de normativas, baterías de pruebas y *ranking* para aplicar a test físicos del entrenamiento deportivo en Cuba? Todo ello enmarcado en el **objeto de estudio** "procesos de tratamiento de la información en los entrenamientos deportivos". El **campo de acción** abarca "las herramientas informáticas para el tratamiento de la información en los entrenamientos deportivos". Se define como **objetivo general** "desarrollar un sistema informático para la gestión de los test físicos en los entrenamientos deportivos".

Derivándose del objetivo general los siguientes **objetivos específicos**:

- Realizar una valoración sobre las posibles metodologías de desarrollo de aplicaciones informáticas y herramientas a utilizar.
- Realizar el diseño de un sistema informático que permita la gestión de test físicos en los entrenamientos deportivos.
- Implementar el sistema diseñado.
- Realizar pruebas a la implementación.

Para dar cumplimiento a los objetivos específicos planteados anteriormente y dar solución a la situación problemática es necesario dar cumplimiento a las siguientes **tareas de investigación**:

- Análisis de las tecnologías y herramientas a utilizar.
- Revisión bibliográfica de las metodologías de desarrollo de aplicaciones informáticas.
- Entrevistas a entrenadores deportivos.
- Definición de los requisitos funcionales y no funcionales del sistema.
- Realización del diagrama de casos de usos.
- Realización de los diagramas de clases.
- Realización del diagrama entidad – relación.

- Implementación de los casos de usos definidos.
- Realización de pruebas que permitan detectar posibles errores.
- Confección de una ayuda para la aplicación.

Una vez realizada la aplicación se espera obtener el siguiente **resultado:**

Contar con una aplicación que les brinde a los profesores y entrenadores, la posibilidad de realizar sus propias normativas a partir de resultados de pruebas, evaluar pruebas y baterías de pruebas con normativas contextualizadas, garantizando un trabajo más flexible y eficaz.

El presente documento está estructurado en: resumen, introducción y cuatro capítulos de los cuales a continuación se sintetiza su contenido.

Capítulo 1. Marco Teórico.

Se plasma un estudio sobre los diferentes conceptos importantes, así como las principales herramientas para la implementación de la aplicación, los posibles lenguajes de programación a utilizar, IDE para el lenguaje seleccionado, gestores de base de datos y algunas de las metodologías de desarrollo de aplicaciones informáticas.

Capítulo 2. Características del sistema.

Se presentan las características del sistema propuesto, describiendo cómo debe funcionar, sus potencialidades y características esenciales. Se exponen los requisitos del sistema, tanto funcionales como no funcionales, los casos de uso, los actores y sus respectivas descripciones.

Capítulo 3. Diseño del sistema.

Se obtienen los artefactos del flujo de trabajo de diseño, etapa fundamental en el desarrollo de una aplicación. Se describen los patrones arquitectónicos y de diseño utilizados. Se generan los diagramas de clases para la conformación del sistema, así como su relación con la base de datos.

Capítulo 4. Implementación y prueba.

Los artefactos generados en el capítulo anterior permiten una introducción a la fase de implementación. Es aquí donde se generan los diagramas de componentes y paquetes de implementación. Se propone el estándar de codificación utilizado. También se tratan las pruebas realizadas para comprobar las funcionalidades del sistema, quedando construido completamente el sistema al terminar el capítulo.

Capítulo 1. Marco Teórico

Introducción

Se plasma un estudio sobre los diferentes conceptos importantes, así como las principales herramientas para la implementación de la aplicación, los posibles lenguajes de programación a utilizar, IDE para el lenguaje seleccionado, gestores de base de datos y algunas de las metodologías de desarrollo de aplicaciones informáticas.

1.1 Conceptos Fundamentales

Percentiles: Valor que divide un conjunto ordenado de datos estadísticos de forma que un porcentaje de tales datos sea inferior ha dicho valor (1).

En dependencia de como se escojan los percentiles, estos crean medidas. En caso de existir el percentil ochenta se alcanza una cota superior, el veinte por ciento que alcancen superar la cota tendrían un máximo nivel. Se define una cota intermedia para definir el grupo de nivel intermedio y una cota inferior para los de nivel bajo. Se pueden definir tantas cotas como sean necesarias y tantos niveles de evaluación que se necesiten.

A continuación se muestra una tabla donde se recoge el criterio de percentiles para una escala de evaluación, en este caso el percentil ochenta con 4 niveles.

Tabla 1. Criterio de percentiles para la escala de evaluación.

Percentil	Nivel
80 ó más	I
79 al 60	II
59 al 30	III
30 ó menos	IV

Normativas: Conjunto de normas aplicables a una determinada materia o actividad (1).

Con las normativas se pueden seleccionar percentiles para evaluar a un grupo de atletas. Por lo general las normativas están establecidas por el INDER, teniendo en cuenta la disciplina o la cantidad de estudiantes que realizan una determinada prueba.

El vocablo test viene del idioma inglés, su traducción significa prueba o investigación. Los test no son más que las tareas estandarizadas de carácter oral o en forma de ejercicio físico, los cuales están sujetos a determinadas leyes estadísticas (5).

El concepto de pedagógico es lo expuesto con claridad que sirve para educar o enseñar (1).

La unión de estas palabras se refiere a un término usado en el mundo del deporte. Los **Test Pedagógicos** o también llamados pruebas, posee dos clasificaciones:

1. Teóricos: Evalúa el nivel de conocimiento del contenido del deporte practicado, como son elementos técnicos y tácticos.
2. Prácticos: Evalúa el nivel alcanzado en las condiciones físicas y el rendimiento técnico-táctico del atleta.

Por las características específicas que puedan tener los test pedagógicos se pueden dividir en:

- Generales: Ofrecen una valoración general del atleta.
- Especiales: Calculan las particularidades específicas de atleta en una disciplina deportiva concreta.

Baterías de Pruebas: Conjunto de test pedagógicos o pruebas.

Ranking: Clasificación de mayor a menor, útil para establecer criterios de valoración.

1.2 Antecedentes

1.2.1 Nacionales

Norma-Test

En la UCI se ha implementado una aplicación web que gestiona toda la información relacionada entre los profesores de la asignatura Educación Física y los estudiantes. Permite el almacenamiento de las evaluaciones de las pruebas realizadas a los estudiantes, además gestiona las baterías de pruebas para otorgar la nota que lleva el estudiante por sus resultados. También muestra un *ranking* entre las distintas pruebas que se realizan a los estudiantes. Se implementó en *Python* (6) con el *framework Django* (7).

1.2.2 Internacionales

Informática y Deportes

"Informática y Deportes" es una empresa argentina con más de 15 años de experiencias en la creación de aplicaciones informáticas centradas en la actividad física. En América Latina es una de las empresas líderes en aplicaciones informáticas en lo que al deporte se refiere. Posee una variada gama de sistemas que brindan la posibilidad de planificar, controlar y evaluar el entrenamiento deportivo de los atletas. Agrupa los deportes en dos categorías principales, de grupo e individuales. La característica principal de estos sistemas es que son propietarios, por lo que para su uso es necesario su compra. Otra de sus funcionalidades de interés, es que está hecho para que sus datos sean consultados por un solo entrenador, no por un grupo de entrenadores que trabajen en conjunto o formen parte de una institución u organización (4).

Campus Deportivo

Es una aplicación informática aplicable a múltiples disciplinas para la planificación y control de entrenamientos por parte de los entrenadores deportivos mediante ejercicios que se pueden modificar e imprimir. Los entrenadores y demás usuarios tienen la posibilidad de realizar ejercicios mediante el diseño de gráficos deportivos para sus entrenamientos y sesiones, así como de modificar los gráficos ya diseñados e introducir estos en el sistema completo de gestión y control del entrenamiento deportivo. El inconveniente principal es la necesidad de compra para su uso, además de no ser multiplataforma y que se hace necesario ejecutarlo solo sobre *Windows*. (3)

1.3 Metodologías de desarrollo

Una metodología de desarrollo se refiere a un marco que se utiliza para estructurar, planificar y controlar el proceso. No existe una metodología universal, las características de cada proyecto exigen que el proceso sea flexible y configurable. La metodología en el transcurso del desarrollo de la creación de una aplicación informática define quién debe hacer qué, cuándo y cómo hacerlo (8).

En la actualidad existen una variada cantidad de metodologías de desarrollo de aplicaciones informáticas, clasificándose en dos grupos, tradicionales y ágiles. Las metodologías tradicionales se centran especialmente en el control del proceso, estableciendo rigurosamente las actividades involucradas, los artefactos que se deben producir, y las herramientas y notaciones que se usarán. Estas propuestas han demostrado ser efectivas y necesarias en un gran número de proyectos, pero también han presentado problemas en otros muchos. Dentro de las metodologías tradicionales se encuentra *RUP* (del inglés *Rational Unified Process*) (9) como la más conocida.

Las metodologías ágiles se centran principalmente en el factor humano o en la creación del sistema. Este tipo de metodologías le dan mayor importancia al individuo, a la colaboración con el cliente y al desarrollo incremental de la aplicación informática con iteraciones muy cortas. Está probada su efectividad en proyectos con requisitos muy cambiantes y cuando se exige reducir drásticamente el tiempo de desarrollo pero manteniendo la calidad requerida. Entre las más conocidas se encuentran *XP* (del inglés *Extreme Programming*) (10), *DSDM* (del inglés *Dynamic Systems Developmemt Method*) (11) y *OpenUp*. (12)

Con los planteamientos anteriores se hará una breve investigación sobre estas metodologías para escoger la que más se adecue al proyecto.

1.3.1 RUP

RUP, es una de las metodologías más generales y usadas que existen en la actualidad, pues se puede adaptar a cualquier proyecto. El proceso propuesto por *RUP* para la creación de aplicaciones informáticas posee tres características fundamentales:

- Dirigido por los casos de usos.
- Centrado en la arquitectura.
- Iterativo e incremental.

El Proceso Unificado de Racional consta de cuatro fases o etapas:

1. **Comienzo o Inicio:** Descripción del negocio y delimitación del proyecto describiendo sus alcances con la identificación de los casos de uso del sistema.
2. **Elaboración:** Definición de la arquitectura del sistema y obtención de una aplicación ejecutable que responde a los casos de uso que la comprometen.
3. **Construcción:** Se obtiene un producto listo para su utilización que está documentado y tiene un manual de usuario.
4. **Transición:** Aplicación lista para su instalación en condiciones reales. Puede implicar reparación de errores.

Cada una de estas etapas se desarrolla sobre un ciclo de iteración que consiste reproducir el ciclo de vida en cascada a menor escala. En estas iteraciones se agrupan todas las actividades de los flujos de trabajo, luego de finalizar cada iteración se obtendrá un incremento del producto.

Una particularidad de esta metodología es que, en cada ciclo de iteración, se hace exigente el uso de artefactos, siendo por este motivo, una de las metodologías más importantes para alcanzar un grado de certificación en el desarrollo de aplicaciones informáticas (9).

1.3.2 DSDM

El método *DSDM* fue creado en Inglaterra por la década de los 90 por un grupo de expertos en la materia del desarrollo de sistemas de información. Esta metodología se basa en programación rápida de aplicaciones (RAD). La estructura de *DSDM* está guiada por nueve principios:

1. El involucramiento del usuario es imperativo.
2. Los equipos de *DSDM* deben tener el poder de tomar decisiones.
3. El foco está puesto en la entrega frecuente de productos.
4. La conformidad con los propósitos del negocio es el criterio esencial para la aceptación de los entregables.
5. El desarrollo iterativo e incremental es necesario para converger hacia una correcta solución del negocio.
6. Todos los cambios durante el desarrollo son reversibles.
7. Los requerimientos están especificados a un alto nivel.
8. Las pruebas están integrada a través del ciclo de vida.
9. Un enfoque colaborativo y cooperativo entre todos los interesados es esencial.

Una de las características principales de esta metodología es que el desarrollo es iterativo e incremental y los clientes cooperan directamente con el equipo de trabajo. *DSDM* propone cinco fases de estudio:

1. Estudio de factibilidad.
2. Estudio del negocio.
3. Modelado funcional.
4. Diseño y construcción.
5. Implantación.

Respecto a los roles que trabaja, especifica tres tipos:

1. **Visionario**: Asegura que se satisfagan las necesidades del negocio.
2. **Usuario embajador**: Brinda el conocimiento del negocio y define los requerimientos funcionales y no funcionales de la aplicación.
3. **Coordinador técnico**: Encargado de mantener la arquitectura, verificar la existencia de los componentes y el cumplimiento de los estándares técnicos.

DSMD ha tenido un refinamiento continuo y es ahora una de las metodologías más aceptadas en el mercado (11).

1.3.3 XP

Es una de las metodologías de desarrollo de aplicaciones informáticas con más éxito en la actualidad, utilizada principalmente para proyectos de corto plazo. Está centrada en las relaciones interpersonales como la clave para el éxito. Consiste en una programación rápida o extrema, cuya característica es tener al usuario final como parte del equipo de desarrollo. Se basa en la retroalimentación continua entre el cliente y el equipo de trabajo, la reutilización de código, comunicación fluida entre todos los involucrados y la realización de pruebas a los principales procesos con la finalidad de mitigar posibles errores futuros. La metodología XP aboga la programación en pares, que no es más que dos programadores en una misma estación de trabajo. XP se define especialmente para proyectos de corto plazo con requisitos imprecisos y muy cambiantes, donde puede haber un alto riesgo técnico (10).

1.3.4 OpenUp/Basic

OpenUP/*Basic* es un marco de trabajo de procesos de desarrollo de aplicaciones informáticas de código abierto. Es un proceso modelo y extensible, dirigido a gestión y desarrollo de proyectos de aplicaciones informáticas basados en desarrollo iterativo, ágil e incremental; y es aplicable a un conjunto amplio de plataformas y aplicaciones de desarrollo.

Este proceso de desarrollo unificado está basado en *RUP*, desarrollado por *IBM* y reconocido mundialmente como uno de los procesos de desarrollo de aplicaciones informáticas de mayor calidad. *OpenUP*/*Basic* permite un abordaje ágil al proceso de desarrollo de aplicaciones informáticas, con sólo proveer un conjunto simplificado de contenidos, fundamentalmente relacionados con orientación, productos de trabajo, roles, y tareas.

OpenUP/*Basic* es un proceso interactivo de desarrollo de aplicaciones informáticas simplificado, completo y extensible. Es un proceso para pequeños equipos de desarrollo que valoran los beneficios

de la colaboración y de los involucrados con el resultado del proyecto, por encima de formalidades innecesarias.

OpenUP está caracterizado por cuatro principios básicos interrelacionados:

- **Colaboración** para alinear los intereses y un entendimiento compartido.
- **Balance** para confrontar las prioridades, es decir, necesidades y costos técnicos para maximizar el valor para los *stakeholders*.
- **Enfoque** en articular la arquitectura para facilitar la colaboración técnica, reducir los riesgos y minimizar excesos y trabajo extra.
- **Evolución** continua para reducir riesgos, demostrar resultados y obtener retroalimentación de los clientes.

OpenUP/*Basic* procura un equilibrio entre las necesidades de los involucrados con los resultados del proyecto y los costos técnicos, con el fin de maximizar el valor de los involucrados y las guías del proceso de desarrollo. Desarrolla un ciclo de vida interactivo que mitiga el riesgo a tiempo y ofrece demostrar resultados en curso al cliente del proyecto (12).

1.4 Fundamentación de la metodología

Una vez concluido el estudio sobre las posibles metodologías de desarrollo que pueden guiar el proceso actual, *OpenUp/Basic* resultó ser la indicada en este caso, debido a que es un modelo ágil e incremental, lo cual se ajusta al tiempo requerido y al personal disponible. Genera los artefactos correspondientes en cada ciclo, necesarios para la culminación de aplicación. Es un proceso ágil e interactivo, que provee un conjunto simplificado de elementos a seguir. Además, es la metodología usada por el departamento de Bioinformática.

Se plantea utilizar esta metodología y la variedad de artefactos propuestos por ella, utilizando el Lenguaje Unificado de Modelado (del inglés *UML*) (13) para el desarrollo de la aplicación informática.

1.5 Herramientas CASE

El desarrollo de sistemas que permitan incrementar la productividad y control de calidad en cualquier proceso de desarrollo de aplicaciones informáticas ha ido en aumento en los últimos años. La Ingeniería de *Software* Asistida por Computadora (*CASE* por sus siglas en inglés) reemplaza al lápiz y al papel por la computadora para transformar la actividad de creación de aplicaciones informáticas en

un proceso automatizado. Esta tecnología contribuye a elevar la productividad y la calidad en los sistemas (14).

1.5.1 Rational Rose Enterprise

IBM Rational Rose Enterprise proporciona un lenguaje de modelado común que permite crear con mayor velocidad aplicaciones informáticas de calidad. Entre sus características principales se encuentra que es compatible con *UML*. Soporta patrones *Analysis, ANSI C++, Rose J y Visual C++, Enterprise JavaBeans 2.0*, permite la ingeniería directa e inversa para construcciones comunes en Java 1.5. Incluye un complemento de modelado web, que proporciona el modelado para el desarrollo de aplicaciones web. Esta herramienta resulta de gran utilidad para los desarrolladores de proyectos por lograr cubrir todo el ciclo de vida del proceso desde su fase inicial hasta que termina (15).

Rational Rose posibilita establecer una trazabilidad entre los modelos. Cada rol tiene su propia vista de arquitectura, utilizando un lenguaje común para comprender y comunicar la estructura y funcionalidad del sistema en construcción.

Sin embargo, cuando estas características la convierten en una herramienta muy eficaz, su particularidad de ser propietaria, la hace ser rechazada por muchas empresas y desarrolladores de software.

1.5.2 Visual Paradigm

La Herramienta *CASE Visual Paradigm* (16) utiliza *UML*. Soporta el ciclo de vida completo del desarrollo de una aplicación informática, desde la fase de análisis hasta el despliegue del mismo, es una herramienta libre que permite el modelado de varios lenguajes de programación y otras tecnologías de forma fácil y asequible. Soporta todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. Entre sus más recientes características se incluyen el modelado colaborativo con el sistema de control de versiones (*CVS*, *Concurrent Versions System*) y *Subversion*, así como la interoperabilidad con modelos *UML2* (meta modelos *UML* 2.x para plataforma *Eclipse*) a través de *XML* (17) de intercambio de metadatos (*XMI* o *XML Metadata Interchange*).

1.5.3 Fundamentación de la herramienta seleccionada

Luego de un análisis de las herramientas existentes para modelar sistemas, se determina que la herramienta que más se ajusta al sistema que se desea realizar es *Visual Paradigm*, ya que ofrece un entorno de creación de diagramas para *UML* 2.0, el mismo tiene la disponibilidad de trabajar en

múltiples plataformas, permite generar código de varios lenguajes de programación como *Java, C++, PHP, CORBA IDL,* Esquema de *XML, Ada y Python,* además de que es muy potente y fácil de utilizar.

1.6 Framework

El vocablo *framework* es muy utilizado en ámbitos del desarrollo de aplicaciones informáticas. Existen una gran variedad de *frameworks* en el mundo, desde la creación de aplicaciones médicas, hasta el desarrollo de juegos de vídeo. Se puede considerar como una aplicación genérica incompleta y configurable a la que se puede añadir las últimas piezas para construir una aplicación concreta.

Los objetivos principales que tiene un *framework* es acelerar el proceso de desarrollo de un sistema, reutilizar código ya existente y promover buenas prácticas como el uso de patrones de diseños (17).

Framework de desarrollo web

Existen una variada cantidad de *frameworks* que son destinados solamente para la creación de aplicaciones web. Dentro de los más usados y conocidos se encuentran *ASP.NET* (18), *Symfony* (19), *Django, Grails* (20) entre muchos otros.

1.6.1 ASP.NET

ASP (del inglés *Active Server Pages*) es una tecnología de *Microsoft* del tipo lado del servidor para páginas web generadas dinámicamente. *ASP.NET* es la versión mejorada de *ASP*. Uno de los objetivos principales de ASP.NET es aprovechar toda la capacidad del *CLR* (del inglés *Common Languaje Runtime*), el *CLR* compila en algún punto todos los códigos de aplicaciones en códigos naturales de máquina. En *ASP.NET* el código se separa de la lógica de la aplicación, brinda la posibilidad de escoger el lenguaje que desee el programador, por defecto trae integrado *C#* y *VB.NET*. Otra característica importante es que el *framework .NET* tiene integrado librerías de clases que son comunes en toda la plataforma *.NET*, esto permite crear aplicaciones con un considerable ahorro de líneas de código. *ASP.NET* por las múltiples funciones que tiene es más lento que la mayoría de los *frameworks* de desarrollo web, pero sin lugar a dudas el mayor inconveniente que presenta, es su condición de propietario (18).

1.6.2 Symfony

Symfony es un *framework* diseñado para optimizar el desarrollo de aplicaciones web, debido a todas las características que presenta. Se encuentra desarrollado completamente con *PHP 5*. Ha sido utilizado en numerosos proyectos reales y se utiliza en sitios web de comercio electrónico de primer

nivel. *Symfony* separa la lógica del negocio, la lógica del servidor y la presentación de la aplicación. Reduce enormemente el tiempo de desarrollo de una aplicación web compleja. Automatiza las tareas más comunes y reutiliza el código cada vez que se crea una nueva aplicación web. Tiene compatibilidad con la mayoría de los gestores de base de datos. Es multiplataforma, sin embargo, su mapeo de objeto relacional para conectarse a las bases de datos es propenso a lentitud, pues conlleva a gastos innecesarios en la generación y configuración del código. El desarrollo de la autenticación en *Symfony* utiliza medidas de seguridad muy débiles, los roles son llamados credenciales. Esto no es una medida de seguridad por roles puesto que para saber a qué usuario se le han otorgado credenciales, antes se tiene que haber leído toda la lista de roles. Al estar desarrollado con *PHP* 5 se debe tener un avanzado conocimiento de este lenguaje (19).

1.6.3 Django

Django es un *framework* web de código abierto escrito en el lenguaje *Python* que permite construir aplicaciones web más rápido y con menos código. *Django* fue inicialmente desarrollado para gestionar aplicaciones web de páginas orientadas a noticias de *World Online,* más tarde se liberó bajo licencia *BSD*. *Django* se centra en automatizar todo lo posible y se adhiere al principio *DRY* (del inglés *Don't Repeat Yourself*) cuya traducción al español es no te repitas. *Django* abstrae a los desarrolladores de los problemas más comunes y acelera las tareas más frecuentes en la programación. Proporciona un método de mapear las *URLs* ejecutando un código en especial para cada una. Muestra y valida formulario de manera muy simple, manipulando el código y adaptándolo a las necesidades de la aplicación. Convierte los datos enviados por los usuarios en estructuras de datos, las cuales pueden ser manipuladas fácilmente. Separa el contenido de la presentación de la lógica del negocio, apoyándose en el uso de plantillas, lo cual es útil cuando se necesite realizar cambios de apariencia (7).

1.6.4 Grails

Grails (20) es un *framework* de desarrollo para aplicaciones web, creado en Java y basado en *Groovy* (21). Permite que se le agreguen extensiones desarrolladas por terceros. Abarca las tres capas del desarrollo, acceso a la base de datos, capa de negocio y presentación. *Grails* consigue un mayor rendimiento basándose en otros *frameworks* de desarrollo de código abierto, principalmente *Spring* (22) e *Hibernate* (23) que son potentes en su campo.

Su principal característica es que está diseñado para que se programe en *Groovy*, el cual es un lenguaje dinámico basado en *Java*, pero que incorpora muchas funcionalidades nuevas permitiendo

programar con menos código. *Grails* se basa en el patrón Modelo Vista Controlador (*MVC*) y soporta *Ajax* (24).

Este *framework* posee buena documentación con una numerosa comunidad de desarrolladores y usuarios, inspirado en *Ruby on Rails* (25) y se encuentra en constante desarrollo.

1.6.5. Dojo *Toolkit*

Dojo Toolkit es un *framework JavaScript* (27) que ofrece componentes de alta calidad y fáciles de usar, con el objetivo de crear aplicaciones Web de alto nivel de respuesta. Los complementos en *Dojo* se usan para enriquecer aplicaciones Web, entre estos se encuentran:

- Menús, pestañas y *tooltips*.
- Soporte para arrastrar y soltar.
- Calendario, selector de tiempo y reloj.
- Tablas ordenables.
- Gráficos dinámicos.

En general, *Dojo* incluye un diseño e implementación estándar para los diferentes navegadores, permitiendo utilizar el mismo código fuente del navegador que se utilice. Presenta un sistema de módulos junto con un sistema de construcción, que le permite dividir códigos en pequeños pedazos manejables. Posee bibliotecas independientes que se cargan en dependencia de las necesidades, que implementan otras funcionalidades (26).

1.6.6 Fundamentación del *framework* seleccionado

El *framework* de desarrollo seleccionado fue *Grails*, por todas las funcionalidades que permite. Posee una corta gestión de configuración para el comienzo de proyectos, lo que permite crear aplicaciones web en menor tiempo. Es altamente expresivo, dinámico y posee una perfecta integración con Java. Otro de los aspectos es los conocimientos que posee el grupo de desarrollo sobre ese *framework*, al haber trabajado con *Grails* en proyectos anteriores. Para el trabajo con las vistas se selecciona *Dojo*, por los complementos que posee y permitir la construcción de las interfaces con un diseño agradable y uniforme.

1.7 Lenguaje de programación

El lenguaje de programación seleccionado es *Groovy* dada la elección del *framework Grails*. Es un lenguaje dinámico y flexible, con el se puede ahorrar casi un 90% de líneas de código respecto al lenguaje *Java*. Incorpora además poderosas características de lenguajes como *Phyton, Ruby* y *Smaltalk*. Permite la integración con todos los objetos y librerías existentes en *Java*. Entre sus principales características (21) se encuentran:

- *Closures*: bloque de código anónimo definido entre llaves, el cual toma argumentos y retorna valores.
- Sintaxis nativas para *List* y *Maps*.
- Soporte nativo para expresiones regulares.
- Soporte para tipificación dinámica y estática.
- Soporta código embebido dentro de cadenas.

1.8 Sistema gestor de base de datos

Un sistema gestor de base de datos se puede definir como un conjunto de herramientas acompañantes del motor de base de datos y permiten interactuar con esta. Una de las funciones más importantes de un gestor de base de datos es la capacidad de realizar copias de seguridad y recuperación de datos. También se encarga de la restricción de accesos no autorizados y muestra una interfaz gráfica al usuario para el trabajo de los datos.

1.8.1 PostgreSQL

PostgreSQL es hoy en día el gestor de bases de datos objeto-relacional de código abierto más avanzado. Ofrece control de concurrencia multiversión, soportando casi todas las sintaxis *SQL*. Cuenta con amplios enlaces a lenguajes de programación importantes, como son *C, C++, Java, Perl* y *Phyton* incluyendo interfaz visual para ellos. Provee una arquitectura fiable y una integridad total de los datos. Trabaja la concurrencia permitiendo el bloqueo de tabla y filas para controlar el acceso a los datos. Es un gestor de base de datos multiplataforma, permite la declaración de funciones propias, así como la definición de disparadores. Soporta el uso de índices, reglas y vistas. Incluye herencia entre tablas aunque no entre objetos, ya que no existen, por lo que a este gestor de bases de datos se le incluye entre los gestores objeto-relacionales (28).

1.8.2 MySQL

MySQL es un sistema de gestión de bases de datos relacional. Su diseño multihilo le permite soportar una gran carga de información de forma muy eficiente. Está diseñado para entornos de producción críticos, con alta carga de trabajo así como para integrarse en aplicaciones para ser distribuido. *MySQL* es una marca registrada de *MySQL AB*, aunque tiene una doble licencia. Los usuarios pueden elegir entre usar el producto *MySQL* como un sistema *Open Source* bajo los términos de la licencia *GNU* (del inglés *General Public License*) o pueden adquirir una licencia comercial estándar de *MySQL AB*.

Se encuentra programado en *C* y en *C++,* probado con un amplio rango de compiladores diferentes, multiplataforma, proporciona sistemas de almacenamiento transaccionales y no transaccionales. El servidor está disponible como un programa separado para usar en un entorno de red *cliente/servidor*. También es disponible como biblioteca y puede ser incrustado en aplicaciones autónomas. Dichas aplicaciones pueden usarse por sí mismas o en entornos donde no hay red disponible. Los clientes pueden conectar con el servidor *MySQL* usando *sockets TCP/IP* en cualquier plataforma (28).

1.8.3 Fundamentación del gestor de base de datos seleccionado

El gestor de base de datos seleccionado es *PostgreSQL 8.2.4,* principalmente por ser multiplataforma lo cual permitirá ser utilizado tanto en *Windows* como en *Linux*, dependiendo de los intereses del equipo. Otro aspecto fundamental a tener en cuenta es que es libre, parámetro en el cual lleva cierta ventaja sobre *MySQL*, el cual conlleva la necesidad una licencia para su uso.

1.9 Entorno Integrado de Desarrollo (IDE)

Un *IDE* es un conjunto de aplicaciones informáticas que consisten en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica. Puede dedicarse exclusivamente a un lenguaje de programación o bien puede utilizarse en varios lenguajes. Para el trabajo con el lenguaje de programación *Groovy* existen tres *IDEs*, *Eclipse* (30), *NetBeans* (31) e *IntelliJ IDEA* (32).

1.9.1 Eclipse

Eclipse (30) es un entorno de desarrollo integrado de código abierto. Posee una funcionalidad muy grande, pero esa funcionalidad es muy genérica. Permite que nuevos componentes puedan utilizar distintos tipos de contenido, para realizar determinadas tareas con contenidos existentes. La Plataforma *Eclipse* permite descubrir, e invocar funcionalidad implementada en componentes llamados *plug-ins*.

Soporta las herramientas proporcionadas por diferentes fabricantes de aplicaciones informáticas independientes. Contiene funciones que permiten manipular diferentes contenidos como son *HTML, Java, C, JSP, EJB, XML,* y *GIF*. Facilita una integración transparente entre todas las herramientas y tipos de contenidos sin tener en cuenta al proveedor. Proporciona entornos de desarrollo gráfico (*GUI* por sus siglas en inglés) o no gráficos. Permite ejecutarse en una gran variedad de sistemas operativos, incluyendo *Windows* y *Linux*.

Existe una gran comunidad que se encarga de mantener a *Eclipse* actualizado, para esto se dedican a crear complementos que se pueden integrar a la plataforma. Actualmente uno de esos complementos permite a *Eclipse* usar *Groovy* y el *framework* de desarrollo *Grails*. Aún este aditamento no está terminado funcionalmente y posee problemas visibles cuando se desea desarrollar una aplicación de importancia.

1.9.2 NetBeans

NetBeans (31) es un proyecto exitoso de código abierto con una gran base de usuarios, una comunidad en constante crecimiento. *Sun MicroSystems* fundó el proyecto de código abierto *NetBeans* en junio 2000 y continúa siendo el patrocinador principal de los proyectos. Hoy en día hay disponibles dos productos: el *NetBeans IDE* y *NetBeans Platform*. Es una herramienta para que los programadores puedan escribir, compilar, depurar y ejecutar programas, está escrito en *Java* pero eso no quiere decir que no sirva para cualquier otro lenguaje de programación. Existe además un número importante de módulos para extender la plataforma. *NetBeans IDE* es un producto libre y gratuito sin restricciones de uso. *NetBeans Platform* es una base modular y extensible usada como estructura de integración para crear grandes aplicaciones de escritorio. El código fuente del *NetBeans* está disponible para su reutilización de acuerdo con la Licencia de Desarrollo y Distribución Común v1.0 (*CDDL* por sus siglas en inglés) y la *GPL* v2 (27).

1.9.3 IntelliJ IDEA

IntelliJ IDEA (32) es un *IDE* centrado en la productividad del código. El editor entiende profundamente el código, hace sugerencias muy adecuadas y ayuda a dar forma a su código. Soporta *Java, Groovy, Scala, XML* con asistencia de código, *JUnit, CVS, Subversion*. Trae integración *Ant* y *Maven*. Entre sus principales características se encuentra la asistencia inteligente de codificación, la generación de código, el estilo de código, la documentación que posee, el análisis de código sobre la marcha lo cual usa un método intuitivo para ayudar a escribir el código. Posee una amplia gama de lenguajes de programación permitidos, los cuales son *Java, JavaScript, Flex, HTML, XHTML, CSS, XML, XSL,*

Ruby, JRuby, Groovy y sintaxis *SQL*. Trae integrado varias tecnologías y *framework* que son *JSP, JSF, EJB, AJAX, GWT, Struts, Struts 2, JBoss Seam, Spring, Hibernate/JPA, Web Services, Rails, Grails, Java ME MIDP/CLDC*. Otra característica de importancia que se debe conocer, es que es una aplicación propietaria y por tanto se debe pagar por su licencia o uso (28).

1.9.4 Fundamentación del Entorno Integrado de Desarrollo

El entorno de desarrollo seleccionado es *NetBeans*, por todas las funcionalidades y la potencia que este posee a la hora de trabajar con el lenguaje de programación *Groovy* y el *framework Grails*. Además, por ser una aplicación libre y multiplataforma.

Conclusiones del capítulo

En el presente capítulo se les da cumplimiento a las dos primeras tareas de la investigación, definiéndose las herramientas y tecnologías a utilizar. Quedando seleccionadas, la metodología de desarrollo *OpenUp* y para realizar el modelado del sistema la herramienta case *Visual Paradigm*, apoyándose en el lenguaje *UML*. El *IDE* seleccionado fue *NetBeans* junto con el lenguaje de programación dinámico *Groovy*. Como *framework* de desarrollo de aplicaciones web, se seleccionó *Grails* y como gestor de base de datos, *PostgreSQL*.

Capítulo 2. Características del sistema

Introducción

Se presentan las características del sistema propuesto, describiendo cómo debe funcionar, sus potencialidades y características esenciales. Se exponen los requisitos del sistema, tanto funcionales como no funcionales, los casos de uso, los actores y sus respectivas descripciones.

2.1 Propuesta del sistema

Se implementará un sistema capaz de gestionar toda la información recogida en las distintas pruebas llevadas a cabo por los entrenadores. Además de gestionar sus propias normativas y percentiles. Los atletas podrán consultar sus resultados y compararlos con los demás deportistas. Se podrán obtener mejores resultados con las nuevas características que se definen a continuación:

1. Trabajar tanto con las normativas establecidas internacionalmente, así como las creadas por el INDER, incluso crear sus propias normativas.

2. Manejar test pedagógicos.

3. Hacer baterías de pruebas con los test pedagógicos.

4. Mostrar un *ranking* con los resultados de las pruebas.

El primer aspecto brindará la posibilidad al entrenador no solo de trabajar con las normativas establecidas internacionalmente o por el INDER, sino que además pueda crear sus propias normativas atendiendo al procedimiento estadístico de los percentiles.

En el segundo punto el entrenador contará con la posibilidad de crear test pedagógicos y estos a su vez contribuirán a la formación de las baterías de pruebas que miden las variadas capacidades físicas o técnicas que se necesitan desarrollar. Las baterías posteriormente arrojarán los resultados de los atletas, mostrando en un *ranking* el nivel de esfuerzo de cada deportista, los atletas tendrán visión de esta tabla. La aplicación proveerá no solo estas funcionalidades sino que también contará con un sistema que el atleta podrá modificar sus valores temporalmente para saber qué resultados podría alcanzar mejorando alguna prueba específica.

La propuesta de crear una nueva aplicación web que gestione los datos de entrenadores sobre sus atletas tendrá la ventaja de tener centralizada y organizada completamente la información. Será una

aplicación independiente del sistema operativo y permitirá nuevas funciones que mejoren el análisis de los resultados de los atletas para confirmar la eficiencia de su entrenamiento.

2.2 Modelo de Dominio

Debido a que se hace difícil encontrar una estructura de los procesos del negocio que tienen que ver con el objeto de estudio, y siendo difícil la identificación de los mismos, se decidió la realización de un Modelo de Dominio (33). Un modelo de dominio se plantea con el objetivo de ayudar a comprender los conceptos que utilizan los usuarios, que serán los mismos a utilizar para el desarrollo de la aplicación. El modelo de dominio se describe a través de un diagrama de clases de *UML*, el cual define las principales clases conceptuales que intervienen en el desarrollo del sistema. El modelo captura los objetos más importantes, representando en un diagrama los eventos reales relacionados con el sistema, que precisan las clases relevantes que intervienen en el contexto planteado. Estos objetos son los elementos a utilizar para aclarar el dominio de un problema, muchos de ellos son: conceptos, atributos, asociaciones y roles.

A continuación se identifican los principales conceptos que se utilizarán en el diagrama Modelo de Dominio:

Atleta: Persona que entrena alguna disciplina deportiva.

Entrenador: Persona encargada de definir sus atletas y llevar el control de las pruebas y evaluaciones.

Prueba: Ensayo o test que se realiza para tener conocimiento previo de un resultado final.

Batería de Prueba: Conjunto de pruebas para la conformación de criterios generales.

Evaluación: Criterio dado a una prueba por el resultado obtenido.

Normativas: Pauta elegida para definir una cota en la evaluación.

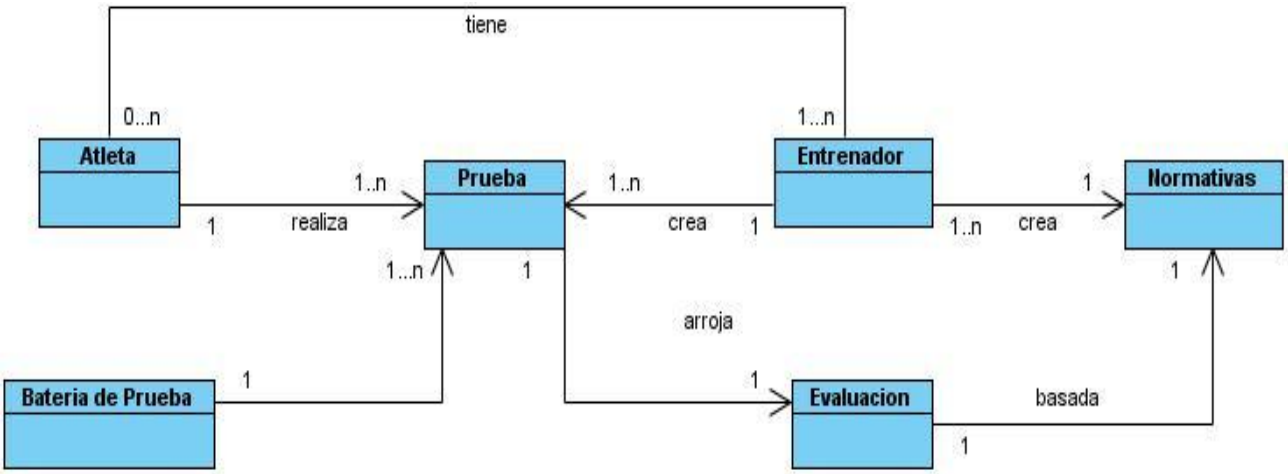

Figura 1. Modelo de Dominio.

2.3 Especificación de los requisitos de la aplicación

La especificación de los requisitos es uno de los flujos de trabajo más importantes, en él se establece qué tiene que hacer exactamente el sistema que se va a desarrollar. Los requisitos son el contrato que debe cumplir la aplicación y que el usuario final debe entender y aceptar.

2.3.1 Requisitos funcionales

Los requisitos funcionales son capacidades o condiciones que el sistema debe cumplir, define lo que el sistema debe hacer. A continuación aparece la lista de los requisitos funcionales:

RF 1. Autenticar usuarios.

RF 2. Registrar usuario.

RF 3. Listar *rankings*.

RF 4. Simular pruebas.

RF 5. Listar pruebas.

RF 6. Crear prueba.

RF 7. Modificar prueba.

RF 8. Eliminar prueba.

RF 9. Listar pruebas aplicadas.

RF 10. Crear prueba aplicada.

RF 11. Listar prueba aplicada.

RF 12. Eliminar prueba aplicada.

RF 13. Listar baterías de pruebas.

RF 14. Crear baterías de prueba.

RF 15. Modificar baterías de prueba.

RF 16. Eliminar baterías de prueba.

RF 17. Listar normativas.

RF 18. Crear normativa.

RF 19. Modificar normativa.

RF 20. Eliminar normativa.

RF 21. Listar percentiles.

RF 22. Crear percentil.

RF 23. Modificar percentil.

RF 24. Eliminar percentil.

RF 25. Listar grupos.

RF 26. Crear grupo.

RF 27. Modificar grupo.

RF 28. Eliminar grupo.

RF 29. Modificar rol a un usuario.

RF 30. Listar usuarios.

RF 31. Modificar usuario.

RF 32. Eliminar usuario.

RF 33. Listar roles.

RF 34. Crear rol.

RF 35. Modificar rol.

RF 36. Eliminar rol.

RF 37. Listar resultados.

RF 38. Crear resultado.

RF 39. Modificar resultado.

RF 40. Eliminar resultado.

2.3.2 Requisitos no funcionales

Los requisitos no funcionales son propiedades o cualidades que el producto debe tener. También puede ser alguna restricción que este debe tener para cumplir una determinada funcionalidad. A continuación se presentan los requisitos no funcionales:

Usabilidad

La usabilidad es la capacidad con la que pueda ser aprendida y operada la aplicación por los usuarios a los que va a ser destinada, en este caso a los atletas y entrenadores del INDER. Para su correcto uso los usuarios deben tener conocimientos básicos de navegación. El administrador del sitio debe tener conocimiento sobre la gestión de usuarios. Los entrenadores deben tener conocimiento sobre los procesos de gestión de pruebas, pruebas aplicadas, baterías de pruebas, normativas y percentiles.

Software

La aplicación se instala en una computadora servidor. Para el correcto funcionamiento del sistema se debe garantizar un servidor con plataforma de funcionamiento de los sistemas operativos Linux o Windows, que soporte *Apache Tomcat Server* 6.0 (32). En las terminales cliente sirve cualquier sistema operativo que tenga instalado Internet Explorer 7 en adelante, Mozilla Firefox 3 en adelante y Opera versión 11 en adelante.

Hardware

Para el servidor:

- Procesador Pentium Dual Core 2.20GHz o mayor.
- 4 GB de RAM o mayor.
- Disco duro de 320 GB o mayor.
- Dispositivo de red.

Para las terminales clientes:

- Procesador Pentium 233 MHz (recomendado 500 MHz o mayor).

- 128 MB de RAM (recomendado 256 o mayor).
- Soporte de video de al menos 800x600 y 24 bits.
- Dispositivo de red.

Confiabilidad

Todas las respuestas de salida del sistema tienen que tener un cien por ciento de precisión y veracidad, esto se garantiza haciendo pruebas al sistema previamente. La gestión, sobre las pruebas y demás funcionalidades estarán protegidas del acceso no autorizado. Los usuarios tendrán un nivel de acceso dependiendo del rol que cumpla.

Interfaz

El diseño de la interfaz debe ser sencillo y amigable. La interfaz solo debe mostrar las funcionalidades del rol de usuario que esté utilizando la aplicación. Se debe garantizar una correcta organización de la información para permitir una adecuada interpretación. Debe ser interactiva y poseer un uso de colores adecuados.

Ayuda y documentación en línea

La aplicación debe estar proporcionada en todo momento con la documentación adecuada para que los usuarios consulten en caso de algún inconveniente.

2.4 Modelo de Casos de Uso

El Modelo de Casos de Uso describe la funcionalidad del nuevo sistema propuesto. Un caso de uso representa una unidad de interacción entre un usuario y el sistema. También se describen los actores del sistema y los casos de uso que van a representar las funcionalidades.

2.4.1 Patrones de Casos de Uso

Son comportamientos que deben existir en el sistema, ayudan a describir qué debe hacer aplicación y cómo interactuar con el usuario. Son generalmente utilizados como plantillas, dicen cómo deberían ser estructurados y organizados los casos de usos (34).

Patrón CRUD

Su nombre es un acrónimo del inglés Create, Read, Update y Delete (Crear, Leer, Actualizar y Eliminar). Este patrón deberá ser usado cuando todas las operaciones contribuyen al mismo valor de

negocio y todas son cortas y simples. Disminuye la cantidad de casos de usos simples uniéndolos en uno solo.

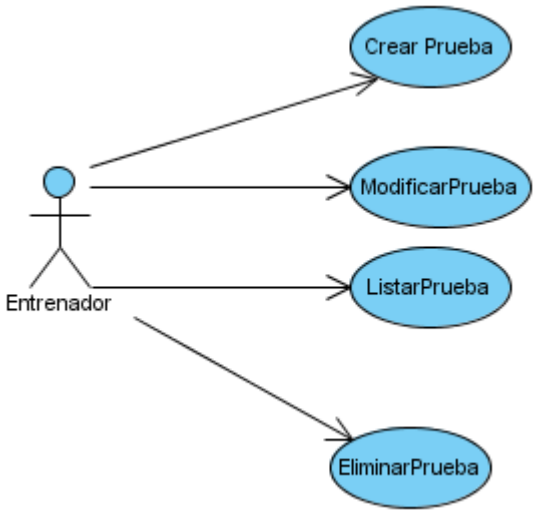

Figura 2. Diagrama de Casos de Usos sin patrón CRUD.

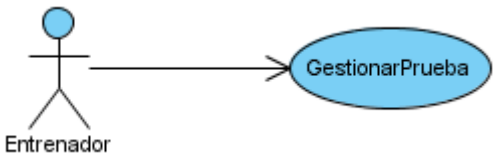

Figura 3. Diagrama de Casos de Usos con patrón CRUD.

Patrón Múltiples Actores

Este patrón elimina la acción de que más de un actor con el mismo rol inicie el mismo caso de uso.

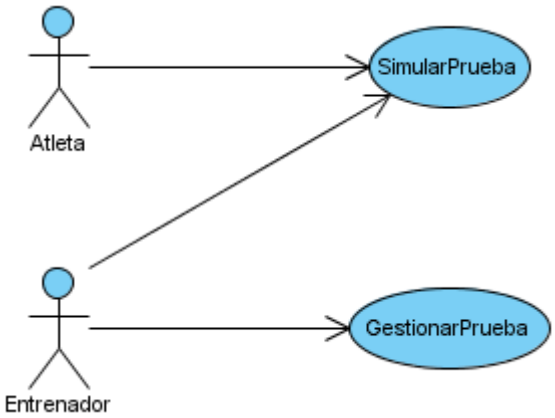

Figura 4. Diagrama de Casos de Usos sin patrón múltiples actores.

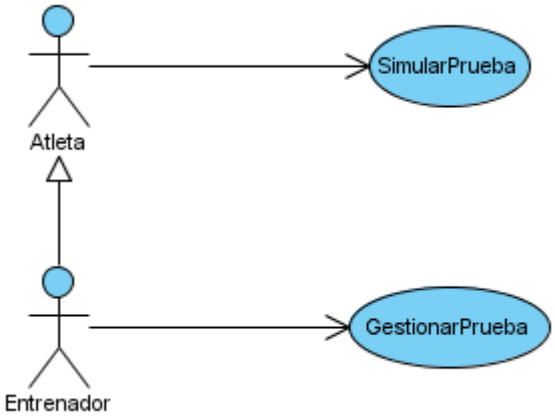

Figura 5. Diagrama de Casos de Usos con patrón múltiples actores.

2.4.2 Determinación y justificación de los actores

Se entiende como personal relacionado con el sistema todo aquel que interactúe con la aplicación y que obtiene un resultado de uno o varios procesos que se ejecutan en la misma.

Tabla 2. Actores del negocio.

Actores del negocio	Descripción
Administrador	Es el encargado de cerciorar que se gestionen correctamente los datos, además otorga los permisos correspondientes a los usuarios que interactúen con el sistema
Entrenador	Usuario que tendrá una interacción más profunda con el sistema.
Atleta	Usuario que podrá interactuar con el sistema, bajo ciertos permisos.
Usuario	Usuario común que solo tendrá acceso de lectura a la información.

2.4.3 Diagrama de Casos de Uso

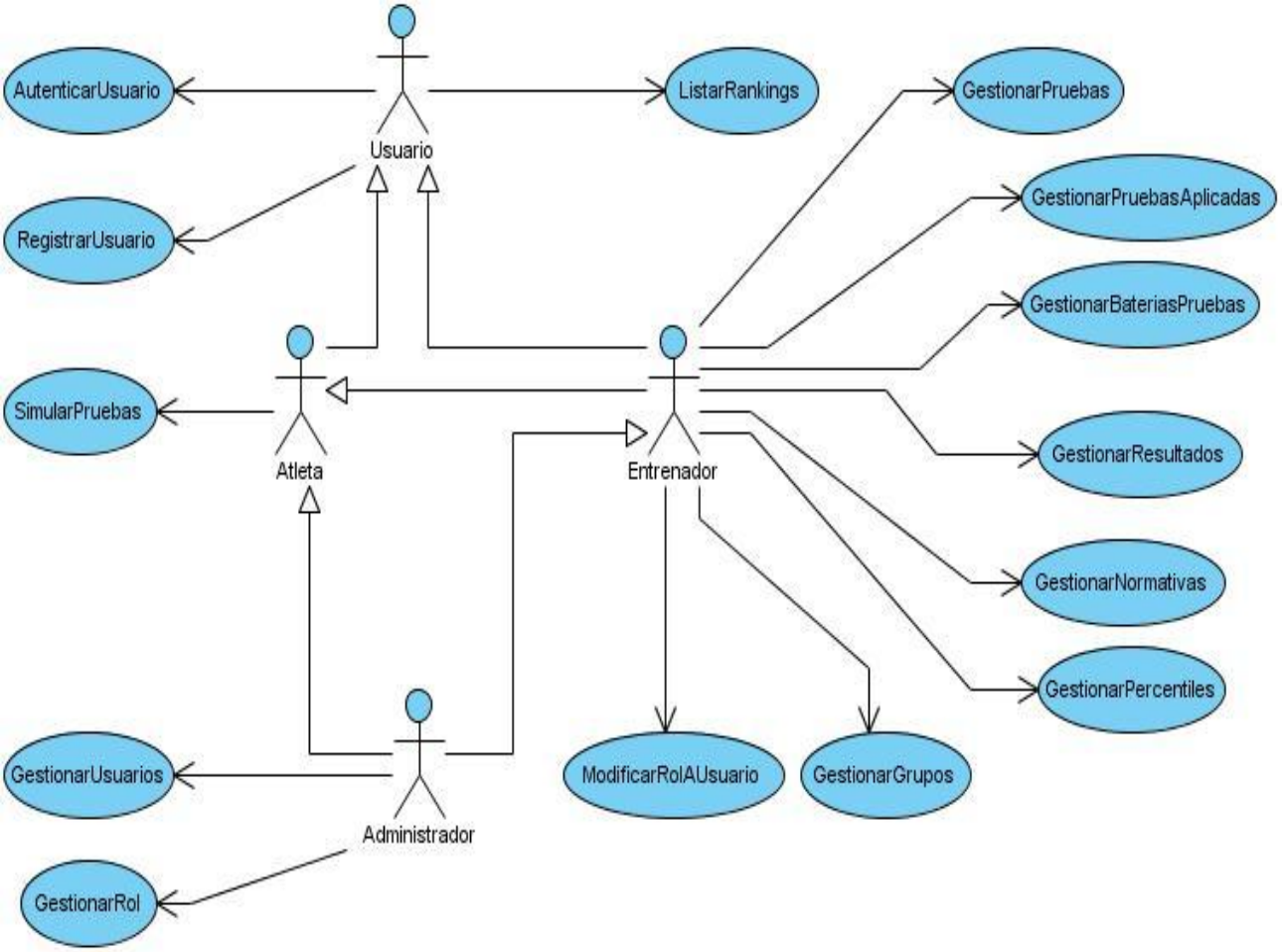

Figura 6. Diagrama de Caso de Uso del Sistema.

2.4.4 Descripciones de Casos de Uso del Sistema

A continuación se muestra la descripción de caso de uso “Gestionar Normativa”, por ser uno de los casos de usos de más importancia en la aplicación. Para consultar las demás descripciones de caso de uso referirse a los ***Anexos***.

Tabla 3. Descripción de caso de uso “Gestionar normativas”.

Caso de Uso	**Gestionar Normativas.**
Actores	Entrenador, administrador.
Resumen	El caso de uso inicia cuando los actores deciden crear, modificar o eliminar

	alguna normativa.
Referencias	RF 17, RF 18, RF 19, RF 20.
Precondiciones	Los actores tienen que estar autenticados previamente.
Flujo normal de eventos	
Acción del Actor	**Acción del Sistema**
1. Selecciona la opción gestionar normativa.	2. Muestra una interfaz con una lista de normativas existentes, además, da la posibilidad a escoger entre varias secciones: **Sección 1:** Crear normativa. **Sección 2:** Modificar normativa. **Sección 3:** Eliminar normativa.
Sección 1: "Crear normativa". Flujo normal de eventos	
Acción del Actor	**Acción del sistema**
1. El usuario selecciona la opción crear normativa.	2. Muestra una interfaz con dos secciones: **Sección 1.1:** Crear normativa de manera manual. **Sección 1.2:** Crear normativa de forma automática.
Sección 1.1: "Crear normativa de manera manual". Flujo normal de eventos	
Acción del Actor	**Acción del sistema**
	1. El sistema muestra los campos nombre, sexo, y valores, para que el usuario llene.

2. Pone el nombre de la normativa. 3. Selecciona el sexo. 4. Introduce los valores para la normativa. 5. Presiona el botón “Crear”.	
	6. Valida el nombre entrado. 7. Crea la normativa.
Flujos alternos	
	6.1 En caso de existir una normativa con el mismo nombre, el sistema muestra un mensaje informándoselo al actor. 6.2 En caso de que los valores estén introducidos de forma incorrecta, el sistema muestra un mensaje informándoselo al actor.
Pos-condiciones	Es creada la nueva normativa.
Sección 1.1: “Crear normativa de forma automático”. Flujo normal de eventos	
Acción del Actor	**Acción del sistema**
	1. El sistema muestra los campos nombre, sexo, percentil y prueba aplicada.
2. Pone el nombre de la normativa. 3. Selecciona el sexo. 4. Selecciona la normativa. 5. Selecciona la prueba aplicada. 6. Presiona el botón “Crear”.	

	7. Valida el nombre entrado. 8. Crea la normativa.
Flujos alternos	
	1.1 En caso de no existir algún percentil se deshabilita la opción de crear una normativa informándole al actor que debe tener creado previamente algún percentil. 7.1 En caso de existir una normativa con el mismo nombre, muestra mensaje informándoselo al actor y retorna al paso uno.
Pos-condiciones	Es creada la nueva normativa.
Sección 2: “Modificar normativa”. Flujo normal de eventos	
Acción del Actor	**Acción del sistema**
1. El usuario selecciona la opción modificar normativa.	2. Muestra una interfaz con dos secciones: **Sección 1.1:** Modificar normativa de manera manual. **Sección 1.2:** Modificar normativa de forma automática.
Sección 1.1: “Modificar normativa de manera manual”. Flujo normal de eventos	
Acción del Actor	**Acción del sistema**
	1. El sistema muestra los campos nombre, sexo, y valores, para que el usuario modifique.

2. Modifica los valores que desee 3. Presiona el botón “Modificar”.	
	4. Valida los datos entrados. 5. Modifica la normativa.
Flujos alternos	
	4.1 En caso de existir una normativa con el mismo nombre, el sistema muestra un mensaje informándoselo al actor. 4.2 En caso de que los valores estén introducidos de forma incorrecta, el sistema muestra un mensaje informándoselo al actor y retornar al paso 1.
Pos-condiciones	Es modificada la normativa.
Sección 1.1: “Modificar normativa de forma automático”. Flujo normal de eventos	
Acción del Actor	**Acción del sistema**
	1. El sistema muestra los campos nombre, sexo, percentil y prueba aplicada.
2. Modifica los valores que desee modificar. 3. Presiona el botón “Modificar”.	
	4. Valida los datos entrados. 5. Crea la normativa.
Flujos alternos	

	4.1 En caso de que los valores estén introducidos de forma incorrecta, el sistema muestra un mensaje informándoselo al actor y retornar al paso 1.
Pos-condiciones	Es modificada la normativa.
Sección 3: “Eliminar normativa”. Flujo normal de eventos	
Acción del Actor	**Acción del sistema**
	1. Muestra una interfaz con los datos de la normativa seleccionada.
2. Presiona el botón “Eliminar”.	3. Muestra mensaje de confirmar la eliminación.
4. Confirma la eliminación de la normativa.	5. El sistema elimina la normativa.
Flujos alternos	
	3.1 En caso de no confirmar la eliminación regresar al paso 1.
Pos-condiciones	Es eliminada la normativa.

Para consultar las demás descripciones de uso, remitirse a los anexos, del 1 al 13.

Conclusiones del capítulo

En el capítulo desarrollado se argumentan los motivos de la realización del Diagrama de Clases del Dominio para lograr un entendimiento mayor de los procesos del negocio. En un estudio paralelo se efectuó el sondeo del sistema, permitiendo la definición de los requisitos del sistema, siendo 40 funcionales y ocho no funcionales, dando como resultado la identificación de 14 casos de usos. Se definieron los actores que intervienen en el sistema y se diseño el Diagrama de Casos de Usos del Sistema. Se detallaron las acciones que realizan los casos de usos en las descripciones textuales de estos.

Capítulo 3. Diseño del sistema

Introducción

Se obtienen los artefactos del flujo de trabajo de diseño, etapa fundamental en el desarrollo de una aplicación. Se describen los patrones de arquitectura y diseño utilizados. Se generan los diagramas de clases para la conformación del sistema, así como la relación de este con la base de datos.

3.1 Patrones

Los patrones de arquitectura expresan el esquema fundamental de organización para sistemas de aplicaciones informáticas. Los patrones de diseño ayudan a materializar esa arquitectura. Un patrón describe un problema que se produce frecuentemente y las pautas para solucionarlo. Se basan en la práctica y aunque la base es aplicar las mismas pautas, la solución nunca es exactamente la misma.

3.1.1 Patrón Modelo – Vista – Controlador

El patrón Modelo-Vista-Controlador (MVC) divide una aplicación interactiva en tres áreas: procesamiento, salida y entrada. Para esto, utiliza las siguientes abstracciones:

- **Modelo**: Encapsula los datos y las funcionalidades. Es independiente de cualquier representación de salida o comportamiento de entrada.
- **Vista**: Muestra la información al usuario. Obtiene los datos del modelo. Pueden existir múltiples vistas del modelo. Cada vista tiene asociado un componente controlador.
- **Controlador**: Reciben las entradas, usualmente como eventos que codifican los movimientos o pulsación de botones del ratón, pulsaciones de teclas, entre otros eventos.

Las Vistas y los Controladores conforman la interfaz de usuario. Un mecanismo de propagación de cambios asegura la consistencia entre la interfaz y el modelo. La separación del modelo de los componentes vista y del controlador permite tener múltiples vistas del mismo modelo. Si el usuario cambia el modelo a través del controlador de una vista, todas las otras vistas dependientes deben reflejar los cambios. Por lo tanto, el modelo notifica a todas las vistas siempre que sus datos cambien. Las vistas, en cambio, recuperan los nuevos datos del modelo y actualizan la información que muestran al usuario (35).

Se evidencia el uso de este patrón en la aplicación al tener de manera separada la vista del controlador, así como las entidades de las ultimas mencionadas, todos estando en paquetes separados.

3.1.2 Patrón *Table Data Gateway*

Mezclar consultas o procedimientos de acceso a la base de datos en la lógica de la aplicación puede causar serios problemas. El patrón *Table Data Gateway* propone una solución para evitar esos errores, utilizando un objeto, que actúa como una puerta de entrada a una tabla dentro de la base de datos. Una instancia de este objeto se ocupa de todas las filas de la tabla. Su principal ventaja es la capacidad de encapsular el acceso a los datos, teniendo una clase entidad como objeto por cada tabla, la cual contiene los métodos para obtener los datos y actualizarlos (36).

En el paquete de las entidades existe una clase correspondiente a cada tabla en la base de datos. De esta manera se evita tener consultas y procedimientos de acceso en el paquete de las clases controladoras.

3.2 Modelo de diseño

Es un modelo de objetos que describe las realizaciones de los casos de usos, sirve como abstracción del modelo de implementación. La particularidad del diseño es modelar el sistema, es decir, es cómo cumple sus objetivos el sistema. Es una actividad esencial de entrada para la implementación.

3.3 Diagramas de clases del diseño

Un diagrama de clases del diseño presenta las clases del sistema con sus relaciones estructurales y de herencia. En el caso de aplicaciones web, las relaciones que existen entre las páginas se representan en el diagrama de clases del diseño, donde cada página representa una clase lógica del sistema.

A continuación se aprecia una tabla con las terminologías utilizadas en los diagramas de clases del diseño para un mejor entendimiento de los mismos:

Tabla 4. Estereotipos del Diagrama de Diseño.

Clases	Estereotipos	Función
Client Page [CP] (en español Página Cliente).		Representa una clase que interactúa con el usuario, su función es visualizar, interactuar y mostrar lo que el usuario necesita.
Server Page [SP] (en español Página Servidora).		Representa una clase para el acceso a datos, su principal función es construir la CP.
Form (en español Formulario).		Es una clase que interactúa directamente con el usuario y su principal función es enviar los datos entrados por el usuario a la SP.

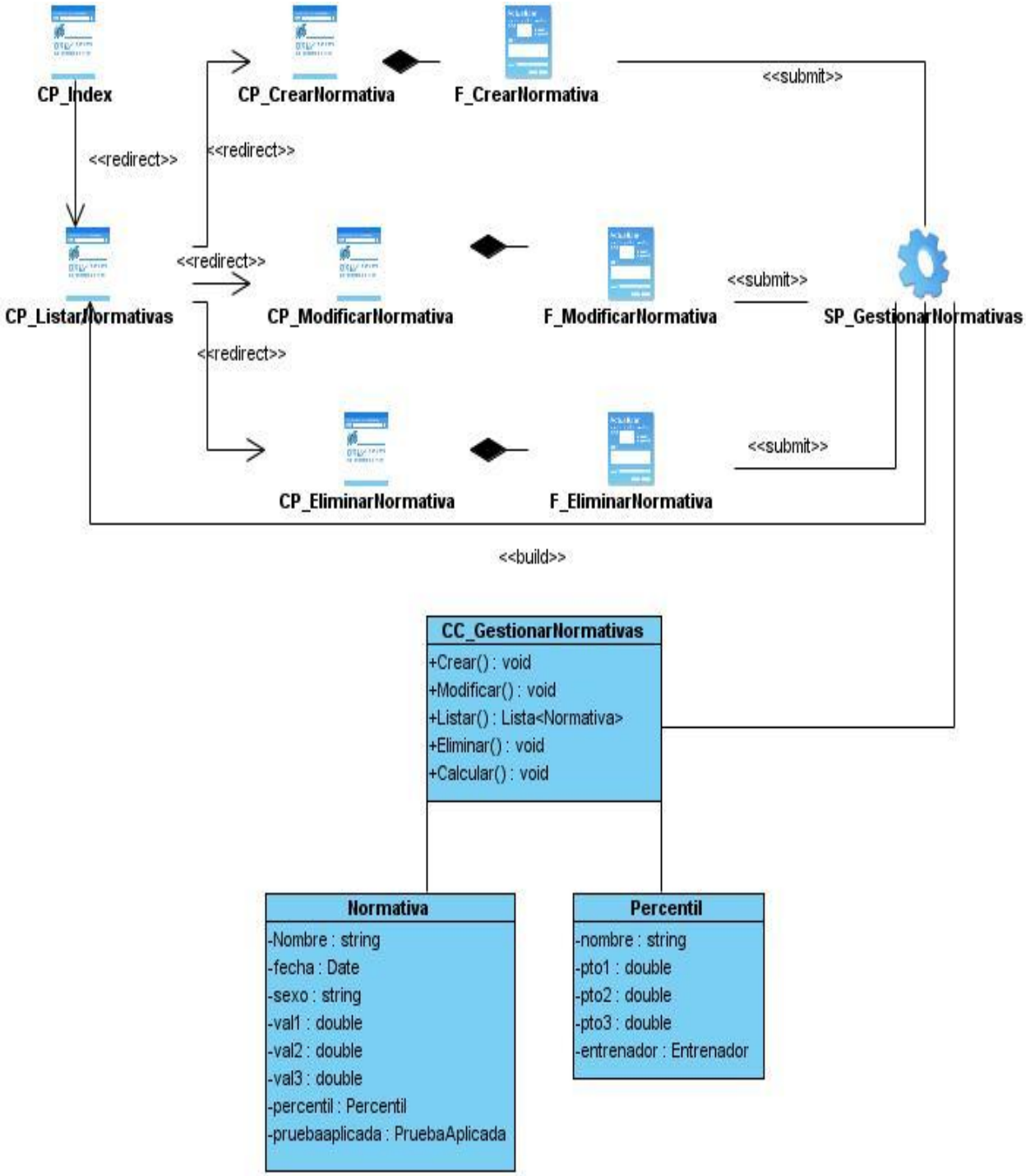

Figura 7. Diagrama de Clases del Diseño “Gestionar Normativas”.

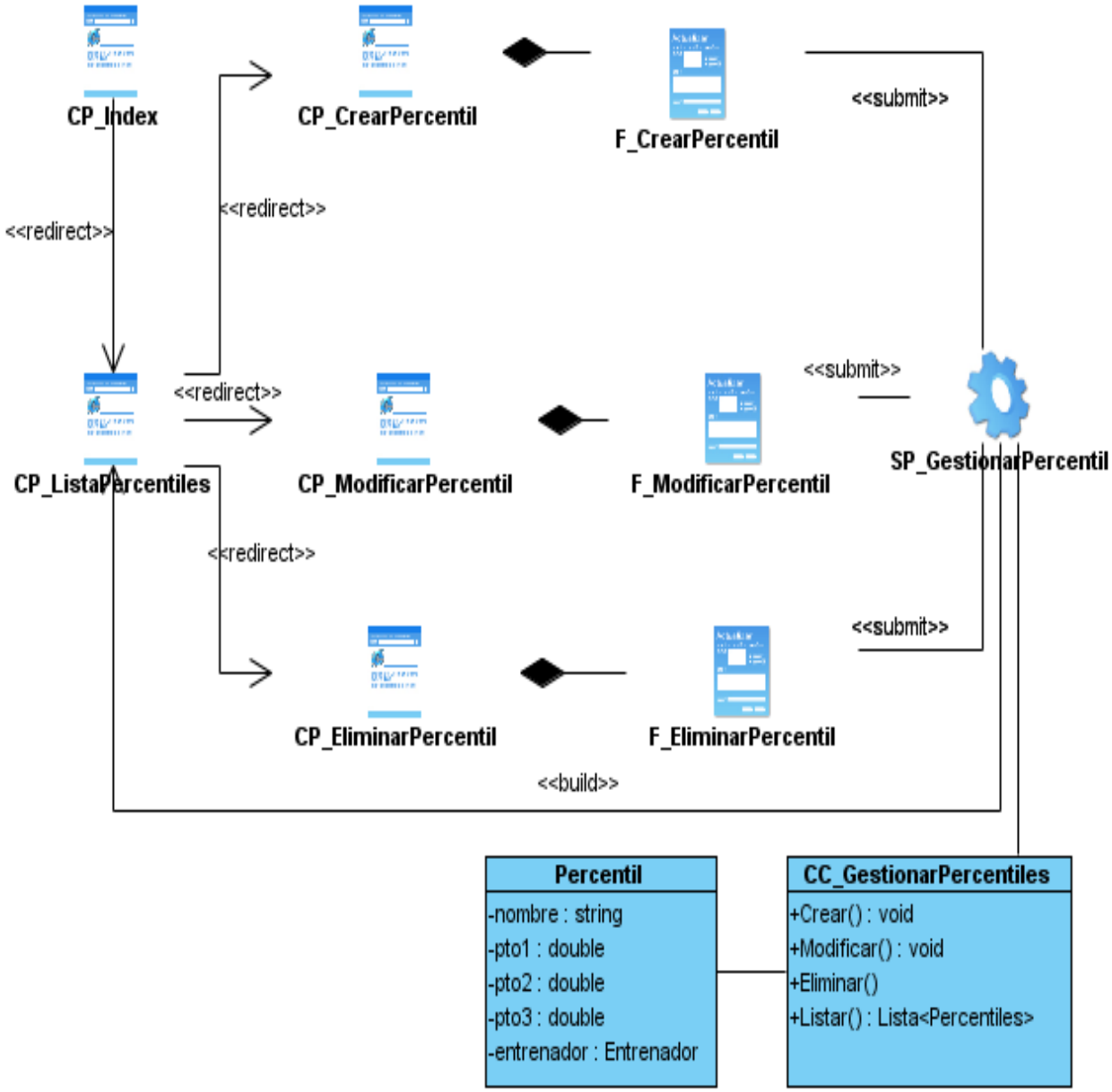

Figura 8. Diagrama de Clases del Diseño “Gestionar Percentiles”.

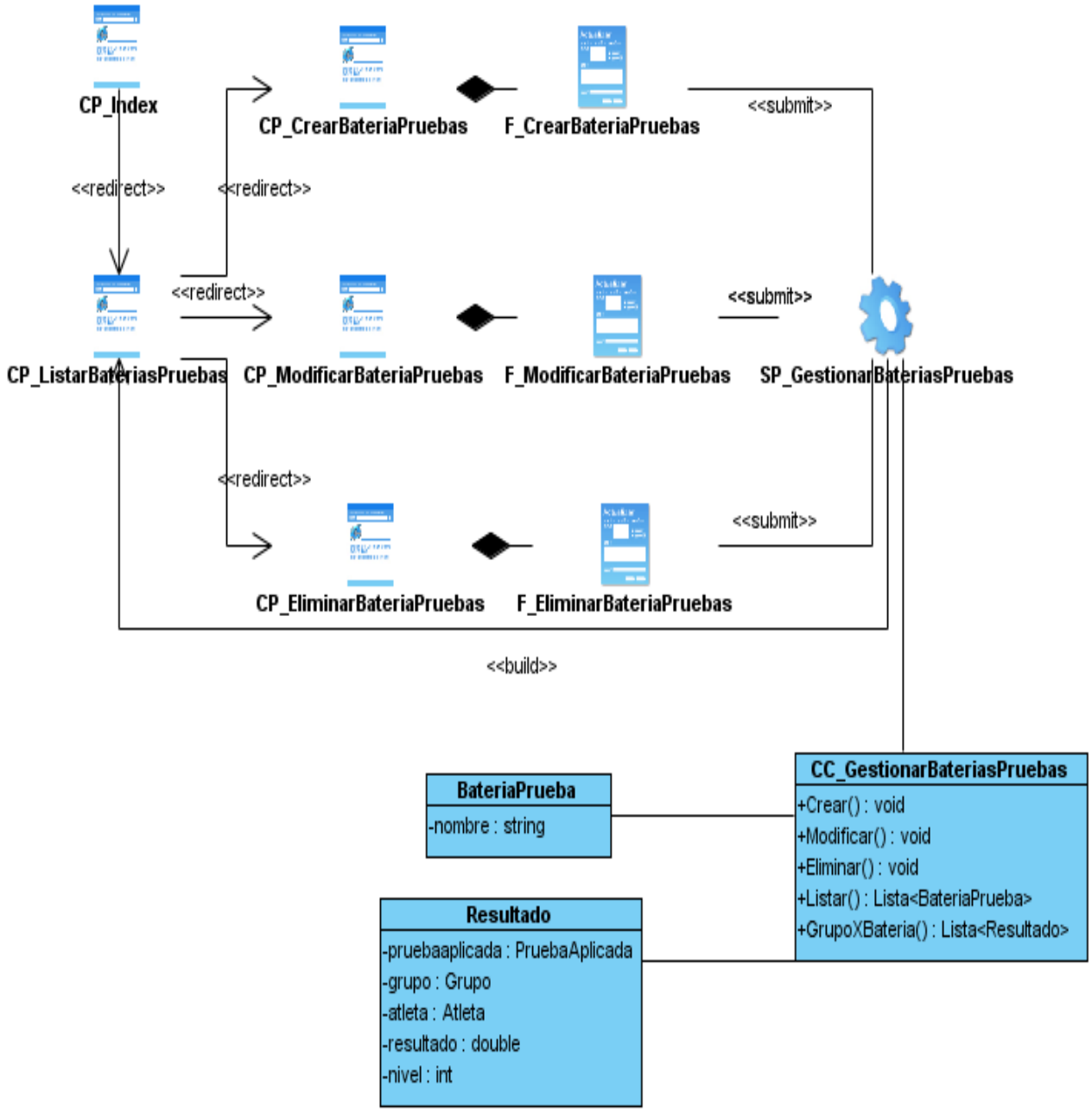

Figura 9. Diagrama de Clases del Diseño “Gestionar Baterías de Pruebas”.

Para consultar los demás diagramas del diseño, remitirse a los ***Anexos***, desde el 16 al 27.

3.4 Diagrama de Clases Persistentes

La persistencia de los datos es la capacidad que tienen de mantener su valor en el espacio y en el tiempo; lo cual está dado por el almacenamiento físico de la información de la clase, para la copia de seguridad en caso del fracaso del sistema, o para el intercambio de información.

A continuación se muestra el diagrama de clases persistentes:

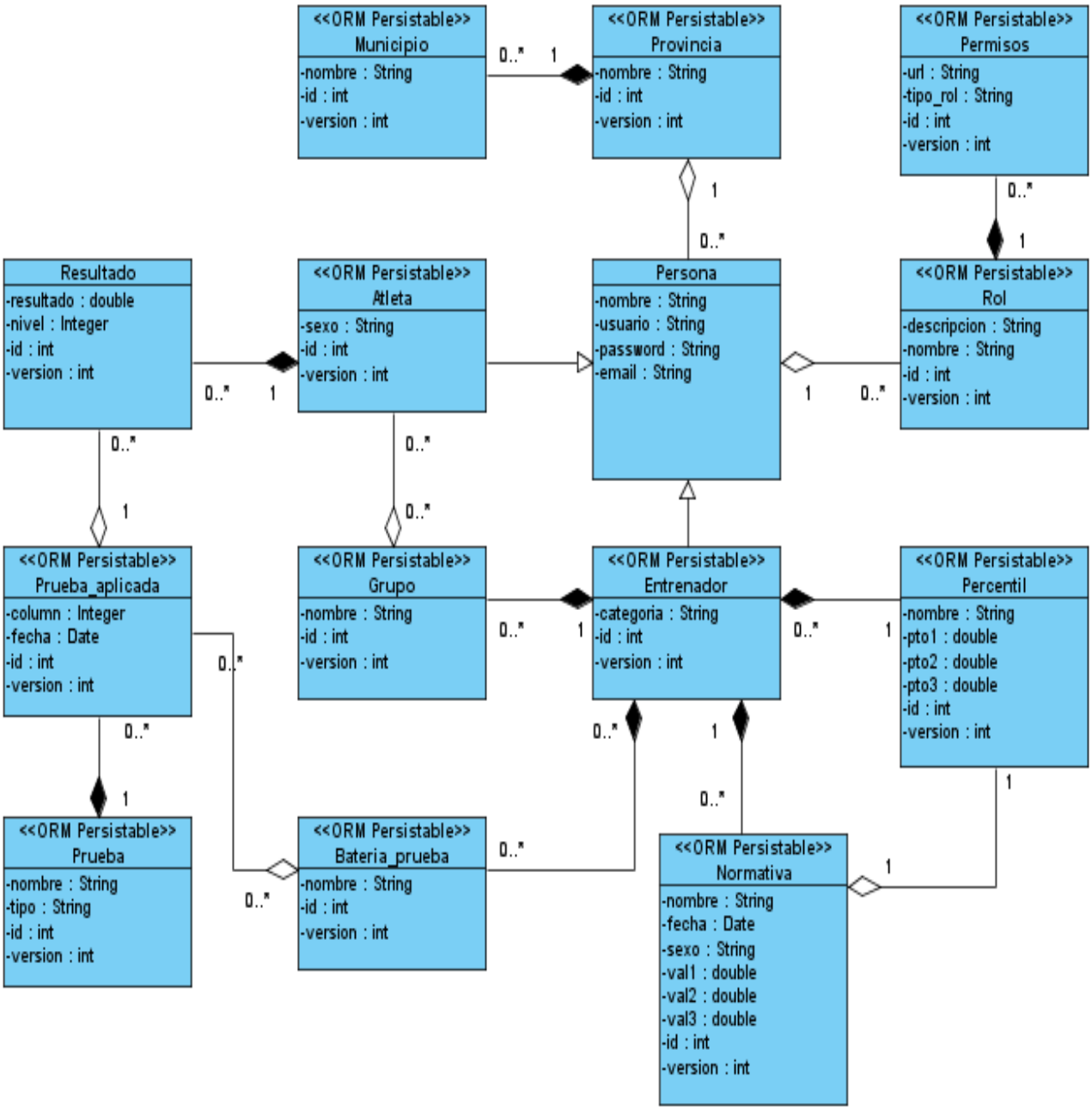

Figura 10. Diagrama de clases persistentes.

3.5 Modelo Entidad Relación

A partir del diagrama de clases persistentes se obtuvo el siguiente Modelo Entidad Relación:

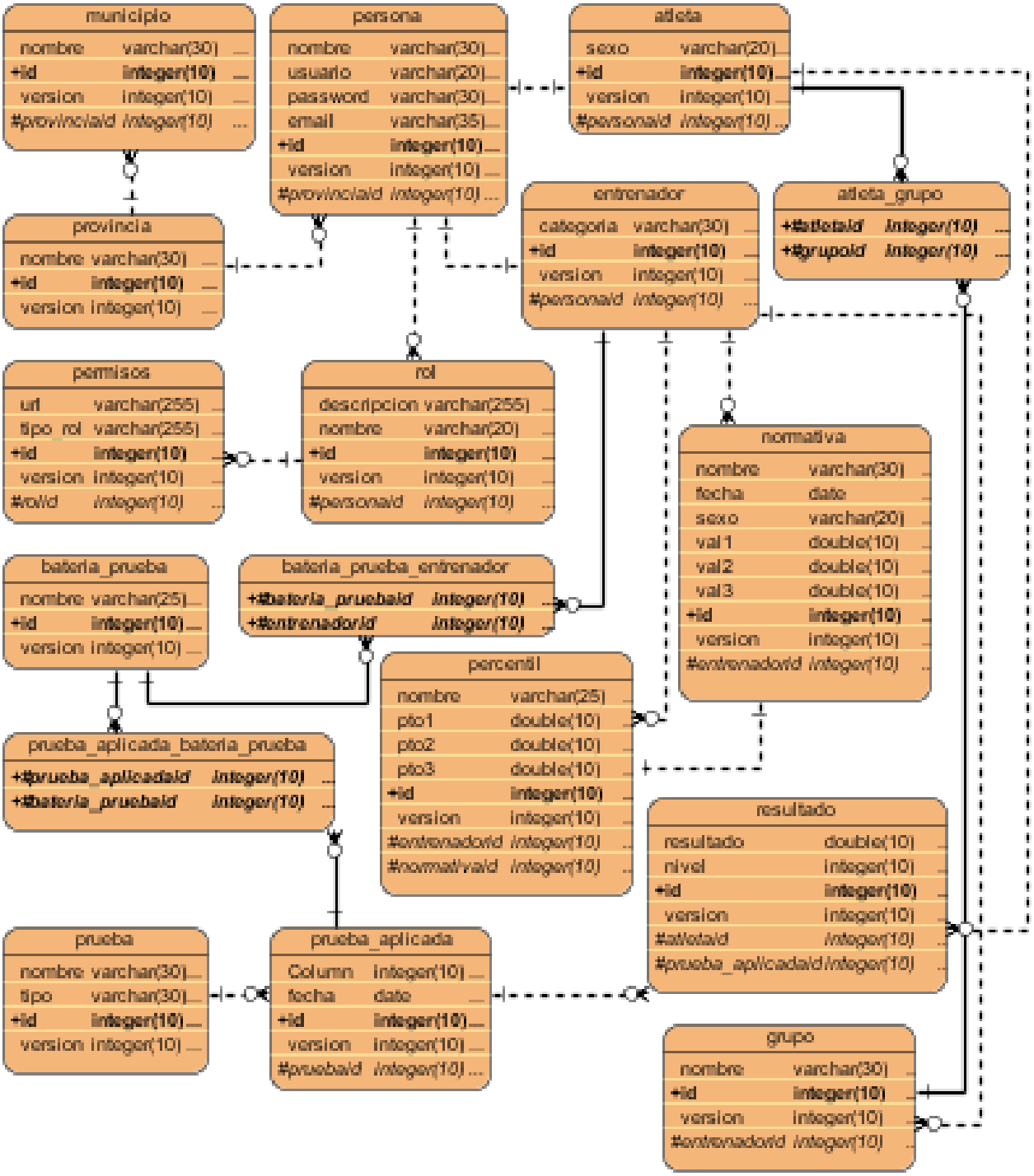

Figura 11. Modelo Entidad – Relación.

Conclusiones del capítulo

En este capítulo se incorporaron los artefactos pertenecientes al flujo de trabajo de Diseño. Se definieron los patrones, tanto de arquitecturas como de diseño, los cuales permiten una mayor organización en la aplicación. Se realizaron los diferentes diagramas definidos para este flujo por la metodología utilizada, quedando plasmado cómo se debe implementar el sistema.

Capítulo 4. Implementación y prueba

Introducción

Los artefactos generados en el capítulo anterior permiten una introducción a la fase de implementación. Es aquí donde se generan los diagramas de componentes y paquetes de implementación. Se propone el estándar de codificación utilizado. También se tratan las pruebas realizadas para comprobar las funcionalidades del sistema, quedando construido completamente el sistema al terminar el capítulo.

4.1 Estándar de codificación

Los estándares de codificación son reglas que se siguen para la escritura del código fuente. Un estilo de programación homogéneo en un proyecto permite que los participantes puedan entender el código en menos tiempo y por tanto sea de fácil mantenimiento.

4.1.1 Indentación y espacios en blanco

Alineación

Se utiliza la tecla *TAB* (cuatro espacios) como unidad de alineación. Solo habrá una línea en blanco antes de comenzar los métodos, en ningún otro lugar habrá un sendero en blanco.

Longitud de línea

Se evita usar líneas mayores de 80 caracteres de longitud, cuando una expresión no quepa en una sola línea, fraccionar siguiendo tres principios generales:

1. Después de una coma.
2. Después de un operador.
3. Alinear al mismo nivel que la línea anterior.

4.1.2 Convenciones de nombres

Los nombres que se utilizan son comprensibles y descriptivos, facilitando el entendimiento del código generado. Se utiliza la notación *CamelCase* (37) (notación camello), que consiste en escribir los identificadores con la primera letra de cada palabra en mayúsculas y el resto en minúscula. De esta notación se usa específicamente la variante *dromedaryCase* (37), que no es más que escribir la

primera palabra en minúscula y de existir más de dos palabras, escribir el resto con la primera letra en mayúscula.

Esta notación se les aplica a todas las convenciones de nombre, desde una variable, hasta el nombre de un método, exceptuando el nombre de las clases que sí se escriben en *CamelCase*.

4.1.3 Comentarios

Un comentario se hace cuando las líneas de códigos de los ficheros son complejas o requieren de algún tipo de aclaración. Los ficheros pueden tener tipos de comentarios, en bloques y en líneas simples. Los comentarios en bloque son aquellos que estarán arriba de los métodos explicando la función del mismo, estos comentarios están destinados a ser procesados por javadoc (38). Ejemplo:

```
7   /*
8   Para realizar esto hay que tener en cuenta la prueba y el tipo de prueba
9   si es de tiempo o de marca (tiempo: la lista es ordenada de menor a mayor)
10  (marca:es ordenada la lista de mayor a menor)... hay que tener en cuenta el
11  sexo del atleta asi como el sexo de la normativa a aplicar, la cantidad de
12  atletas que hicieron la prueba y los ptos criticos del percentil utilizado
13  */
14  def calcula = {
15  }
```

Figura 12. Comentario en bloque.

Nunca existirá un comentario en bloque dentro de un método. Dentro de las funcionalidades solo se podrá poner comentarios de líneas y son específicos para alguna línea de código compleja, o alguna variable que se argumente el motivo de su creación. Ejemplo:

```
14  def create = {
15      def personInstance = new Person()  //se crea una nueva instancia de persona
16      personInstance.properties = params
17      return [personInstance: personInstance]
18  }
```

Figura 13. Comentario en línea.

4.2 Diagrama de componentes

Representa como un sistema esta divido en componentes mostrando las dependencias entre estos. Se visualiza con facilidad la estructura general de la aplicación y su comportamiento. Facilita el entendimiento del modelo de implementación, pues presenta los componentes lógicos de la aplicación, ya sean código fuente, librerías, bibliotecas, binarios o ejecutables.

En este diagrama se diferencian las vistas, los controladores y las clases de dominio, las cuales son instancias de tablas de la base de datos.

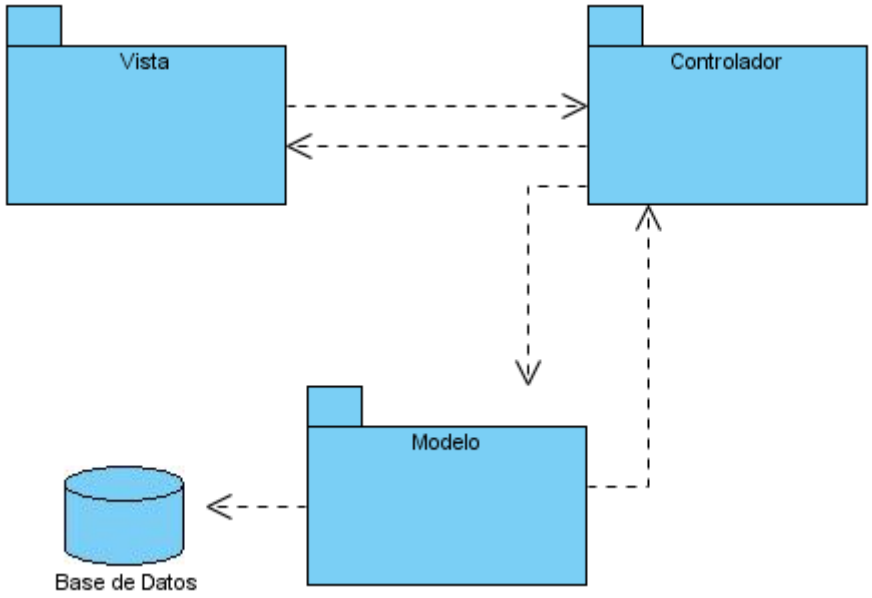

Figura 14. Diagrama de componentes general.

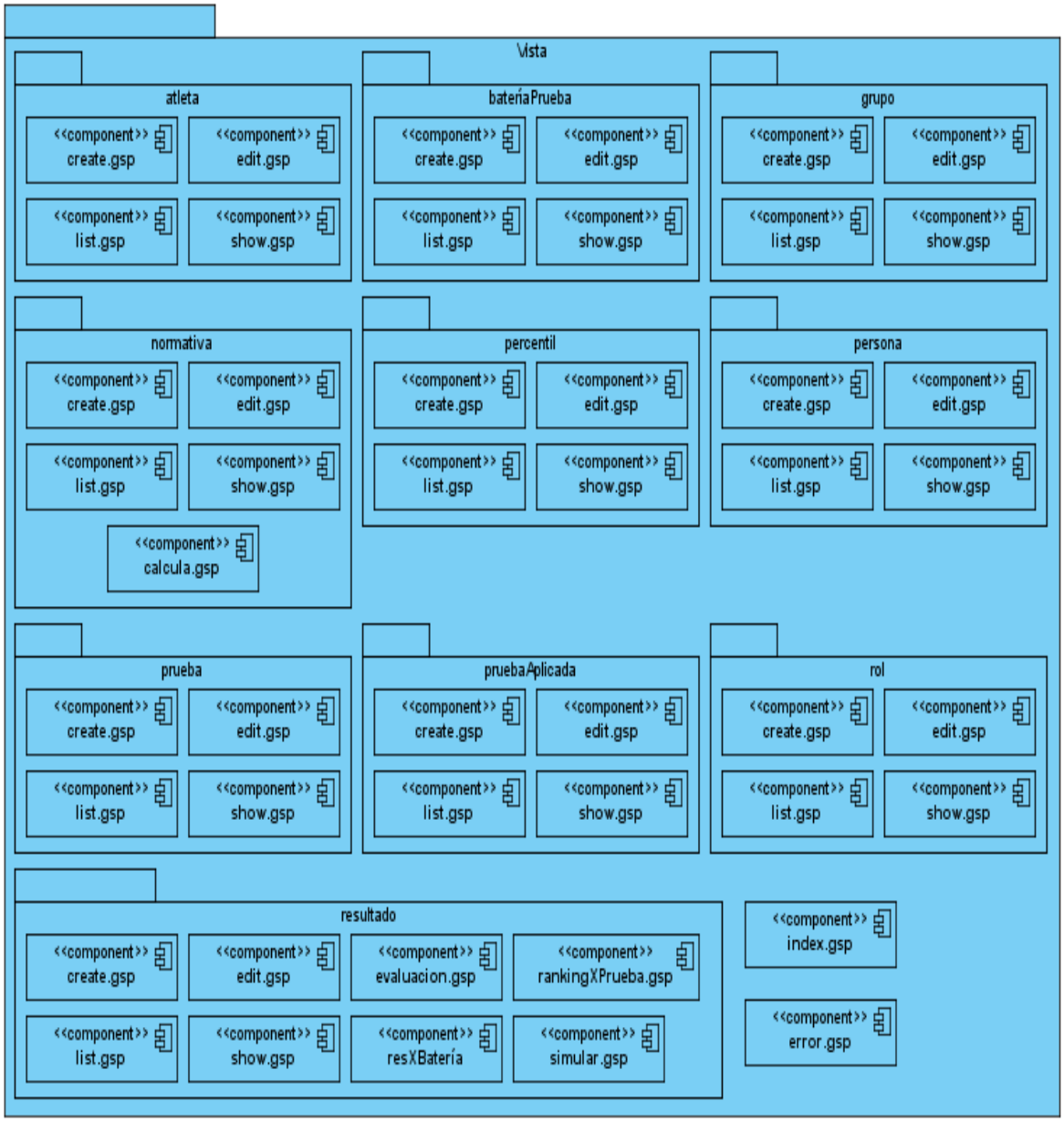

Figura 15. Diagrama de componentes del paquete Vista.

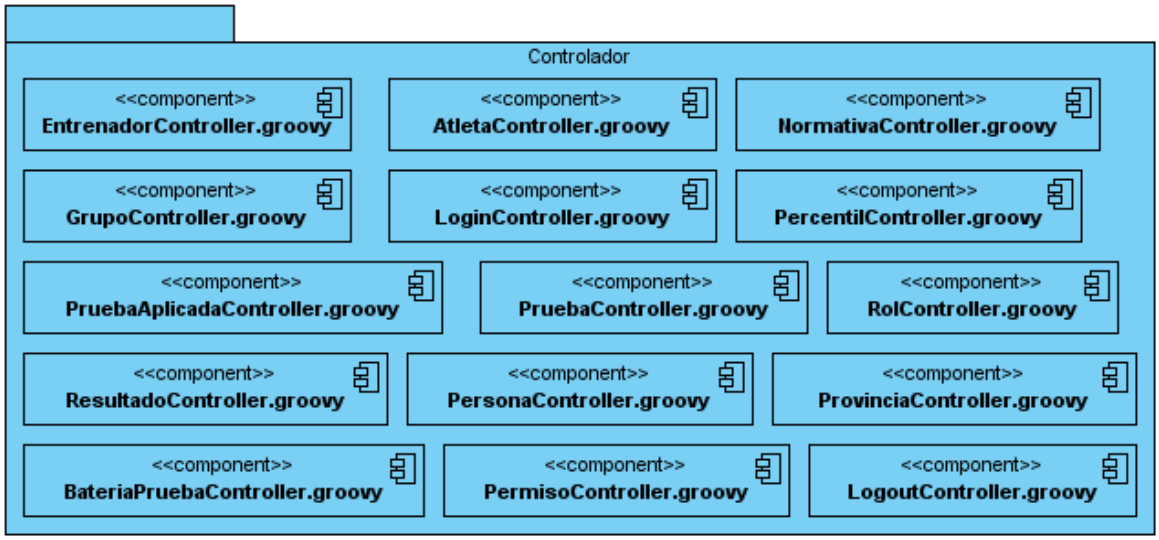

Figura 16. Diagrama de componentes del paquete Controlador.

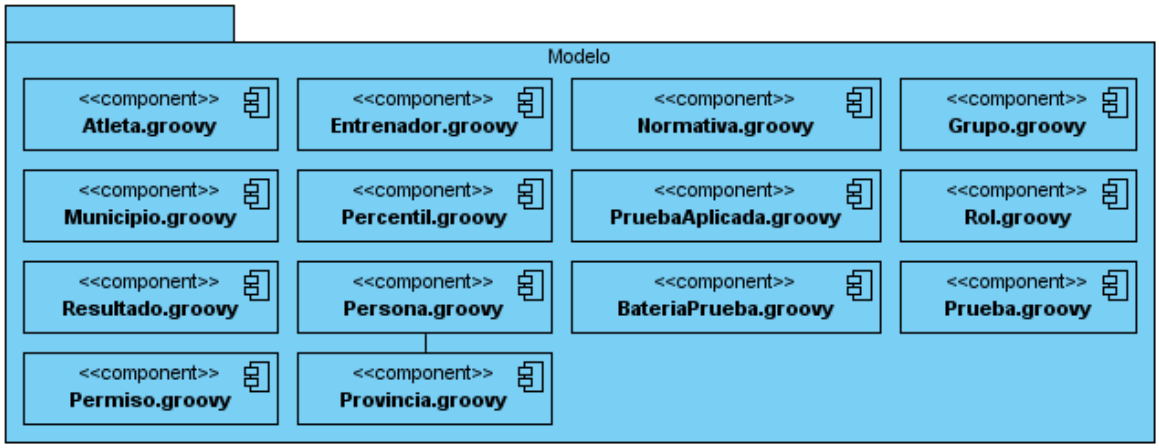

Figura 17. Diagrama de componentes del paquete Modelo.

4.3 Diagrama de despliegue

Muestra las relaciones físicas entre los componentes físicos y las aplicaciones. Es representado por un grafo de nodos mostrando las conexiones de comunicación entre estos. Estos nodos pueden contener instancias de componentes, siendo un recurso de ejecución, tal como una computadora, un dispositivo o memoria.

Se representa la computadora cliente, la cual se comunica con el terminal servidor por el protocolo de transferencia de hipertexto (HTTP por sus siglas en inglés) que es donde está montado el sistema

sobre un servidor Web, que mantiene comunicación con la base de datos por la *Java Database Connectivity* (JDBC).

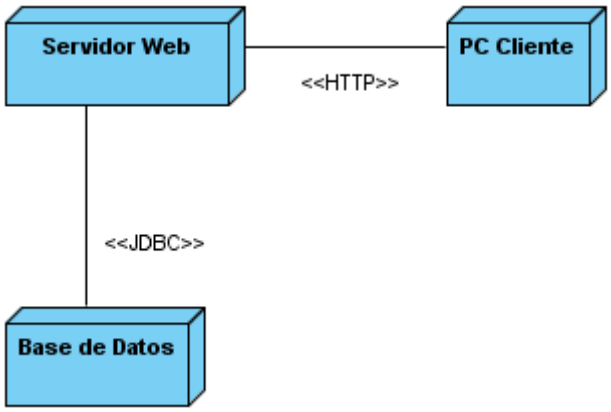

Figura 18. Diagrama de despliegue.

4.4 Modelo de Prueba

En la construcción de la aplicación en cada una de sus fases es importante la realización de pruebas, fundamentalmente en la fase de implementación. Estas pruebas son realizadas para medir la calidad del sistema y descubrir errores de cualquier tipo que pueda tener la aplicación.

4.4.1 Casos de prueba de caja negra

Se refiere a las pruebas que llevan a cabo sobre la interfaz gráfica del sistema. La prueba se limita a la entrada de datos para estudiar las respuestas del sistema, sin preocuparse de lo que esté pasando dentro de la aplicación.

Gestionar Normativa:

1. **Descripción general**

El caso de uso inicia cuando los actores deciden crear, modificar o eliminar alguna normativa.

2. **Condiciones de ejecución**

Los actores tienen que estar autenticados previamente.

3. Secciones a probar en el Caso de Uso Gestionar Normativa.

Nombre de la sección	Escenarios de la sección	Descripción de la funcionalidad
SC 1: Crear normativa de forma automática	EC 1.1: Introducir nombre de la normativa, selecciona el sexo, selecciona el percentil, selecciona la prueba aplicada sobre la cual va a ser calculada	El sistema verifica los datos y crea la normativa
	EC 1.2: Introducir nombre existente de normativa	El sistema verifica los datos y muestra un mensaje de error indicando que el nombre
	EC 1.3: Introducir valores incorrectos	El sistema verifica los datos y muestra un mensaje de error indicando que los datos no son correctos
SC 2: Crear normativa de forma manual	EC 2.1: Introducir nombre de la normativa, selecciona el sexo, introducir los valores	El sistema verifica los datos y crea la normativa
	EC 2.2: Introducir nombre existente de normativa	El sistema verifica los datos y muestra un mensaje de error indicando que el nombre ya existe.
	EC 2.3: Introducir valores incorrectos	El sistema verifica los datos y muestra un mensaje de error indicando que los datos no son correctos

SC 3: Modificar normativa	EC 3.1: Cambia el nombre de la normativa, selecciona el sexo, modifica los valores	El sistema verifica los datos y modifica la normativa
	EC 3.2: Introducir nombre existente de normativa	El sistema verifica los datos y muestra un mensaje de error indicando que el nombre ya existe.
	EC 3.3: Introducir valores incorrectos	El sistema verifica los datos y muestra un mensaje de error indicando que los datos no son correctos

4. Descripción de las variables.

No	Nombre del campo	Clasificación	Valor Nulo	Descripción
1	Nombre	Campo de texto	No.	Solo letras, solo números o una combinación de ambos.
2	Sexo	Lista desplegables	No	Muestra una lista de sexo
3	Percentil	Lista desplegable	No	Muestra una lista con los percentiles existentes creados por el usuario autenticado
4	Prueba Aplicada	Lista desplegable	No	Muestra una lista con las pruebas aplicadas existentes creadas por el usuario autenticado

5	Valor 1	Campo de texto	No	Solo números
6	Valor 2	Campo de texto	No	Solo números
7	Valor 3	Campo de texto	No	Solo números

Gestionar Pruebas:

1. **Descripción general**

El caso de uso inicia cuando los actores deciden crear, modificar o eliminar alguna prueba.

2. **Condiciones de ejecución**

Los actores tienen que estar autenticados previamente.

3. **Secciones a probar en el Caso de Uso Gestionar Normativa.**

Nombre de la sección	Escenarios de la sección	Descripción de la funcionalidad
SC 1. Crear prueba	EC 1.1: Introducir nombre de la prueba, selecciona el tipo de prueba	El sistema verifica los datos y crea la prueba

	EC 1.2: Introducir nombre existente de prueba	El sistema verifica los datos y muestra un mensaje de error indicando que el nombre ya existe
	EC 1.3: Introducir valores incorrectos	El sistema verifica los datos y muestra un mensaje de error indicando que los datos no son correctos
SC 2. Modificar prueba	EC 2.1: Modificar nombre de la prueba, selecciona el tipo de prueba	El sistema verifica los datos y crea la prueba
	EC 2.2: Modificar nombre existente de prueba	El sistema verifica los datos y muestra un mensaje de error indicando que el nombre ya existe
	EC 2.3: Introducir valores incorrectos	El sistema verifica los datos y muestra un mensaje de error indicando que los datos no son correctos

4. **Descripción de las variables.**

No	Nombre del campo	Clasificación	Valor Nulo	Descripción
1	Nombre	Campo de texto	No.	Solo letras, solo números o una combinación de ambos.
2	Tipo	Lista desplegable	No	Muestra una lista con los tipos de pruebas existentes

Gestionar Baterías de Pruebas:

1. **Descripción general**

El caso de uso inicia cuando los actores deciden crear, modificar o eliminar alguna batería de prueba.

2. Condiciones de ejecución

Los actores tienen que estar autenticados previamente.

3. Secciones a probar en el Caso de Uso Gestionar Normativa.

Nombre de la sección	Escenarios de la sección	Descripción de la funcionalidad
SC 1. Crear batería de prueba	EC 1.1: Introducir nombre de la batería de prueba, selecciona las pruebas aplicadas que va a tener la batería	El sistema verifica los datos y crea la prueba
	EC 1.2: Introducir nombre existente de batería de prueba	El sistema verifica los datos y muestra un mensaje de error indicando que el nombre ya existe
	EC 1.3: Introducir valores incorrectos	El sistema verifica los datos y muestra un mensaje de error indicando que los datos no son correctos
SC 2. Modificar batería de prueba	EC 2.1: Modificar nombre de la batería de prueba, selecciona el tipo de prueba	El sistema verifica los datos y crea la batería de prueba
	EC 2.2: Modificar nombre existente de batería de prueba	El sistema verifica los datos y muestra un mensaje de error indicando que el nombre ya existe
	EC 2.3: Introducir valores incorrectos	El sistema verifica los datos y muestra un mensaje de error indicando que los datos no son correctos

4. Descripción de las variables.

No	Nombre del campo	Clasificación	Valor Nulo	Descripción
1	Nombre	Campo de texto	No.	Solo letras, solo números o una combinación de ambos.
2	Pruebas aplicadas	Lista	No	Muestra una lista con las pruebas aplicadas existentes creadas por el usuario autenticado

Conclusiones del capítulo

Como resultado de este capítulo se realizó una estandarización del código. Se obtuvo el diagrama de componentes y el diagrama de despliegue del sistema. Se le realizaron pruebas de cajas negras a las más importantes interfaces visuales del sistema.

Conclusiones

Una vez concluida la investigación e implementación de la aplicación web que gestione el proceso de los test físicos en los entrenamientos deportivos, se ha dado cumplimiento a los objetivos trazados y se ha obtenido los siguientes resultados:

- Se realizó una valoración sobre las posibles metodologías de desarrollo de aplicaciones informáticas y herramientas a utilizar.
- Se realizó el diseño de un sistema informático que permita la gestión de test físicos en los entrenamientos deportivos.
- Se implementó el sistema diseñado.
- Se validó la aplicación probando la eficacia del mismo, con una buena satisfacción por parte del cliente.

Recomendaciones

- Continuar la gestión de resultados deportivos a partir de la creación de un plan de entrenamiento.
- Incluir información deportiva en la aplicación.

Referencias Bibliográficas

1. Real Academia Española. [Online] http://rae.es.

2. Nuevas tecnologías de la información y la comuncación (TICs). Ruiz, Antonio Munuera. La Habana : s.n., 2006.

3. campusdeportivo. [Online] http://www.campusdeportivo.com/cas/disenador.asp.

4. Informática & Deportes. [Online] http://www.entrenar.com.ar./empresa.php.

5. Guerra, MsC. Edecio Pérez. EfDeportes. [Online] http://www.efdeportes.com/efd126/las-pruebas-o-tests-en-el-deporte.htm.

6. python. [Online] http://www.python.org/.

7. Django en español. [Online] http://django.es/.

8. Calleja, Manuel Arias. Estandares de configuración.

9. Ivar Jacobson, Grady Booch, James Rumbaugh. El Proceso Unificado de Desarrollo de Software. La Habana : Félix Varela.

10. José H. Canós, Patricio Letelier y Mª Carmen Penadés. Métodologías Ágiles en el Desarrollo de Software. Valencia : s.n.

11. DSDM Consortium. [Online] http://www.dsdm.org/.

12. OpenUP. [Online] http://ndpsoftware.com/OpenUpBasic/index.htm.

13. Cheesman, John and Daniels, John. UML components, a simple process for specifying component based software. Londres : s.n.

14. Ruth Priscila Landeros Gómez, Jorge Luis Del Juncal Huerta. Herramientas Case. México : s.n.

15. IBM. [Online] http://www-142.ibm.com/software/products/es/es/enterprise/.

16. Visual paradimg UML. Visual paradimg. [Online] [Cited: 05 veinte, 2011.] http://www.visual-paradigm.com/product/vpuml/.

17. CODEBOX. [Online] http://www.codebox.es/glosario.

18. Sitio Oficial de APS .Net. [Online] http://www.asp.net/.

19. François Zaninotto, Fabien Potencier. Symfony 1.0, la guía definitiva. s.l. : Apress.

20. Observatorio de Grails. [Online] http://observatoriodegrails.com/.

21. Página oficial de Groovy. [Online] http://groovy.org.es/.

22. SpringHispano. [Online] http://www.springhispano.org/.

23. Hibernate. [Online] http://www.hibernate.org/.

24. ASP.net. [Online] http://www.asp.net/ajax.

25. Ruby on Rails. [Online] http://www.rubyonrails.org.es/.

26. Dojo toolkit. [Online] http://dojotoolkit.org/.

27. w3schools. [Online] http://www.w3schools.com/js/js_intro.asp.

28. Quiñones, Ernesto. PostgreSQLPE. [Online] http://www.postgresql.org.pe/articles/introduccion_a_postgresql.pdf.

29. Development MySQL. [Online] http://dev.mysql.com/doc/refman/5.0/es/features.html.

30. Garrido, Jesús Manuel Montero. Plataforma Eclipse. Introducción Técnica.

31. NetBeans. [Online] http://netbeans.org/.

32. JetBRAINS. [Online] http://www.jetbrains.com/idea/.

33. MSDN Magazine. [Online] http://msdn.microsoft.com/es-es/magazine/ee236415.aspx.

34. EcuRed. [Online] http://www.ecured.cu/index.php/Patrones_de_Casos_de_Uso.

35. Welicki, León. msdn. [Online] http://msdn.microsoft.com/es-es/library/bb972251.aspx.

36. Garzás, Javier. Sistemas de información.

37. Notación y estilos en programación. [Online] http://jmm.iriux.com/articulos/notacion-estilo-programacion.html.

38. Javadoc 5.0 Tool. [Online] http://download.oracle.com/javase/1.5.0/docs/guide/javadoc/index.html.

Bibliografía

1. Calleja, Manuel Arias. Estandares de configuración.

2. Cheesman, John and Daniels, John. UML components, a simple process for specifying component based software. Londres : s.n.

3. François Zaninotto, Fabien Potencier. Symfony 1.0, la guía definitiva. s.l. : Apress.

4. Garrido, Jesús Manuel Montero. Plataforma Eclipse. Introducción Técnica.

5. Garzás, Javier. Sistemas de información.

6. Ivar Jacobson, Grady Booch, James Rumbaugh. El Proceso Unificado de Desarrollo de Software. La Habana : Félix Varela.

7. José H. Canós, Patricio Letelier y Mª Carmen Penadés. Métodologías Ágiles en el Desarrollo de Software. Valencia : s.n.

8. Ruiz, Antonio Munuera. Nuevas tecnologías de la información y la comuncación (TICs). La Habana : s.n., 2006.

9. Ruth Priscila Landeros Gómez, Jorge Luis Del Juncal Huerta. Herramientas Case. México : s.n.