Universidad de las Ciencias Informáticas

Facultad 1

Implementación del módulo Registro y Control Docente del sistema de Gestión Universitaria

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

Autores: Lidiana Hernández Legrá

Yudileidis Peña Carballosa

Tutores: Ing. Ailec Granda Dihigo

Ing. Ariel Enrique Novo Rijo

Cotutora: Ing. Zenia Veigas Chkout

Ciudad de La Habana, 23 de junio del 2010

"Año 52 de la Revolución"

Declaración de autoría

Declaramos que somos las únicas autoras de esta investigación y autorizamos a la facultad 1 de la Universidad de las Ciencias Informáticas a hacer uso de la misma en su beneficio. Para que así conste firmamos la presente a los ____ días del mes de _____ del año _____.

Autores:

_____ _____

Lidiana Hernández Legrá Yudileidis Peña Carballosa

Tutores:

_____ _____

Ing. Ailec Granda Dihigo Ing. Ariel Enrique Novo Rijo

Cotutora:

_____

Ing. Zenia Veigas Chkout

Dedicatoria

A mi abuela Celia: que aunque la extrañe muchísimo y no esté a mi lado fue y será siempre una madre para mí.

A mi viejito Luis: por haber hecho de mí lo que soy hoy.

A mi tía Marilín: que aunque ya no está conmigo, se que estaría orgullosa de mí.

A mis padres Mirkian y Clodomiro: por traerme al mundo y ayudarme a crecer.

A mi familia: por estar ahí cuando los necesité y ser una pieza fundamental para que llegara hasta aquí.

Yudileidis.

A mis sobrinas Keila y Karen,

el consejo:
"Aunque el estudio te abrume,
prosigue siempre adelante
que una mujer ignorante
es una flor sin perfume."

Lidiana.

Agradecimientos

Agradecida estoy a Dios porque todo se lo debo a Él. Gracias Señor, porque eres mi mejor amigo, en todo tiempo has estado a mi lado y me has enseñado que contigo nada es imposible.

Gracias a mis padres, ha sido tanta la entrega... gracias por ayudarme en mis estudios universitarios, por el apoyo, por sus oraciones, por estar siempre ahí. Este logro no es mío, es de ustedes. Los amo.

Gracias Samuel, Lemuel, por cada consejo, por "aguantarme" todos estos años. Por todo lo que me dieron. Esta victoria, es de ustedes también. Los quiero mucho.

Dianly e Hildamary, ustedes son mis hermanas, gracias por poder contar con personas tan ejemplares como ustedes.

Gracias a Dios, por mi amor, Emilio... Gracias mi vida, por todo lo que me has dado, por ser quien eres para mí, por tu apoyo, consejo, amor….

Al resto de mi familia, soy feliz por tenerlos.

A mis amigos, que tanto me dieron, que revisaron la ortografía, que programaron conmigo, que me dieron un consejo, una palabra de aliento y aclararon dudas: Annia, José Alejandro, Aymé, Reiniel, Yisel Hernández, Yisel Niño, Laritza, Iran, Luis Daniel y Adly.

A Alejandro y Juan Emilio, quienes me impulsaron a estudiar en la UCI.

Fueron muchos los profesores que fomentaron mi formación profesional, a todos ellos, gracias, incluso a los de niveles inferiores.

Muchas gracias a todos los que en el anonimato, estuvieron orando por mí, en cada fase de mi carrera universitaria, a mis hermanos de la UCI y de Iglesia "Restauración". Gracias a todos los amigos que siempre estuvieron conmigo.

Gracias por la UCI, porque fue instrumento de perfección para mí.

A mi compañera de tesis, por estar a mi lado todo este tiempo.

Gracias a nuestros tutores y cotutora. Sin ustedes no habríamos alcanzado la cumbre de esta tesis.

Muchas gracias a nuestro Jefe de Departamento Ronny, por toda la ayuda y el apoyo que nos brindó en el desarrollo de esta tesis.

Lidiana.

Agradecimientos

A la UCI por haberme hecho crecer estos 5 años y formarme como profesional.

A mi compañera de tesis por "soportarme", por su apoyo y ayuda incondicional. A mis tutores por su compromiso ante esta tarea.

Gracias a todas las amistades que tuve estos años que siempre me apoyaron, en especial a LAS FEAS (Daily, Yalina, Yeny), Eddy, Imer, Alain, Lorián y Alexis.

A Gabriela y Sailín por tolerar mis "malcriadeces" y ser incondicionales conmigo. A mis hermanos del alma Leandro y Leonel por confiar siempre en mí y estar cuando más los necesité.

A mi abuelo Luis por ser como un padre, mi guía y orgullo, quien nunca descansó para que llegara hasta aquí.

A mi mamá y mi papá por darme su amor y confiar en mí.

A mis tíos Marilín, Alfonso, Felipe, Richard e Ita por ayudarme y ser parte fundamental de mi crecimiento.

Yusnelis, Yusnalia, Marilín y Carmen María, ustedes son como hermanas para mí, gracias por su apoyo, no saben cuánto me alegra saber que puedo contar con ustedes incondicionalmente.

Gracias a mi abuela Isabel por estar siempre para mí. Haydee fuiste una madre más en estos 5 años, gracias por tu amor.

Aunque hace poco que conozco a Marcelo le agradezco que no haya perdido la fe en mí y cuidó de mi mamá cuando yo no estaba.

Siempre me apoyó y estuvo a mi lado brindándome aliento para que siguiera adelante, gracias a mi novio Isidro, por tanto amor.

Gracias a Yasel Rafael por ser una pieza fundamental en los primeros años de mi carrera y apoyarme en cada paso que daba.

Gracias a todas las amistades de mi compañera de tesis (Annia, Jose Alejandro, Yisel Niño, Laritza, Irán) y a su novio Emilio por cada revisión que hicieron al documento, por cada línea de código que ayudaron a implementar y por sus consejos.

A todos los profesores que de una forma u otra impulsaron mí formación educativa desde la primera vez que escribí mi nombre hasta el último día en la UCI.

Yudileidis.

Resumen

La presente investigación pretende mejorar la gestión de los procesos de control docente de la Universidad de las Ciencias Informáticas (UCI). Actualmente estos procesos se gestionan parcialmente con el sistema Akademos que fue desarrollado con tecnologías propietarias o privativas, el mismo no se ajusta al proceso de migración a *software* libre de la universidad y del país, además presenta algunas limitaciones pues no gestiona la existencia de varias carreras en la universidad, como se prevé en la UCI.

Por esta razón surge la necesidad de desarrollar una aplicación informática más flexible, configurable y fácil de usar, realizándose la implementación del módulo Registro y Control Docente del sistema de Gestión Universitaria en la UCI, el cual cuenta con la gestión de más de una carrera, la gestión de bonificaciones, asistencia, evaluaciones, grupos docentes y nomencladores de estados de grupo docente y estados de asistencia.

Para el desarrollo del sistema se estudiaron ampliamente las tecnologías, herramientas y lenguajes de programación que facilitan el desarrollo de aplicaciones como soluciones informáticas, aunque su selección, incluyendo la metodología a utilizar, fue definida por el grupo de trabajo del departamento de Gestión Universitaria de la UCI. Se describen y documentan los procesos que fueron implementados. Finalmente, se hicieron pruebas para verificar la calidad, confiabilidad y disponibilidad, con el objetivo de obtener la aceptación de los clientes y usuarios finales.

Palabras claves

gestión, proceso de control docente, aplicación informática, módulo, Registro y Control Docente, implementación.

Índice

Tablas

Figuras

Introducción

El creciente desarrollo de la informática a nivel mundial proporciona beneficios a todos los sectores de la humanidad. En el mundo se han desarrollado diversos sistemas de gestión académica como soluciones informáticas para el sector docente, ya que en un centro educacional la gestión académica constituye una actividad fundamental, que permite la organización y control de todo el proceso docente. Estos sistemas gestionan las actividades que están vinculadas con el ambiente docente de los estudiantes, regularizando el tránsito de los mismos desde su ingreso a una carrera universitaria hasta su egreso.

En Cuba existen sistemas que automatizan el proceso de gestión académica pero que a su vez presentan deficiencias al no incluir todas las funcionalidades necesarias para la completa gestión académica. Algunos de estos programas no son desarrollados con tecnologías *OpenSource* (de código abierto) o no involucran a todos los actores implicados en el proceso de control docente, siendo limitados por estas razones.

En la Universidad de las Ciencias Informáticas (UCI) se hace necesario gestionar toda la información docente, principalmente los resultados académicos alcanzados por los estudiantes. Dicha información es manejada por las secretarias docentes y profesores de cada facultad haciendo uso del sistema automatizado para la gestión académica Akademos, el cual fue desplegado en la UCI desde el curso 2004/2005.

Este sistema se desarrolló teniendo en cuenta que el dinamismo de gestión académica constituye la principal fuente de riesgo para un sistema que intente automatizarlo. Permite la matrícula, registrar evaluaciones, registrar asistencias, registrar premios, gestión de plan de estudio, registro de profesor, realizar reportes y gestionar expediente.

Akademos es poco flexible para registrar evaluaciones y tiene un mal diseño de la base de datos, provocando que no se puedan hacer modificaciones al sistema. Es desarrollado con herramientas propietarias, lo cual no cumple con los planes de migración a *software* libre que se han trazado la UCI y el país, no posee una documentación consistente que respalde su desarrollo, limitando así las posibilidades de retroalimentación de nuevos miembros del equipo y dificultando en gran manera la realización de nuevas iteraciones e incluso la posibilidad de parchear[*] el sistema. Además, actualmente la UCI tiene la característica de contar con sólo una carrera de estudio, la de Ingeniería en Ciencias Informáticas, y este sistema fue diseñado de acuerdo a esta particularidad, por

[*] Ver Glosario de términos

lo que no soluciona las necesidades de un centro multidisciplinario, siendo ésta la visión futura de la universidad.

Siendo analizada la situación problémica, surge la necesidad de implementar el módulo Registro y Control Docente del sistema de Gestión Universitaria. Este módulo tiene como novedades la gestión de más de una carrera en la universidad, gestión de bajas, traslados y licencias, siendo parte del subsistema de Gestión de Pregrado el cual gestiona los procesos de formación de los estudiantes en la universidad, que incluyen la gestión de personal, la gestión de carreras y el registro y control docente de todas las actividades de formación. Cuenta además con los módulos Diseño de Carrera, Personal y Secretaría, Tesis y Títulos, Planificación Docente y Reportes, Gráficas y Estadísticas.

Por lo anteriormente expuesto se formula el siguiente **problema científico:** ¿cómo mejorar la gestión de los procesos de control docente de la Universidad de las Ciencias Informáticas?

El **objeto de estudio** está constituido por los procesos de gestión docente, cuyo **campo de acción** se enmarca en los procesos de gestión docente en la UCI.

Como **objetivo general** se plantea implementar el módulo Registro y Control Docente del sistema de Gestión Universitaria para la gestión de los procesos de control docente en la UCI, derivándose los siguientes **objetivos específicos:**

- Analizar las técnicas de programación, plataformas, herramientas y metodologías* de desarrollo de *software*[*] que faciliten la implementación del módulo Registro y Control Docente.
- Desarrollar el módulo Registro y Control Docente del sistema de Gestión Universitaria.
- Validar la propuesta de solución.

Se **defiende la idea** de que, contándose en la UCI con el módulo implementado Registro y Control Docente del sistema de Gestión Universitaria, se podrá mejorar la gestión de los procesos de control docente.

Para el cumplimiento organizado y bien distribuido de los objetivos se establecen las siguientes **tareas** a cumplir:

- Desarrollar habilidades en el uso del lenguaje de programación PHP*, del *framework** CodeIgniter y del gestor de base de datos PostgreSQL.
- Describir las tecnologías y herramientas que serán usadas en la realización de la propuesta de solución.
- Implementar la propuesta de solución.
- Realizar pruebas de aceptación.

[*] Ver Glosario de términos

Métodos Teóricos

- **Histórico-lógico**

Está asociado a la realización de un análisis de las distintas fases lógicas sucesivas por las que ha transcurrido el control docente para su progreso en un determinado período de tiempo y de esta manera se pueda entender cómo funcionan éstas.

Posibilita un mejor análisis histórico de los procesos de control docente, es decir, permite analizar la trayectoria de los procesos, desde su desenvolvimiento hasta las conexiones históricas más importantes.

- **Analítico-sintético**

Este método permite resolver los rasgos que caracterizan al control docente, examinando los distintos documentos que están vinculados con este tema y comprender específicamente las características más relevantes de éstos.

Posibilita el análisis de los procesos de registro y control docente para determinar con exactitud cómo éste funciona. Como resultado, se toman todas las características principales para lograr la implementación del sistema.

Métodos Empíricos

- **Observación**

Este método permite investigar los procesos externamente sin tener que llegar a la esencia de los mismos, lo que ayuda al planteamiento del problema científico, además de permitir conocer bien el proceso delimitado como objeto de estudio, que contribuye a tener un conocimiento más detallado de lo que se quiere, lo que hace falta hacer y cómo hay que hacerlo.

- **Revisión de documentos**

La revisión de documentos se realiza en pos de la determinación del estado del arte del objeto de investigación.

El contenido de la investigación se encuentra distribuido en el documento de la siguiente manera:

- **Capítulo 1. Fundamentación teórica**

Este capítulo incluye un estado del arte de las distintas técnicas de programación que existen a nivel internacional, nacional y de la universidad. Se hace referencia a los lenguajes de programación

usados para la creación de aplicaciones *web*[*], así como algunas de las herramientas tales como sistemas gestores de base de datos, *frameworks* y entornos de desarrollo, además de argumentar brevemente acerca de las plataformas y metodologías de desarrollo de *software*.

- **Capítulo 2. Descripción y análisis de la solución propuesta**

Teniendo en cuenta el análisis entregado por los analistas del proyecto, el cual rige la implementación de la propuesta de solución, se describen las clases que modelan la solución y las principales funcionalidades de cada una de ellas. Se establecen los estándares de codificación a utilizar y se detallan Tareas de Ingeniería útiles para el desarrollo del módulo.

- **Capítulo 3. Validación de la solución propuesta**

En este capítulo se aborda el tema referente a la realización de pruebas, se analizan algunas de las características que presentan las pruebas unitarias y de aceptación, siendo éstas últimas, las empleadas para la validación de la solución propuesta por medio de los casos de prueba realizados al sistema.

[*] Ver Glosario de términos

Capítulo 1: Fundamentación teórica

1.1 Introducción

Para la creación de sistemas informáticos, tanto *web* como de escritorio, se emplean en el mundo y en la Universidad de las Ciencias Informáticas (UCI) diferentes técnicas y lenguajes de programación. Intervienen además algunas herramientas como *frameworks*, gestores de base de datos, entornos de desarrollo y metodologías de desarrollo de *software* sobre determinadas plataformas. Antes de crear una aplicación es muy importante conocer bien las herramientas y tecnologías para su realización, teniendo en cuenta las características particulares de cada uno de estos aspectos y las prestaciones del sistema que se quiere realizar.

1.2 Técnicas de programación

En la actualidad la computadora ha ganado un espacio considerable como herramienta de trabajo, ya que su utilidad se debe precisamente a su capacidad de efectuar grandes cálculos que no pueden realizar los seres humanos. La potencia de las máquinas actuales es tal, que inclusive los cálculos pequeños, por su tamaño, escapan al poder de la capacidad limitada del hombre.

Sin embargo, se debe organizar el cálculo de manera tal que la limitación del ser humano sea suficiente para asegurar que es establecido el efecto deseado. Esta organización incluye la creación de programas por medio de lenguajes de programación.

Los lenguajes, más allá del propósito para el que fueron creados, pueden diferenciarse por la forma de trabajo que presentan al programador, ofreciendo diversas formas de ver y pensar un programa antes de escribirlo. Así comenzaron a surgir distintas técnicas de programación, cada una de ellas representada por una familia de lenguajes.

1.2.1 Programación estructurada

La programación estructurada es una metodología (un paradigma[*]) de programación basada en el concepto de la unidad y del alcance (la gama de la visión de los datos de una declaración ejecutable del código). Un programa estructurado se compone de unas o más unidades (o módulos) escrito por el programador o sacado de una librería; cada módulo se compone de uno o más procedimientos, también llamado una función, una rutina, un subprograma o un método, dependiendo del lenguaje de

[*] Ver Glosario de términos

programación. Es posible que éste tenga niveles múltiples o alcances, con métodos definidos dentro de otros procedimientos. Cada alcance puede contener las variables que no se pueden considerar en alcances externos.

La programación estructurada (PE), es un estilo de programación con el cual el programador elabora programas, cuya organización es la más clara posible, mediante el uso de tres estructuras básicas de control lógico, a saber: secuencia, selección e iteración.

La PE está compuesta por un conjunto de técnicas que han ido evolucionando y aumentando considerablemente la productividad del programa reduciendo el tiempo de depuración y mantenimiento del mismo. Utiliza un número limitado de estructuras de control, reduciendo así considerablemente los errores.

Esta técnica incorpora diseño descendente (top-down*), lo cual significa que el problema se descompone en etapas o estructuras jerárquicas; recursos abstractos (simplicidad), consiste en descomponer las acciones complejas en otras más simples capaces de ser resueltas con mayor facilidad; estructuras básicas, como las estructuras secuenciales, donde cada acción sigue a otra acción secuencialmente. La salida de una acción es la entrada de otra, estructuras selectivas, que en éstas se evalúan las condiciones y en función del resultado de las mismas se realizan unas acciones u otras y se utilizan expresiones lógicas, y por último las estructuras repetitivas que son secuencias de instrucciones que se repiten un número determinado de veces.

La PE tiene como ventaja que los programas son más fáciles de entender, se reduce la complejidad de las pruebas, aumenta la productividad del programador y los programas quedan mejor documentados internamente.

Un programa está estructurado si posee un único punto de entrada y sólo uno de salida, existen de “1 a n” caminos desde el principio hasta el fin del mismo y por último, que todas las instrucciones son ejecutables sin que aparezcan bucles infinitos.

La PE es Turing completa, lo cual quiere decir que todo algoritmo computable puede ser escrito en términos de programación estructurada.

El principal inconveniente de este método de programación, es que se obtiene un único bloque de programa, que cuando se hace demasiado grande puede resultar problemático su manejo. (Alvarez, 2010)

* Ver Glosario de términos

1.2.2 Programación procedimental

Es un paradigma de programación basado en el concepto de "llamado de procedimientos", también conocidos como rutinas, subrutinas, métodos o funciones que simplemente contienen series de pasos computacionales. Cualquier método puede ser llamado en cualquier punto durante la ejecución de un programa, incluyendo otros que sean externos o en él mismo.

Se fundamenta en una serie de descomposiciones sucesivas del problema inicial, y está inspirado en la técnica "Divide y Vencerás" que usaba el conquistador Alejandro Magno para derrotar a sus enemigos. Su utilización tiene muchos beneficios, entre los que se encuentran la facilidad en la escritura, lectura y comprensión de los programas y el permitir ahorrar espacio que de otro modo estaría ocupado por código duplicado.

Con la programación procedimental se pueden combinar las secuencias de instrucciones repetibles en un solo lugar. Una llamada de procedimiento se utiliza para invocarlo y después que la secuencia es procesada, el flujo de control regresa al punto de llamada para proseguir con la ejecución de la instrucción ubicada inmediatamente después del que hizo la llamada.

Con esta metodología el programa se divide en partes independientes, cada una de las cuales ejecuta una única actividad o tarea. Estas partes se analizan, se codifican separadas de las demás y se ponen aparte. Siguiendo un método ascendente o descendente de desarrollo se llegará a la descomposición final del problema en módulos en forma jerárquica. Las descomposiciones resultantes reciben luego el refinamiento progresivo del repertorio de instrucciones que van a formar parte de cada pieza del programa.

Si la tarea asignada a cada procedimiento es demasiado compleja, éste deberá descomponerse en otros más pequeños. Este proceso de subdivisión sucesiva continúa hasta que cada uno tenga solamente una tarea específica que realizar. Esta tarea puede ser entrada, salida, procesamiento de datos, control de otros métodos o una combinación de éstos.

Un procedimiento puede transferir temporalmente el control (bifurcación) a otro. Sin embargo, el que recibe este control debe eventualmente devolverlo al que lo envió. Los procedimientos son independientes en el sentido de que ninguno puede tener acceso directo a otro, excepto al módulo al que llama y a sus propios submódulos. No obstante, los resultados producidos por éstos pueden ser utilizados por cualquier otro cuando se transfiera el control a ellos.

Dado que los procedimientos son independientes, diferentes programadores pueden trabajar simultáneamente en disímiles partes del mismo programa. Esto reducirá el tiempo de diseño del algoritmo y posterior codificación. Además, se pueden modificar radicalmente sin afectar a los demás, incluso sin alterar su función principal.

Algunos lenguajes de programación procedimental son el C, Basic, COBOL, Matlab y Pascal.

La programación procedimental es rígida e inflexible, con ella hay pérdida excesiva de tiempo en la corrección de errores, la documentación es deficiente e insuficiente. (Díaz, 2010)

1.2.3 Programación modular

La programación modular es una generalización de la procedimental. Aquí los procedimientos con una funcionalidad común son agrupados en módulos separados. Un programa por consiguiente, ya no consiste solamente de una sección. Ahora está dividido en varias secciones más pequeñas que interactúan a través de llamadas a procedimientos y que integran el programa en su totalidad.

El programa principal coordina las llamadas a procedimientos en módulos separados y pasa los datos apropiados en forma de parámetros.

Cada módulo puede contener sus propios datos. Esto permite que cada uno maneje un estado interno que es modificado por las llamadas a procedimientos del mismo. Sin embargo, solamente hay un estado por módulo el cual existe cuando más una vez en todo el programa.

La programación modular cuenta con diferentes técnicas derivadas, como lo es el método descendente (*top-down*) también conocido como "de arriba abajo", que consiste en establecer una serie de niveles de mayor a menor complejidad que den solución al problema. Luego se crea una relación entre las etapas de la estructuración de forma que una etapa jerárquica y su inmediato inferior se relacionen mediante una interfaz claramente definida de entradas y salidas de información. Cuenta además con el método ascendente (*bottom-up*[*]) que se refiere a la identificación de aquellos procesos que necesitan computarizarse conforme vayan apareciendo, su análisis como sistema y su codificación, o bien, la adquisición de paquetes de *software* para satisfacer el problema inmediato.

Cuando la programación se realiza internamente y se hace un enfoque ascendente, es difícil llegar a integrar los subsistemas al grado tal de que el desempeño global sea fluido. Los problemas de integración entre los subsistemas son sumamente costosos y muchos de ellos no se solucionan hasta que la programación alcanza la fecha límite para la integración total del sistema. En esta fecha, ya se cuenta con muy poco tiempo, presupuesto o paciencia de los usuarios, como para corregir aquellas delicadas interfaces, que en un principio, se ignoran. (Díaz, 2010)

[*] Ver Glosario de términos

1.2.4 Programación concurrente

La concurrencia no es un término fácil de definir. Informalmente, un programa concurrente es uno que ejecuta más de una actividad simultáneamente. Sin embargo, en muchos casos esta simultaneidad es una ilusión.

En algunos sistemas concurrentes, las diferentes actividades son realizadas en diferentes CPU (*Central Processing Unit* o Unidad Central de Procesamiento) y la concurrencia es real (concurrencia física). En otros sistemas, sólo hay una CPU y las diferentes actividades se ejecutan secuencialmente siguiendo una estrategia de entrelazamiento basada en repartir el tiempo de ejecución de la CPU (concurrencia virtual).

En la programación concurrente se supone que hay un procesador utilizable por cada tarea; no se hace ninguna suposición de si el procesador será una unidad independiente para cada uno de ellos o si será una sola CPU que se comparte en el tiempo entre las tareas.

Independientemente de cómo se vaya a ejecutar realmente el programa, en un sistema monoprocesador o multiprocesador, el resultado debe ser el correcto. Por ello, se supone que existe un conjunto de instrucciones primitivas que son o bien parte del sistema operativo (SO) o bien parte de un lenguaje de programación, y cuya correcta implementación y corrección está garantizada por el sistema. (Drake, 2008)

1.2.5 Programación funcional

El paradigma de programación funcional es uno de los fundamentales entre los llamados de programación declarativa. Como tal, permite asociar los componentes de especificación y programación en las tareas de solución automática de problemas.

Los lenguajes funcionales ofrecen al programador un buen número de recursos expresivos que permiten resolver problemas complejos mediante programas pequeños y robustos. (Lucas, 2009)

La programación funcional apareció como un paradigma independiente a principios de la década del ´60. Su creación es debida a las necesidades de los investigadores en el campo de la inteligencia artificial y en sus campos secundarios del cálculo simbólico, pruebas de teoremas, sistemas basados en reglas y procesamiento del lenguaje natural. Estas necesidades no estaban cubiertas por los lenguajes imperativos de la época.

La característica principal de la programación funcional es que los cálculos se ven como una función matemática que hacen corresponder entradas y salidas. No hay noción de posición de memoria y por tanto, no hay necesidad de una instrucción de asignación. Los bucles se modelan a través de la recursividad ya que no hay manera de incrementar o disminuir el valor de una variable.

Como aspecto práctico casi todos los lenguajes funcionales soportan el concepto de variable, asignación y bucle. Estos elementos no forman parte del modelo funcional "puro".

Haskell es un lenguaje funcional puro, no es un lenguaje muy rápido, pero se prioriza el tiempo del programador sobre el tiempo de computación. (Universidad de Huelva, 2005)

1.2.6 Programación lógica

La programación lógica, junto con la funcional, forma parte de lo que se conoce como programación declarativa. En los lenguajes tradicionales, la programación consiste en indicar cómo resolver un problema mediante sentencias; en la programación lógica, se trabaja de una forma descriptiva, estableciendo relaciones entre entidades, indicando no cómo, sino qué hacer. La ecuación de Robert Kowalski (universidad de Edimburgo) establece la idea esencial de la programación lógica: algoritmos = lógica + control. Es decir, un algoritmo se construye especificando conocimiento en un lenguaje formal (lógica de primer orden), y el problema se resuelve mediante un mecanismo de inferencia (control) que actúa sobre aquél.

La programación lógica construye base de conocimientos mediante reglas y hechos. Un lenguaje de programación lógica por excelencia es el Prolog. (Rossel, 2007)

1.2.7 Programación orientada a objetos

La programación orientada a objetos (POO) permite organizar programas de forma parecida a como los objetos del mundo real se organizan, es una forma de concebir un programa de computadora. Modela una aplicación como una colección de objetos que se comunican entre sí para alcanzar una meta común. Como su mismo nombre indica, se basa en la idea de un objeto, que es una combinación de variables locales y procedimientos llamados métodos, que juntos, conforman una entidad de programación.

Tiene sus orígenes en la simulación; el primer lenguaje orientado a objetos fue Simula, esto se debe a que en simulación generalmente se modela una aplicación como un conjunto de entidades. (González, y otros, 2009)

La POO está basada en cuatro aspectos: definición de tipos de datos abstractos, herencia, encapsulamiento y polimorfismo.

El término encapsulamiento se usa para describir la combinación de estructuras de datos y de métodos que son manipulados por el objeto. La llamada a un objeto es lo que se denomina pasar un "aviso" a un objeto.

En la POO, encapsular significa, reunir y controlar el grupo resultante como un todo y no individualmente. La abstracción es un término externo al objeto, que controla la forma en que es visto por los demás. La estructura jerárquica, la cual consiste en la clasificación y organización de las abstracciones según su naturaleza es la herencia. La herencia se define como una jerarquía de extracciones, y la relación entre clases, donde se comparte la estructura y el comportamiento de una o más clases consideradas como clases superiores o una superclase, finalmente se puede resumir como una unidad independiente por sí misma heredada de una abstracción o superclase. (2009)

Un tipo de dato abstracto (TDA) es un modelo matemático compuesto por una colección de operaciones definidas sobre un conjunto de datos para el modelo. Los TDA se caracterizan por exportar un tipo, definir un conjunto de operaciones para la manipulación y utilizar la interfaz como único mecanismo de acceso a la estructura de datos.

Una clase es una representación real o concreta de un TDA, por lo tanto puede proporcionar los detalles de implementación para la estructura de datos y las operaciones que se definen. En una clase los atributos representan a la estructura de datos utilizada y los métodos a las operaciones. (Departamento de Técnicas de Programación, 2010)

El polimorfismo se refiere a la posibilidad de definir múltiples clases con funcionalidad diferente, pero con métodos o propiedades denominados de forma idéntica, que pueden utilizarse de manera intercambiable mediante código cliente en tiempo de ejecución. (Microsoft Corporation, 2010)

1.2.8 Programación orientada a aspectos

La programación orientada a aspectos (POA) es un paradigma de programación relativamente reciente que no anula a la programación orientada a objetos, sino que se integra a ésta.

La POA es una nueva metodología de programación que aspira a soportar la separación de competencias para los aspectos. Es decir, que intenta separar los componentes y los aspectos unos de otros, proporcionando mecanismos que hagan posible abstraerlos y componerlos para formar todo el sistema.

POA es un desarrollo que sigue al paradigma de la orientación a objetos, y como tal, soporta la descomposición orientada a objetos, además de la procedimental y la descomposición funcional. Pero, a pesar de esto, POA no se puede considerar como una extensión de la POO, ya que puede utilizarse con los diferentes estilos de programación ya mencionados.

¿Qué es un aspecto?

El nuevo paradigma de la POA es soportado por los llamados lenguajes de aspectos, que proporcionan constructores para capturar los elementos que se esparcen por todo el sistema. Estos elementos se llaman aspectos.

"Un aspecto es una unidad modular que se disemina por la estructura de otras unidades funcionales. Los aspectos existen tanto en la etapa de diseño como en la de implementación. Un aspecto de diseño es una unidad modular del diseño que se entremezcla en la estructura de otras partes del diseño. Un aspecto de programa o de código es una unidad modular del programa que aparece en otras unidades modulares del programa. "(Gregor Kiczales). (Quintero, 2000)

Los aspectos son la unidad básica de la POA, y pueden definirse como las partes de una aplicación que describen las cuestiones clave relacionadas con la semántica esencial o el rendimiento. También pueden verse como los elementos que se dispersan por todo el código y que son difíciles de describir localmente con respecto a otros componentes.

El soporte para este nuevo paradigma se logra a través de una nueva clase de lenguajes, llamados lenguajes orientados a aspectos (LOA), los cuales brindan mecanismos para capturar y declarar aquellos elementos que se desparraman por todo el sistema (aspectos).

Los LOA son aquellos lenguajes que permiten separar la definición de la funcionalidad "principal" de la definición de los diferentes aspectos. Algunos de estos LOA son: COOL (*COOrdination Language*), RIDL (*Remote Interaction and Data transfers aspect Language*), MALAJ (*Multi Aspect Language for Java*), AspectC, AspectC++ y AspectJ.

1.3 Tendencias y tecnologías actuales

El país ha identificado desde muy temprano la conveniencia y necesidad de dominar e introducir la práctica, en la sociedad, de las Tecnologías de la Informática y las Comunicaciones (TIC). En la UCI, su desarrollo y seguida modernización ha permitido el logro de un mejor funcionamiento de la misma, manteniéndose como la llamada "Ciudad Digital" sin estar ajena a los avanzados cambios en este ámbito.

1.3.1 Software libre

El *software* libre, es un movimiento tecnológico que ha revolucionado la sociedad. Presenta características especiales que han permitido la experimentación de nuevas formas de desarrollo y mantenimiento de programas, nuevos modelos económicos, y nuevas normas legales. Es un asunto de libertad, no de precio. Para entender el concepto, se debe pensar en "libre" como en libertad de

expresión. (Ferriol Ortiz, y otros, 2009) *Software* libre se refiere a la libertad de los usuarios para ejecutar, copiar, distribuir, estudiar, cambiar y mejorar el *software*. De modo más preciso, se refiere a cuatro libertades de los usuarios del *software* como usar el programa con cualquier propósito, estudiar cómo funciona, y adaptarlo a las necesidades del usuario, el poder distribuir copias con lo que se puede ayudar a otros, y mejorar el programa haciendo públicas las mejoras, de modo que toda la comunidad se beneficie.

Para que las libertades de hacer modificaciones y publicar versiones mejoradas tengan sentido, se debe tener acceso al código fuente del programa. Por lo tanto, la posibilidad de acceder al código fuente es una condición necesaria para el *software* libre.

Esto da la medida de la viabilidad económica de este sistema libre, que por ser libre no es necesariamente gratuito, sino que da la posibilidad de comercializarlo, regalarlo, prestarlo con total libertad y protegerlo legalmente. Evidentemente es la alternativa para los países subdesarrollados.

El costo al usar *software* libre es menor, pues el precio total de propiedad del sistema operativo libre Linux, es menos de la mitad que el de Windows. Gran parte del ahorro proviene de no tener que pagar licencia y de sus menores costos de administración. Da pie a la innovación tecnológica, donde el desarrollo en comunidad de este sistema y el conocimiento del código fuente propician que a cada instante, un desarrollador necesite nuevas actualizaciones y las realice él mismo, proponiendo nuevas funcionalidades al programa. Está expuesto al escrutinio público; el proceso de revisión pública al que está sometido el desarrollo del *software* libre imprime un gran dinamismo al proceso de corrección de errores. Cada mejora es socializada libremente, la comunidad puede cambiar la realidad de las innovaciones. Todo esto es visto como ventajas del uso de *software* libre.

Dentro de estas ventajas también se encuentra la independencia del proveedor gracias a la disponibilidad del código fuente. Al disponer del código fuente de la aplicación, es posible desarrollar internamente las mejoras o las modificaciones necesarias. De este modo, se contribuye a la formación de profesionales en nuevas tecnologías, al desarrollo local y de la industria nacional de *software*. Garantiza además la privacidad y la seguridad, pues el *software* libre, por su carácter abierto, dificulta la introducción de código malicioso, espía o de control remoto, debido a que el código lo revisan muchos usuarios y desarrolladores que pueden detectar posibles puertas traseras. En el mundo del *software* libre, cualquier programador puede realizar una auditoría para comprobar que no se ha introducido ningún código malicioso, y, a su vez, cualquier entidad puede añadir libremente encriptación adicional a la aplicación que utilice para proteger sus datos.

Como desventaja se puede destacar que la curva de aprendizaje es mayor, pues no es fácil aprender a trabajar con el *software* libre si ya se ha usado un *software* propietario. Dos personas que

nunca han tocado una computadora posiblemente tarden lo mismo en aprender a utilizar el *software* libre, que el *software* propietario, pero si ya ha usado un *software* propietario, generalmente se tarda más en aprender a usar el libre. También se necesita dedicar recursos a la reparación de errores, ya que el *software* libre se adquiere sin garantías explícitas, se vende tal cual, aunque pueden existir garantías específicas para determinadas situaciones, por lo que hay que dedicar recursos para reparar cualquier daño del *software*, aunque en el *software* propietario es imposible reparar errores, hay que esperar a obtener una nueva versión.

1.3.2 Aplicaciones *web*

En inglés se denomina "*browser-based aplication*", es decir, aplicación basada en navegadores. Son programas que se diseñan para funcionar a través de un navegador de Internet, es decir, son aplicaciones que se ejecutan de forma *online*. (de Pereda, 2007)

Una aplicación *web* es un conjunto de páginas *web* estáticas y dinámicas. Una página *web* estática es aquella que no cambia cuando un usuario la solicita: el servidor *web* envía la página al navegador solicitante sin modificarla. Por el contrario, el servidor modifica las páginas dinámicas antes de enviarlas al navegador solicitante. La naturaleza cambiante de este tipo de página es la que le da el nombre de dinámica.

Existen ventajas del uso de aplicaciones *web*, como la compatibilidad multiplataforma, las actualizaciones, acceso inmediato, vía cuenta *online*, facilidad de prueba, menores requerimientos de memoria local, menos errores, múltiples usuarios conectados de forma concurrente, mayor seguridad en los datos y que no se requieren complicadas combinaciones de *hardware*/software* para utilizar estas aplicaciones. Son fáciles de usar, no requieren conocimientos avanzados de computación.

1.4 Lenguaje de programación

Un lenguaje de programación es un conjunto de instrucciones, órdenes, comandos y reglas que permite la creación de programas. Los ordenadores, como todo equipo electrónico, funcionan por las señales eléctricas que reciben o dejan de recibir. Cuentan con pequeños interruptores por donde pasa o no la corriente, de ahí que cada interruptor pueda encontrarse sólo en dos estados: apagado o encendido, 0 ó 1. Esto es lo que da la posibilidad de que la máquina ejecute algún proceso según la combinación de estados de cada uno de los interruptores. Los lenguajes permiten al programador indicar lo que debe hacer un programa, sin tener que escribir largas cadenas de ceros y unos, sino

palabras (instrucciones) más comprensibles por las personas. Cada una de estas instrucciones está asociada a una combinación específica de ceros y unos que son entendidas por el ordenador.

Un lenguaje de programación es una herramienta que permite crear programas y *software*. Como ejemplo de ello, tenemos a Delphi, Visual Basic, Pascal, Java, entre otros.

Los lenguajes de programación facilitan la tarea de programación, ya que disponen de formas adecuadas que permiten ser leídas y escritas por personas, y ejecutadas por el ordenador. Representan en forma simbólica y en manera de un texto, los códigos que ejecuta la computadora. (HispaNetwork Publicidad y Servicios, 2007)

Para el desarrollo de sistemas *web*, se emplean diferentes y numerosos lenguajes, los cuales se diferencian entre sí en dependencia de las características específicas del producto final que se quiera obtener.

1.4.1 Lenguaje del lado del cliente

Un lenguaje del lado cliente es totalmente independiente del servidor, lo cual permite que la página *web* pueda ser albergada en cualquier sitio. Pero una página no se verá bien si el ordenador cliente no tiene instalados los *plug-in** adecuados. El código, tanto del hipertexto como de los *scripts*, es accesible por cualquier cliente y ello puede afectar a la seguridad.

1.4.1.1 HTML

Desde el surgimiento de Internet se han publicado sitios *web* gracias al lenguaje HTML. Es un lenguaje estático para el desarrollo de sitios *web* (acrónimo en inglés de *HyperText Markup Language*, en español Lenguaje de Marcas Hipertextuales). Desarrollado por el *World Wide Web Consortium* (W3C).

HTML es sencillo, permite describir hipertexto, el texto es presentado de forma estructurada y agradable, no necesita de grandes conocimientos cuando se cuenta con un editor de páginas *web*, sus archivos son pequeños, permite un despliegue rápido, es fácil aprendizaje y lo admiten todos los exploradores.

Es un lenguaje estático, que no permite la creación de páginas dinámicas. La interpretación de cada navegador puede ser diferente, guarda muchas etiquetas que pueden convertirse en “basura” y esto dificulta la corrección, y son además muy limitadas. (Pérez Valdés, 2007)

1.4.1.2 XML

XML sigla en inglés de *eXtensible Markup Language* (Lenguaje de Marcas eXtensible), es un metalenguaje extensible de etiquetas desarrollado por el *World Wide Web Consortium* (W3C). Permite definir la gramática de lenguajes específicos (de la misma manera que HTML). Por lo tanto, XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades.

XML no ha nacido sólo para su aplicación en Internet, sino que se propone como un estándar para el intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de texto, hojas de cálculo y casi cualquier cosa imaginable.

Es una tecnología sencilla que tiene a su alrededor otras que la complementan y la hacen mucho más grande y con unas posibilidades mucho mayores. Tiene un papel muy importante en la actualidad ya que permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil. Es extensible, lo que quiere decir que una vez diseñado un lenguaje y puesto en producción, igual es posible extenderlo con la adición de nuevas etiquetas de manera de que los antiguos consumidores de la vieja versión todavía puedan entender el nuevo formato. El analizador es un componente estándar, no es necesario crear uno específico para cada lenguaje, lo que posibilita el empleo de sólo uno de los tantos disponibles. De esta manera se evitan *bugs** y se acelera el desarrollo de la aplicación. Si un tercero decide usar un documento creado en XML, es sencillo entender su estructura y procesarlo. Mejora la compatibilidad entre aplicaciones. (Pérez Valdés, 2007)

1.4.1.3 JavaScript

Este es un lenguaje interpretado que no requiere compilación. Fue creado por Brendan Eich en la empresa *Netscape Communications*. Es utilizado principalmente en páginas *web*, similar a Java, aunque no es un lenguaje orientado a objetos, pues no dispone de herencia. La mayoría de los navegadores en sus últimas versiones interpretan código JavaScript.

El código JavaScript puede ser integrado dentro de las páginas *web*. Para evitar incompatibilidades el *World Wide Web Consortium* (W3C) diseñó un estándar denominado DOM* (en inglés *Document Object Model*, en su traducción al español Modelo de Objetos del Documento).

Este código JavaScript es ejecutado en el cliente, y es visible por cualquier usuario. Para poderse ejecutar, debe ser descargado completamente. (Pérez Valdés, 2007)

* Ver Glosario de términos

1.4.2 Lenguaje del lado del servidor

Se les clasifica así a los lenguajes de programación que usan la tecnología cliente/servidor (un servidor es un ordenador remoto, en algún lugar de una red, que proporciona información según se le solicite, mientras que un cliente funciona en su computadora local, se comunica con el servidor remoto y pide a éste información) que se ejecutan del lado del servidor y de los cuales los usuarios sólo obtienen el beneficio del procesamiento de la información.

1.4.2.1 ASP

Es una tecnología desarrollada por Microsoft para el desarrollo de sitios *web* dinámicos. ASP en inglés es *Active Server Pages*, liberado por Microsoft en 1996. Las páginas *web* desarrolladas bajo este lenguaje necesitan tener instalado *Internet Information Server* (IIS*).

ASP no necesita ser compilado para ejecutarse. Es una tecnología dinámica que cuando el usuario solicita un documento de este tipo, las instrucciones de programación dentro del *script* son ejecutadas para enviar al navegador únicamente el código HTML resultante. La ventaja principal radica en la seguridad que tiene el programador sobre su código, ya que éste se encuentra únicamente en los archivos del servidor que al ser solicitado a través de la *web*, es ejecutado, por lo que los usuarios no tienen acceso más que a la página resultante en su navegador.

El código es desorganizado y se necesita escribir mucho código para realizar funciones sencillas, además la tecnología es propietaria. (Pérez Valdés, 2007)

1.4.2.2 JSP

Es un lenguaje para la creación de sitios *web* dinámicos, acrónimo de *Java Server Pages*. Está orientado a desarrollar páginas *web* en Java. JSP es un lenguaje multiplataforma, creado para ejecutarse del lado del servidor.

Fue desarrollado por Sun Microsystems, para la creación de aplicaciones *web* potentes. Posee un motor de páginas basado en los *servlets** de Java. Para su funcionamiento se necesita tener instalado un servidor Tomcat.

* Ver Glosario de términos

Se caracteriza por tener el código separado de la lógica del programa y puede ser incrustado en código HTML, las páginas son compiladas en la primera petición, además permite separar la parte dinámica de la estática en las páginas *web*. (Pérez Valdés, 2007)

1.4.2.3 Python

Es un lenguaje de programación creado en el año 1990 por Guido van Rossum, es el sucesor del lenguaje de programación ABC. Python es comparado habitualmente con Perl. Los usuarios lo consideran como un lenguaje más limpio para programar. Permite la creación de todo tipo de programas incluyendo los sitios *web*.

Su código no necesita ser compilado, es interpretado. Es un lenguaje de programación multiparadigma, lo cual fuerza a que los programadores adopten por un estilo de programación particular.

Es libre y de fuente abierta, propósito general, con gran cantidad de funciones y librerías, sencillo y rápido de programar, multiplataforma, tiene licencia de código abierto (O*penSource*), orientado a objetos y portable. (Pérez Valdés, 2007)

1.4.2.4 PHP

PHP, acrónimo recursivo de "PHP: *Hypertext Preprocessor*". Es un lenguaje de programación, diseñado originalmente para el diseño de páginas *web* dinámicas, interpretado, de alto nivel, embebido en páginas HTML y ejecutado en el servidor. (WHMCompleteSolution, 2010) Este lenguaje de programación es gratuito e independiente de la plataforma, rápido, con una gran librería de funciones y mucha documentación.

Todas las funciones del sistema están explicadas y ejemplificadas detalladamente en el archivo de ayuda. Es libre, por lo que se presenta como una alternativa de fácil acceso para todos. Permite las técnicas de POO, tiene una biblioteca nativa de funciones sumamente amplia e incluida. No requiere definición de tipos de variables y cuenta con manejo de excepciones (desde PHP5).

Puede ser desplegado en la mayoría de los servidores *web* y en casi todos los sistemas operativos y plataformas sin costo alguno.

Si bien PHP no obliga a quien lo usa a seguir una determinada metodología a la hora de programar (muchos otros lenguajes tampoco lo hacen), aún estando dirigido a alguna en particular, el programador puede aplicar en su trabajo cualquier técnica de programación y/o desarrollo que le permita escribir código ordenado, estructurado y manejable. Un ejemplo de esto son los desarrollos

que en PHP se han hecho del patrón Modelo Vista Controlador (o MVC), éste permite separar el tratamiento y acceso a los datos, la lógica de control y la interfaz de usuario en tres componentes independientes.

Permite la conexión a diferentes tipos de servidores de bases de datos tales como MySQL, Postgres, Oracle, ODBC, DB2, Microsoft SQL Server, Firebird y SQLite. El código fuente escrito en PHP es invisible al navegador y al cliente ya que es el servidor el que se encarga de ejecutar el código y enviar su resultado HTML al navegador. Esto hace que la programación en PHP sea segura y confiable. (Martínez, 2001)

1.5 Sistema de gestión de base de datos

Un sistema de gestión de base de datos (SGBD), es un tipo de *software* muy específico dedicado a servir de interfaz entre la base de datos, el usuario y las aplicaciones que la utilizan. Se compone de un lenguaje de definición de datos, de un lenguaje de manipulación de datos y de un lenguaje de consulta. Permite definir los datos a distintos niveles de abstracción y manipularlos, garantizando la seguridad e integridad de los mismos.

Permite definir una base de datos, especificar tipos, estructuras y restricciones de la información que ésta domina; construirla, en lo que se refiere a guardarlos en algún medio controlado por el mismo SGBD, y poder manipularla, como realizar consultas, actualizarla y generar informes.

1.5.1 MySQL Server

MySQL Server es la base de datos de código abierto más usada del mundo desarrollado y proporcionado por MySQL AB, empresa cuyo negocio consiste en proporcionar servicios en torno al servidor de bases de datos MySQL.

Los fundadores buscaban la forma de crear un manejador de bases de datos que fuera “rápido”, todavía más rápido que el sistema final. Así surgió MySQL, primero como un producto de la empresa y después como *software* de dominio público.

El servidor MySQL fue desarrollado originalmente para manejar grandes bases de datos mucho más rápido que las soluciones existentes y ha estado siendo usado exitosamente en ambientes de producción sumamente exigentes por varios años. Aunque se encuentra en desarrollo constante, éste ofrece hoy un conjunto rico y útil de funciones. Su conectividad, velocidad, y seguridad hacen que sea un servidor bastante apropiado para acceder a bases de datos en Internet.

MySQL tiene como unas de las más importantes características que está escrito en C y C++; trabaja bajo diferentes plataformas; cuenta con procesos multihilo y una capacidad de trabajar servidores con varios procesadores; provee sistema transaccional con la tabla Innodb*; soporta muchos tipos de columnas para las tablas: *FLOAT, DOUBLE, CHAR, VARCHAR, TEXT, BLOB, DATE, TIME, DATETIME, TIMESTAMP, YEAR, SET, ENUM* y *OpenGIS* (Modelo Geométrico) y maneja la memoria a través de manejo del *buffer** y caché*. (González Castellanos, y otros, 2005)

1.5.2 Oracle

Está concebido con el fin de manejar grandes cantidades de información, además de admitir conexiones concurrentes de multitud de usuarios hacia los mismos datos.

Aporta funcionalidades como soporte y tratamiento de una gran cantidad de información, sustenta el acceso de una gran suma de usuarios concurrentemente a los datos, seguridad de acceso a la información, restringiéndolo según las necesidades de cada usuario, integridad referencial en la estructura de la base de datos, conectividad entre las aplicaciones de los clientes (estructura cliente/servidor) y entre bases de datos remotas, portabilidad y compatibilidad.

Para establecer un entorno de trabajo con este SGBD se necesita el *software* Oracle para el servidor en la versión diseñada específicamente para el sistema operativo concreto (UNIX, Windows NT, entre otros), además del *software* Oracle para el cliente, que se ubicará en los puestos de trabajo de los usuarios (también en dependencia del sistema operativo) el cual estará formado por un grupo de herramientas específicas desarrollada para la relación cliente/servidor. (González Castellanos, y otros, 2005)

1.5.3 SQL Server 2000

SQL Server es el sistema de gestión de base de datos representativo de la firma mundialmente conocida: Microsoft. SQL Server 2000. Proporciona agilidad a sus operaciones de análisis y administración de datos.

Ha obtenido importantes galardones en pruebas de referencia por su escalabilidad y velocidad. Es un producto de base de datos totalmente habilitado para *web* que proporciona una compatibilidad fundamental con el lenguaje de marcado extensible (XML, *Extensible Markup Language*) y la capacidad para realizar consultas en Internet aún por encima del servidor de seguridad. (González Castellanos, y otros, 2005)

* Ver Glosario de términos

1.5.4 PostgreSQL

Como sistema de gestión de bases de datos, se escogió PostgreSQL, el cual es un sistema de gestión de bases de datos objeto-relacional basado en el proyecto Ingres, de la universidad de Berkeley, California. Pionero en muchos de los conceptos existentes en el sistema objeto-relacional actual, PostgreSQL incluye características de la orientación a objetos como pueden ser: la herencia, tipos de datos, funciones, restricciones, disparadores, reglas e integridad transaccional, además de otras específicas del gestor, como lo son: un mejor soporte para *sub-selects*, *triggers* [*], vistas y procedimientos almacenados.

PostgreSQL permite la implementación del SQL estándar (92, 99 y 2003), soporte para distintos tipos de datos, como datos sobre redes (MAC, IP), cadenas de *bits**, numéricos, secuencias, cadenas, fechas y horas según la localidad, binarios, BLOBs, así como la creación de tipos de datos propios, la declaración de funciones propias, la definición de dispensadores, y también soporte en la mayoría de los sistemas operativos más utilizados incluyendo, Linux, varias versiones de UNIX, BeOS y Windows. (González Castellanos, y otros, 2005)

1.6 *Framework*

Esto no es más que una estructura de soporte definida mediante la cual otro proyecto de *software* puede ser desarrollado y organizado. Puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros programas para ayudar a desarrollar y unir los diferentes componentes de un proyecto. Representa una arquitectura que modela las relaciones generales de las entidades del dominio. Provee una estructura y una metodología de trabajo la cual extiende o utiliza las aplicaciones del dominio.

Son diseñados con el intento de facilitar el desarrollo de aplicaciones, permitiendo a los diseñadores y programadores pasar más tiempo identificando requerimientos de *software* que tratando con los tediosos detalles de bajo nivel de proveer un sistema funcional. Es el esqueleto sobre el cual varios objetos son integrados para una solución dada. No es más que una base de programación que atiende a sus descendientes (manejado de una forma estructural y/o en cascada) posibilitando cualquier respuesta ante las necesidades de sus miembros, o secciones de una aplicación *web*.

[*] Ver Glosario de términos

1.6.1 jQuery

jQuery es una biblioteca o *framework* de Javascript que permite simplificar la manera de interactuar con los documentos HTML, manipular el árbol DOM, manejar eventos, desarrollar animaciones y agregar interacción con la tecnología AJAX* a páginas *web*.

jQuery es *software* libre y de código abierto, posee un doble licenciamiento bajo la licencia MIT y de la GNU *General Public License*. Al igual que otras bibliotecas, ofrece una serie de funcionalidades basadas en Javascript que de otra manera requerirían de mucho más código. Es decir, con las funciones propias de esta biblioteca se logran grandes resultados en menos tiempo y espacio.

Consiste en un único fichero JavaScript que contiene las funcionalidades comunes de DOM, eventos, efectos y AJAX.

La característica principal de esta biblioteca es que permite cambiar el contenido de una página *web* sin necesidad de recargarla, mediante la manipulación del árbol DOM y peticiones AJAX.

Es un producto con una buena aceptación por parte de los programadores y muy difundido en el mercado del software, por lo que se supone que es una de las mejores opciones. Además, es un producto estable, bien documentado y con un gran equipo de desarrolladores a cargo de la mejora y actualización del *framework*. Cuenta con una dilatada comunidad de creadores de *plugins* o componentes, lo que hace fácil encontrar soluciones ya creadas en jQuery para implementar asuntos como interfaces de usuario, galerías, votaciones y efectos diversos. (Álvarez, 2009)

1.6.2 Symfony

Symfony es un *framework* que tiene como objetivo fundamental automatizar los patrones más utilizados en la elaboración de sistemas *web*, además de obligar a clarificar a los programadores el código. Establece un estándar de código legible, encapsula operaciones complejas en simples líneas de código, ahorrando mucho más tiempo a la hora de mostrar datos directamente de la base de datos.

Este *framework* está implementado bajo el lenguaje PHP 5, ha sido utilizado en varias aplicaciones *web* obteniéndose resultados satisfactorios. Es compatible con la mayoría de gestores de base datos existentes, se puede comentar sobre MySQL, PostgreSQL, Oracle y SQL Server de Microsoft. También otorga facilidades en diferentes plataformas. Symfony provee una abstracción de la base de datos, que permite una mayor facilidad de obtención de los datos haciendo a la vista y a las acciones independientes del gestor de base de datos. En la vista provee de *helpers** que facilitan el

* Ver Glosario de términos

trabajo para los diseñadores en el código HTML de la aplicación, aunque es necesario crear una nueva vista, mantiene el modelo y el controlador original. (Rodríguez Moreno, y otros, 2009)

Es fácil de instalar y configurar en la mayoría de plataformas, independiente del sistema gestor de bases de datos, suficientemente flexible como para adaptarse a los casos más complejos, basado en la premisa de "convenir en vez de configurar", en la que el desarrollador sólo debe configurar aquello que no es convencional y sigue la mayoría de mejores prácticas y patrones de diseño para la *web*. (Zaninotto, y otros, 2009)

1.6.3 CodeIgniter

CodeIgniter es un conjunto de herramientas para personas que construyen aplicaciones *web* usando PHP. Su objetivo es permitirle desarrollar proyectos mucho más rápido que si lo escribiese desde cero, proporcionándole un rico juego de librerías para tareas comúnmente necesarias, así como una interfaz simple y una estructura lógica para acceder a esas librerías.

CodeIgniter permite creativamente enfocarse en el proyecto minimizando la cantidad de código necesario para una tarea dada.

CodeIgniter se encuentra bajo una licencia *OpenSource* Apache/BSD-*style*, lo cual facilita éxito de las tendencias de migración a *software* libre del país y de la UCI, pudiéndose utilizar en cualquier proyecto que se desee.

CodeIgniter está escrito para ser compatible con PHP 4. Aunque corre en PHP5, simplemente no toma ventaja de cualquiera de las características nativas que sólo están disponibles en esa versión.

Es verdaderamente liviano, ya que el núcleo del sistema sólo requiere unas pocas librerías que permiten realizar las tareas de desarrollo *web* más comunes como acceder a una base de datos, mandar un correo electrónico, validar datos de un formulario, mantener sesiones, manipular imágenes, trabajar con datos XML-RPC, entre otras, aunque se le pueden agregar librerías cargándolas dinámicamente a pedido, basado en las necesidades de un proceso dado, así que el sistema base es muy delgado y bastante rápido.

Usa la arquitectura Modelo Vista Controlador (MVC), que permite una buena separación entre lógica y presentación. Esto es particularmente bueno para proyectos en los cuales los diseñadores están trabajando con sus archivos de plantilla, ya que el código en esos archivos será mínimo.

CodeIgniter cuenta con clases de base de datos llenas de características con soporte para varias plataformas, una clase de FTP* (*File Transfer Protocol* o Protocolo de Transferencia de

* Ver Glosario de términos

Archivos), localización, paginación, encriptación de datos, puntos de referencia, cacheo de páginas completas, historial de errores, clase de calendario, clase de codificación Zip*, clase de prueba de unidad, entre otras características. (Ellis, y otros, 2009)

1.6.2.1 Modelo Vista Controlador

CodeIgniter está basado en el patrón de desarrollo Modelo-Vista-Controlador (MVC). Es una aproximación al *software* que separa la lógica de la aplicación de la presentación.

En la práctica, permite que sus páginas *web* contengan mínima codificación ya que la presentación es separada del código PHP.

El Modelo representa la estructura de datos. Típicamente sus clases de modelo contendrán funciones que lo ayudarán a recuperar, insertar y actualizar información en su base de datos.

La Vista es la información que es presentada al usuario. Ésta normalmente será una página *web*, pero en CodeIgniter, la podemos ver como un fragmento de una página, como un encabezado o un pie de página. También puede ser una página RSS (*Really Simple Syndication,* formato que permite emitir contenidos desde un sitio para que sean agregados fácilmente en aplicaciones o sitios *web*), o cualquier otro tipo de "página".

El Controlador sirve como un intermediario entre el Modelo, la Vista y cualquier otro recurso necesario para procesar la petición HTTP* (*HyperText Transfer Protocol* o Protocolo de Transferencia de Hipertexto) y generar una página *web*.

CodeIgniter tiene un enfoque bastante flexible del MVC, ya que los Modelos no son requeridos. Si no se necesita agregar separación, o se descubre que mantener los modelos hace que se requiera más complejidad que la deseada, se ignora y se construye la aplicación mínimamente usando Controladores y Vista. CodeIgniter también permite incorporar sus códigos existentes, o incluso desarrollar librerías de núcleo para el sistema, habilitándolo a trabajar de forma que tenga más sentido para el desarrollador. (Ellis, y otros, 2009)

1.7 Entorno de desarrollo

Un entorno de desarrollo integrado o IDE (*Integrated Development Environment*), es un programa informático compuesto por un conjunto de herramientas de programación. Puede dedicarse

* Ver Glosario de términos

en exclusiva a un solo lenguaje de programación o bien, poder utilizarse para varios. (Babylon Programa de Traducción, 2009)

Un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (GUI). Los IDE pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes.

Los IDE proveen un marco de trabajo amigable para la mayoría de los lenguajes de programación tales como C++, Python, Java, C#, Delphi, Visual Basic, entre otros. En algunos lenguajes, un IDE puede funcionar como un sistema en tiempo de ejecución, en donde se permite utilizar el lenguaje de programación en forma interactiva, sin necesidad de trabajo orientado a archivos de texto.

1.7.1 Eclipse

Eclipse es un entorno de desarrollo integrado de código abierto, multiplataforma. Típicamente ha sido usado para desarrollar entornos integrados de desarrollo (IDE), como el IDE de Java llamado Java Development Toolkit (JDT) y el compilador (ECJ) que se entrega como parte de Eclipse (y que son usados también para desarrollar el mismo Eclipse). Sin embargo, también se puede usar para otros tipos de aplicaciones cliente.

Eclipse emplea módulos (*plug-in*) para proporcionar toda su funcionalidad al frente de la plataforma de cliente rico, a diferencia de otros entornos monolíticos donde las funcionalidades están todas incluidas, las necesite el usuario o no. Este mecanismo de módulos es una plataforma ligera para componentes de *software*. De esta forma, Eclipse puede extenderse usando otros lenguajes de programación como C/C++, Python y PHP, puede además trabajar con sistemas de gestión de base de datos.

La arquitectura *plug-in* permite escribir cualquier extensión deseada en el ambiente. En cuanto a las aplicaciones clientes, Eclipse provee al programador *frameworks* muy ricos para el desarrollo de aplicaciones gráficas, definición y manipulación de modelos de *software* y aplicaciones *web*. (Makarov, 2009)

1.7.2 Zend Studio

Zend Studio es una herramienta profesional para la construcción de *software*, excelente para trabajar proyectos usando PHP. Es concebido con el fin de crear aplicaciones altamente fiables, proporciona una facilidad de uso inigualable, escalabilidad, fiabilidad, y la extensión que los

programadores profesionales y de empresas requieren para desarrollar, distribuir, depurar y administrar aplicaciones PHP críticas de negocios. Permite conectarse directamente a bases de datos diseñadas en algunos de los gestores más utilizados tales como MySQL, Oracle, Microsoft SQL Server, PostgreSQL y SQlite.

Tiene características de depuración avanzadas, incluyendo condiciones límite, visualización de errores, vistas avanzadas, variables y *buffer* de salida. Asegura la protección máxima de ubicaciones de proyectos o en Internet con depuradores remotos. Facilita el desarrollo y colaboración en equipo mediante la administración efectiva de su código fuente utilizando CVS* (*Concurrent Versions System* o Sistema de Control de Versiones) o *Subversion* directamente desde Zend Studio. Tiene soporte para PHP5 completo, analizador de código, carpeta de código, completado de código, coloreado de sintaxis, administrador de proyecto, editor de código, depurador de gráficos y asistentes. (Makarov, 2009)

1.7.3 NetBeans

NetBeans es un IDE escrito en Java de código abierto (*OpenSource*) gratuito para desarrolladores de *software*. Ofrece todas las herramientas necesarias para crear aplicaciones profesionales, empresariales, *web* y móviles con el lenguaje Java, JavaFX, C/C ++ y lenguajes dinámicos como PHP, JavaScript, Groovy y Ruby. NetBeans es fácil de instalar y listo para usar y se puede ejecutar tanto en Windows, Linux, Mac OS X y Solaris.

La plataforma NetBeans es una base modular y extensible usada como una estructura de integración para crear aplicaciones de escritorio de gran tamaño. Ofrece servicios comunes permitiéndole al desarrollador enfocarse en la lógica específica de su aplicación.

Es el primer IDE que soporta totalmente Java EE6 y Sun GlassFish Enterprise Server v3. Mejora el soporte para PHP, JavaFX y C/C++. (Sun, 2009)

1.8 Plataforma

Una plataforma es la tecnología básica del *software* y *hardware* de un ordenador que define cómo funciona y qué otro tipo de *software* se puede emplear con él. En ocasiones se puede considerar sinónimo de sistema operativo como Windows y Linux mientras que en otras no, como en el caso de Java y .NET.

* Ver Glosario de términos

1.8.1 .NET

La plataforma .NET es una capa de *software* que se coloca entre el sistema operativo (SO) y el programador que abstrae los detalles internos del SO. Es portable debido a la abstracción del programador respecto al SO, una aplicación .NET puede ser ejecutada en cualquier SO de cualquier máquina que disponga de una versión de la plataforma. Sólo está disponible para la familia Windows aunque se está desarrollando una versión para Linux de Corel.

Es multilenguaje puesto que cualquier lenguaje de programación puede adaptarse a la plataforma .NET y ejecutarse en ella.

Microsoft define .NET como un entorno para la construcción, desarrollo y ejecución de servicios *web* y otras aplicaciones que consiste en tres partes fundamentales: el *Common Language Runtime* (entorno de ejecución), las *Framework Classes* (clases de la plataforma) y ASP.NET. (Sotomayor Basilio, 2010)

1.8.2 Java

Java es una plataforma virtual que provee el lenguaje de programación Java, la *Java Virtual Machine* (JVM), máquina virtual con su propio *set* de instrucciones y la *Java Platform API*, una interfaz para la programación de aplicaciones.

Está basada únicamente en *software* que corre por encima de las basadas en *hardware*. (Universidad de Chile, Departamento de Ciencias de la Computación, 2009)

1.8.3 GNU/Linux

Debido a un particular giro de acontecimientos, la versión de GNU* más usada actualmente es «Linux», y muchos usuarios no son conscientes del alcance de su conexión con el Proyecto GNU. Efectivamente hay un Linux, y las personas lo usan, pero no es el sistema operativo. Linux es el núcleo: el programa del sistema que asigna los recursos de la máquina a los otros programas que se ejecuten. El núcleo es una parte esencial de todo SO, pero inútil por sí solo; sólo puede funcionar en el contexto de un SO completo. Linux se usa normalmente en combinación con el sistema operativo GNU: el sistema completo es básicamente GNU, con Linux actuando de núcleo. El Proyecto GNU apoya tanto a los sistemas GNU/Linux como al sistema GNU. Financia la reescritura de las extensiones relacionadas con Linux de la biblioteca de C de GNU, de modo que ahora se integran bien, y los sistemas GNU/Linux modernos usan la versión actual de la biblioteca sin necesidad de hacerle modificaciones. También financió las primeras etapas del desarrollo de Debian GNU/Linux.

GNU/Linux es un sistema operativo de *software* libre que cumple las normas POSIX, su base es un núcleo o Kernel monolítico llamado Linux combinado con un grupo de librerías y herramientas. Su estructura general es la típica de cualquier sistema UNIX (núcleo, "intérprete de comandos", aplicaciones). GNU/Linux tiene todas las características que se pueden esperar de un moderno y flexible sistema operativo. Incluye multitarea real, memoria virtual, librerías compartidas, dirección y manejo propio de memoria. Es sin lugar a dudas uno de los ejemplos más prominentes del *software* libre y del desarrollo del código abierto.

Varias son las distribuciones de GNU/Linux que se han creado a nivel mundial gracias al trabajo constante de los desarrolladores y promotores del *software* libre en el mundo entero, algunas de ellas son: Debian, Mandriva, Ubuntu, Novel/Suse, Red Hat y Gentoo. (Zaballa Coca, 2009)

1.9 Metodología de desarrollo de *software*

Las metodologías imponen un proceso disciplinado sobre el desarrollo de *software* con el fin de hacerlo más predecible y eficiente. Lo hacen desarrollando un proceso detallado con un fuerte énfasis en planificar inspirado por otras disciplinas de la ingeniería.

Las metodologías ingenieriles han estado presentes durante mucho tiempo. No se han distinguido precisamente por ser muy exitosas. Aún menos por su popularidad. La crítica más frecuente a estas metodologías es que son burocráticas. Hay tanto que hacer para seguir la metodología que el ritmo entero del desarrollo se retarda.

Hoy en día existen numerosas propuestas metodológicas que inciden en distintas dimensiones del proceso de desarrollo.

El uso en la actualidad de las metodologías ágiles para el desarrollo de *software* es muy común debido a las ventajas que las mismas proporcionan con respecto a las metodologías de desarrollo tradicionales. En los años 90 muchos desarrolladores de *software* se dieron cuenta de la necesidad de crear metodologías de desarrollo livianas y maniobrables, por el ambiente cambiante y turbulento dentro de los procesos de desarrollo. Las metodologías ágiles comparten algunos aspectos comunes, a pesar de que los detalles de desarrollo varían en dependencia de la metodología que se emplea, además forman parte del movimiento de desarrollo ágil de *software*, que se basan en la adaptabilidad de cualquier cambio como medio para aumentar las posibilidades de éxito de un proyecto. Se le denomina ágil como la habilidad de responder de forma versátil al cambio para maximizar los beneficios.

Las metodologías ágiles varían en su forma de responder al cambio, pero en general comparten características como el uso de procesos de construcción iterativos, la entrega de *software*

funcional lo más pronto posible y el privilegio del valor del equipo de trabajo sobre el valor del proceso. Se puede ver como ventaja ya que de esta manera se fortalece la comunicación y la colaboración.

De manera general este tipo de metodología favorece el desarrollo de *software* de forma colaborativa, debido a que se le presta más atención a los recursos humanos que a las herramientas de desarrollo, además de que son adaptables al cambio sin demasiada burocracia.

El principal objetivo de estas metodologías es permitir a los equipos desarrollar *software* rápidamente y respondiendo a los cambios que puedan surgir a lo largo del proyecto.

Las propuestas de metodologías más tradicionales que 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 pueden incluso limitar la capacidad de los desarrolladores para llevar a cabo el proyecto.

1.9.1 eXtreme Programming (XP)

Un proceso ligero, de bajo riesgo, flexible, predecible, científico y divertido de desarrollar *software*. Pura lógica es la palabra con que describen la Programación Extrema algunos conocedores y desarrolladores, las prácticas de este método, están basadas en la simplicidad, la comunicación y el reciclado continuo de código, XP es la integración de las prácticas de métodos tradicionales resumiendo o utilizando lo más práctico y eficaz. Es un cambio revolucionario llevar las prácticas al extremo, teniendo como fin la satisfacción del cliente, *software* de calidad, integración del cliente en el equipo de trabajo y adaptación a los cambios, ya sean de tecnología o de metáforas dentro del negocio.

Esta metodología involucra los siguientes roles:

- Jefe del proyecto: organiza y guía las reuniones, asegura las condiciones adecuadas para el proyecto.
- Programador: es el responsable de las decisiones técnicas y de construir el sistema sin distinción entre analistas, diseñadores o codificadores. En XP, los programadores diseñan, programan y realizan las pruebas.
- Cliente: es parte del equipo, determina qué construir y cuándo, establece las pruebas funcionales.
- Encargado de pruebas (*Tester*): ayuda al cliente con las pruebas funcionales y se asegura de que las pruebas funcionales se superan.
- Encargado de seguimiento (*Tracker*): *Metric Man*, observa sin molestar y conserva datos históricos.

- Entrenador (*Coach*): el líder del equipo, toma las decisiones importantes, es el principal responsable del proceso y tiende a estar en un segundo plano a medida que el equipo madura.
- Consultor: un miembro externo del equipo con un conocimiento específico en algún tema necesario para el proyecto. Guía al equipo para resolver un problema específico.
- Gestor (*Big boss*): el vínculo entre clientes y programadores, ayuda a que el equipo trabaje efectivamente creando las condiciones adecuadas. Su labor esencial es de coordinación. (Letelier, y otros, 2010)

Los artefactos que se generan en esta metodología son:

- Historias de Usuario: representan una breve descripción del comportamiento del sistema, emplean terminología del cliente sin lenguaje técnico, se realiza una por cada característica principal del sistema, se emplean para hacer estimaciones de tiempo y para el plan de lanzamientos, reemplazan un gran documento de requisitos y presiden la creación de las pruebas de aceptación.
- Tareas de Ingeniería: son actividades que los programadores conocen que el sistema debe hacer.
- Tarjetas CRC (Clase Responsabilidad Colaborador): estas tarjetas se dividen en tres secciones que contienen la información del nombre de la clase, sus responsabilidades y sus colaboradores. (Moraga, 2005)

El ciclo de vida de XP consta de seis fases:

- Exploración.
- Planificación de la entrega.
- Iteraciones.
- Producción.
- Mantenimiento.
- Muerte del proyecto.

1.9.2 Scrum

Scrum es una metodología para el desarrollo ágil de productos, expuesta por Hirotaka Takeuchi e Ikujiro Nonaka, los cuales exponen según sus ideas que en el mercado competitivo de los productos tecnológicos, además de los conceptos básicos de calidad, coste y diferenciación, se exige también rapidez y flexibilidad. Los nuevos productos representan cada vez un porcentaje más importante en el volumen de negocio de las empresas. El mercado exige ciclos de desarrollo más cortos. Esta

metodología debe su nombre a un estudio realizado por sus creadores sobre nuevas prácticas de producción de *software*, comparando las mismas con un juego de *rugby*.

Aunque surgió como modelo para el desarrollo de productos tecnológicos, también se emplea en entornos que trabajan con requisitos inestables y que requieren rapidez y flexibilidad; situaciones frecuentes en el desarrollo de determinados sistemas de *software*.

Scrum es una metodología de desarrollo muy simple, que requiere trabajo duro, porque la gestión no se basa en el seguimiento de un plan, sino en la adaptación continua a las circunstancias de la evolución del proyecto. (Peña Cruz, y otros, 2008)

Scrum posee tres fases fundamentales: una breve fase de planificación, en la cual se realizan las labores básicas de una planificación donde se puede encontrar la visión general del proyecto (estimación muy general, viabilidad del sistema) y construcción del *backlog* por un lado y por otro el desarrollo de la arquitectura al detalle; otra de desarrollo, en la cual tienen lugar los famosos *sprints*; y otra final de entrega y balance de los éxitos y fracasos logrados. (Luix, 2009)

Los roles comprometidos dentro de esta metodología se encuentran el *product owner* o dueño del producto, que representa la voz del cliente y aporta la visión de negocio. Él se encarga de escribir las Historias de Usuario, les da prioridad y las ubica en la lista de requisitos del producto; el scrum *master* o facilitador, que tiene como principal papel el de dejar el camino libre de obstáculos e impedimentos para que el resto consiga el objetivo del *sprint* y el equipo, que tiene la responsabilidad de entregar el producto. Además del papel de los siguientes, que no son actores esenciales, pero sí están implicados y deben ser tenidos en cuenta, juegan: los usuarios del producto o aplicación, los clientes y vendedores, los gestores y directivos. (PymeCrunch, 2008)

Los artefactos que se generan en esta metodología son:

- *Sprint:* es la base del desarrollo de Scrum.
- *Product backlog:* crea un listado con los requisitos de los usuarios o propietarios del sistema para planificar el proyecto.
- *Sprint backlog*: especifica la serie de tareas que se van a desarrollar según los requisitos señalados. (Moraga, 2005)

1.9.3 Scrum-XP

Scrum-XP surgió en Cuba, en la Universidad de las Ciencias informáticas (UCI) y no es más que la unión de XP y Scrum, para el logro de un buen desarrollo de *software*, propuesta en el 2007 por la ingeniera Malay Rodríguez Villar, y probada en los proyectos que trabajan con *software* libre, obteniendo buenos resultados. Esta metodología surgió con el nombre metodología ágil Gladys Marsi

Peñalver Romero UNICORNIOS revisión 2 (MA-GMPR-UR2) la cual fue renombrada poco tiempo después.

La metodología está dividida en cuatro fases, que son precisamente la base de la estructura del nuevo expediente de proyecto, estas son:

- Planificación-Definición: en esta fase se generan todos los documentos que se encuentran relacionados con la concepción inicial del sistema, así como la definición del mismo. También se incluyen algunos que están vinculados a la primera parte de los procesos de Ingeniería de *Software* tales como los relacionados con el negocio, los requisitos y el diseño.
- Desarrollo: en la primera parte de esta fase se generan todos los documentos relacionados con la planificación de las iteraciones, y además se recogen las principales definiciones que se manejan en la metodología y otros términos de difícil entendimiento para los clientes, así como de las tareas a realizar durante la implementación. Además, se genera el código fuente en la etapa de implementación y, como última parte de esta etapa, los documentos relacionados con las pruebas.
- Entrega: en esta fase se realiza la entrega del *software* y su documentación, generándose aquellos documentos que son imprescindibles para el entrenamiento y entendimiento del producto.
- Mantenimiento: se realizan las actividades relacionadas con el soporte del *software* y se generan los documentos relacionados con los cambios que puedan ocurrir en el mismo.

Cada una de estas fases está compuesta por una serie de actividades tales como escribir la visión y escribir la reserva del producto que son las que generan los artefactos como la plantilla de concepción del sistema y la lista de reserva del producto que quedan incluidos en el nuevo expediente de proyecto. Estas actividades están recogidas en el guión de la metodología. Para la definición de los artefactos que se generan en cada una de las fases se tiene en cuenta como elemento fundamental, las características de las metodologías ágiles, las cuales tienen como premisa la no duplicación de esfuerzos, así como la integración del cliente en el equipo de desarrollo, esto garantiza que no haya necesidad de documentaciones extensas, sólo se documenta lo necesario para una futura reutilización. (Bello del Pino, y otros, 2009)

1.9.4 Proceso Racional Unificado

RUP, en inglés *Rational Unified Process*, es un proceso para el desarrollo de un proyecto de *software* que define claramente quién, cómo, cuándo y qué debe hacerse en el proyecto. Es una

metodología tradicional. Tiene tres características esenciales: está dirigido por los casos de uso, los cuales orientan el proyecto según lo que el usuario necesita, está centrado en la arquitectura, lo que relaciona, la toma de decisiones que indican cómo tiene que ser construido el sistema y en qué orden, y por último, es iterativo e incremental, lo que divide el proyecto en mini-proyectos donde los casos de uso y la arquitectura cumplen sus objetivos de manera más depurada.

1.10 Fundamentación de las tecnologías y herramientas a utilizar

Después de haber abordado las distintas técnicas de programación, lenguajes, plataformas, herramientas y metodologías de desarrollo de *software*, se especifica la selección por parte del Departamento de Gestión Universitaria de la UCI el uso del paradigma de programación orientada a objetos, lenguaje de programación PHP 5.0, sobre la plataforma GNU/Linux, distribución Ubuntu 9.10, sistema gestor de base de datos PostgreSQL, *frameworks* jQuery y CodeIgniter, entorno de desarrollo NetBeans 6.7.1 y la metodología ágil Scrum-XP.

1.11 Conclusión parcial

En el desarrollo del capítulo se realiza un análisis de las técnicas de programación y lenguajes usados en el mundo, en Cuba y en la UCI, así como herramientas como sistemas gestores de base de datos, *frameworks* sobre los que se realizan aplicaciones *web* e IDEs empleados para la programación. También se comenta acerca de las plataformas sobre las que corren los sistemas y metodologías que rigen el proceso de desarrollo de *software*.

La elección de las tecnologías, herramientas y lenguajes de programación fue definida por el grupo de trabajo del Departamento de Gestión Universitaria, teniendo en cuenta las tendencias de *software* libre de la Universidad de las Ciencias Informáticas y del país, por el conocimiento previo del equipo de trabajo, y por la flexibilidad, potencia, y múltiples posibilidades que ofrecen.

De esta forma se concluye con la fundamentación teórica y se da paso a la descripción y análisis de la solución propuesta.

Capítulo 2. Descripción y análisis de la solución propuesta

2.1 Introducción

Para la implementación del módulo Registro y Control Docente, de acuerdo con el diseño realizado por los analistas, es necesario aplicar un estándar de codificación que haga el programa más comprensible para otros miembros del equipo de desarrollo y que cumpla con las funcionalidades necesarias, además de desglosarlas en tareas que permitan programar de una manera más clara y organizada.

2.2 Diseño propuesto por el analista

Utilizando la metodología Scrum-XP se analizó y diseñó el sistema por los analistas, generando como artefacto principal las Historias de Usuario (HU) que son entregadas al programador para la implementación.

Las HU representan una breve descripción del comportamiento del sistema, emplean terminología del cliente sin lenguaje técnico, se realiza una por cada característica principal del sistema, se emplean para hacer estimaciones de tiempo y para el plan de lanzamientos, reemplazan un gran documento de requisitos y presiden la creación de las pruebas de aceptación.

Estas deben proporcionar sólo el detalle suficiente como para poder hacer razonable la estimación de cuánto tiempo requiere la implementación de la HU, difiere de los casos de uso porque son escritos por el cliente, no por los programadores, empleando terminología del cliente.

Cada HU cuenta con una breve descripción de alguna funcionalidad en específico que debe cumplir el sistema y las interfaces correspondientes para la solución de las mismas.

Para la implementación del módulo Registro y Control Docente se han definido las siguientes HU:

- Gestionar nomencladores.
- Crear grupos docentes.
- Gestionar bonificaciones.
- Registro de asistencia.
- Registro de evaluaciones.

De este diseño propuesto por los analistas se pueden identificar las funcionalidades a implementar para que el sistema cumpla con las exigencias del cliente, basándose en la descripción detallada de las HU y la propuesta de diseño de interfaz.

2.2.1 Historia de Usuario Gestionar nomencladores

La HU permite gestionar los nomencladores**:** estado de asistencia y estado de grupos docentes que incluyen las siguientes actividades ubicadas en la barra flotante: Crear nomenclador, Mostrar nomenclador, así como las opciones Modificar nomenclador y Activar o Desactivar nomenclador.

Figura 1 Vista de nomencladores

Para Crear nomenclador se introducen los campos nombre, descripción y se selecciona la opción de estado activo. Se muestran los botones Aceptar y Cancelar, si selecciona Aceptar se guardan los datos y se muestra un mensaje notificando la acción, si selecciona Cancelar se muestra Mostrar nomenclador.

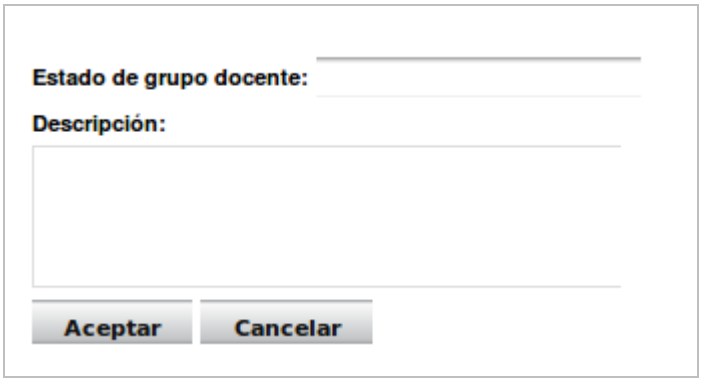

Figura 2 Crear nomenclador estado de grupo docente

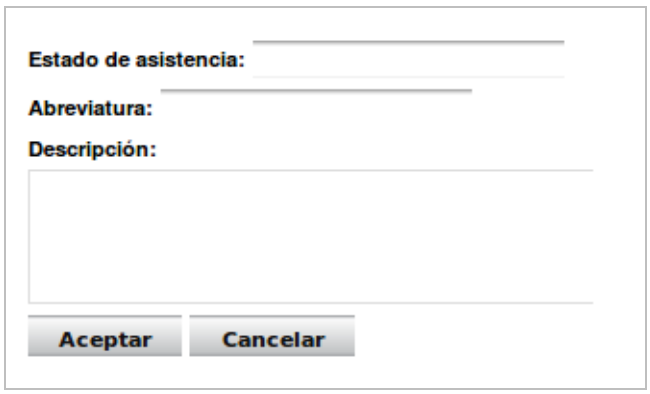

Figura 3 Crear nomenclador estado de asistencia

Para Mostrar nomenclador se muestra un listado con todos los nomencladores (según el tipo) creados anteriormente con los datos nombre, y la descripción. A partir del listado se muestra la opción Modificar nomenclador.

Figura 4 Mostrar nomenclador estado de grupo docente

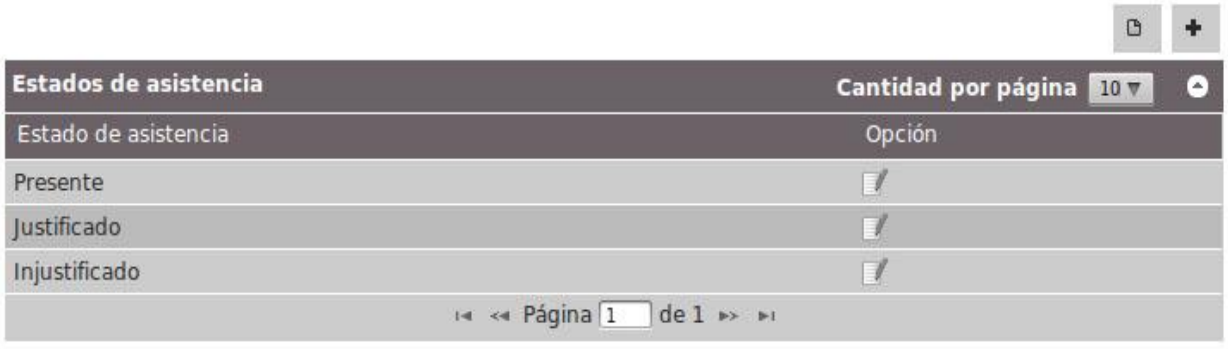

Figura 5 Mostrar nomenclador estado de asistencia

Para Modificar nomenclador se selecciona en Mostrar nomenclador la opción de Editar. Se muestran los datos que se pueden modificar de forma editable. Se muestran los botones Guardar cambios y Cancelar, si selecciona Guardar cambios se guardan los cambios como su nombre lo indica

y se muestra un mensaje notificando la acción, si selecciona Cancelar se muestra Mostrar nomenclador.

Figura 6 Modificar nomenclador estado de asistencia

2.3 Descripción de las nuevas clases u operaciones necesarias.

HU Gestionar nomencladores

Nombre: estado_asistencia	
Tipo de clase: controladora	
Atributo	**Tipo**
Para cada responsabilidad:	
Nombre:	index()
Descripción:	Carga la vista de registrar estado asistencia.
Nombre:	listar()
Descripción:	Renderiza la vista listar_estados_asistencia.
Nombre:	obtenerEstadosAsistencia()
Descripción:	Carga el grid* de los estados de asistencia.
Nombre:	crear()
Descripción:	Renderiza la vista registrar_estado_asistencia.
Nombre:	crearEstadoAsistencia()
Descripción:	Registra un estado de asistencia.
Nombre:	modificar($idEstadoAsistencia)

Descripción:	Renderiza la vista para modificar un estado de asistencia.
Nombre:	modificarEstadoAsistencia()
Descripción:	Modifica un estado de asistencia.

Tabla 1. Descripción de la clase controladora estado de asistencia.

Nombre: estado_grupo_docente	
Tipo de clase: controladora	
Atributo	**Tipo**
Para cada responsabilidad:	
Nombre:	index()
Descripción:	Carga la vista de registrar estado de grupo docente.
Nombre:	listar()
Descripción:	Renderiza la vista listar_estados_grupo_docente.
Nombre:	obtenerEstadosGrupoDocente()
Descripción:	Carga en un grid los estados de grupos docentes.
Nombre:	crear()
Descripción:	Renderiza la vista registrar_estado_grupo_docente.
Nombre:	crearEstadoGrupoDocente()
Descripción:	Registra un estado de grupo docente.
Nombre:	modificar($idEstadoGrupoDocente)
Descripción:	Renderiza la vista modificar_estado_grupo_docente.
Nombre:	modificarEstadoGrupoDocente()
Descripción:	Modifica un estado de grupo docente.

Tabla 2. Descripción de la clase controladora estado de grupo docente.

Nombre: tb_nestado_asistencia_base	
Tipo de clase: modelo	
Atributo	**Tipo**
Para cada responsabilidad:	
Nombre:	obtenerTbNestadoAsistenciaPorPagina($inicio,$limit,$elementoOrdenar,$d

	irOrdenar)
Descripción:	Devuelve un arreglo con los estados de asistencia.
Nombre:	registrarTbNestadoAsistencia($params)
Descripción:	Registra un *record* y devuelve el id generado.
Nombre:	modificarTbNestadoAsistencia($arrIds,$params)
Descripción:	Actualiza los datos de un registro de estado de asistencia.
Nombre:	obtenerTbNestadoAsistenciaDadoIdTbNestadoAsistencia($arrIds)
Descripción:	Devuelve los datos de un registro.
Nombre:	obtenerTbNestadoAsistenciaDadoAtributosTbNestadoAsistencia($params, $elementoOrdenar,$dirOrdenar)
Descripción:	Devuelve un arreglo con los datos.
Nombre:	obtenerCantidadTbNestadoAsistencia()
Descripción:	Devuelve la cantidad de elementos existentes en la tabla.

Tabla 3. Descripción de la clase modelo de estado de asistencia.

Nombre: tb_nestado_grupo_docente_base	
Tipo de clase: modelo	
Atributo	**Tipo**
Para cada responsabilidad:	
Nombre:	obtenerTbNestadoGrupoDocentePorPagina($inicio,$limite,$elementoOrdenar,$dirOrdenar)
Descripción:	Devuelve un arreglo con los estados de grupos docentes.
Nombre:	registrarTbNestadoGrupoDocente($params)
Descripción:	Registra un *record* y devuelve el id generado.
Nombre:	modificarTbNestadoGrupoDocente($arrIds,$params)
Descripción:	Actualiza los datos de un registro.
Nombre:	obtenerTbNestadoGrupoDocenteDadoIdTbNestadoGrupoDocente($arrIds)
Descripción:	Devuelve los datos de un registro.
Nombre:	obtenerTbNestadoGrupoDocenteDadoAtributosTbNestadoGrupoDocente($params,$elementoOrdenar,$dirOrdenar)
Descripción:	Devuelve un arreglo con los datos.
Nombre:	obtenerCantidadTbNestadoGrupoDocente()

Descripción:	Devuelve la cantidad de elementos existentes en la tabla.

Tabla 4. Descripción de la clase modelo de estado de grupo docente.

Nombre: estado_asistencia_lib	
Tipo de clase: librería	
Atributo	**Tipo**
Para cada responsabilidad:	
Nombre:	__construct()
Descripción:	Constructor de la clase, se instancia el *core* de CodeIgniter y se carga el *manager* de registro docente.
Nombre:	obtenerCantidadEstadosAsistencia()
Descripción:	Devuelve la cantidad de estados de asistencia.
Nombre:	obtenerEstadosAsistencia($inicio, $limite, $elementoOrdenar, $dirOrdenar)
Descripción:	Devuelve los estados de asistencia condicionados por lo parámetros establecidos.
Nombre:	registrarEstadoAsistencia($parametros)
Descripción:	Registra un nuevo estado de asistencia.
Nombre:	modificarEstadoAsistencia($parametros)
Descripción:	Modifica un estado de asistencia.
Nombre:	obtenerEstadoAsistenciaDadoIdEstadoAsistencia($idEstadoAsistencia)
Descripción:	Retorna un estado de asistencia según su id.
Nombre:	obtenerArregloAsociativoEstadosAsistencia()
Descripción:	Retorna un arreglo asociativo de estados de asistencia.

Tabla 5. Descripción de la clase librería de estado de asistencia.

Nombre: estado_grupo_docente_lib	
Tipo de clase: librería	
Atributo	**Tipo**
Para cada responsabilidad:	
Nombre:	__construct()

Descripción:	Constructor de la clase, se instancia el *core* de CodeIgniter y se carga el *manager* de registro docente.
Nombre:	obtenerCantidadEstadosGrupoDocente()
Descripción:	Devuelve la cantidad de estados de grupo docente.
Nombre:	obtenerEstadosGrupoDocente($inicio, $limite, $elementoOrdenar, $dirOrdenar)
Descripción:	Devuelve los estados de grupo docente condicionados por lo parámetros establecidos.
Nombre:	registrarEstadoGrupoDocente($parametros)
Descripción:	Registra un nuevo estado de grupo docente.
Nombre:	modificarEstadoGrupoDocente($parametros)
Descripción:	Modifica un estado de grupo docente.
Nombre:	obtenerEstadoGrupoDocenteDadoIdEstadoGrupoDocente($idEstadoGrupoDocente)
Descripción:	Devuelve un estado de grupo docente dado un id.
Nombre:	obtenerArregloAsociativoEstadosGrupoDocente()
Descripción:	Retorna un arreglo asociativo de estados de grupo docente.

Tabla 6. Descripción de la clase librería de estado de grupo docente.

Nombre: listar_estados_asistencia	
Tipo de clase: vista	
Atributo	**Tipo**
Para cada responsabilidad:	
Nombre:	load_grid_libs()
Descripción:	Carga los estados de asistencia en un grid.

Tabla 7. Descripción de la clase vista listar estado de asistencia.

Nombre: registrar_estado_asistencia	
Tipo de clase: vista	
Atributo	**Tipo**
Para cada responsabilidad:	

Nombre:	load_js()
Descripción:	Carga el js para validar la entrada de datos.
Nombre:	site_url()
Descripción:	Concatena al URL base.

Tabla 8. Descripción de la clase vista registrar estado de asistencia.

Nombre: modificar_estado_asistencia	
Tipo de clase: vista	
Atributo	**Tipo**
Para cada responsabilidad:	
Nombre:	load_js()
Descripción:	Carga el js para validar la entrada de datos.
Nombre:	site_url()
Descripción:	Concatena al URL base.

Tabla 9. Descripción de la clase vista modificar estado de asistencia.

Nombre: listar_estados_grupo_docente	
Tipo de clase: vista	
Atributo	**Tipo**
Para cada responsabilidad:	
Nombre:	load_navbar_libs()
Descripción:	Carga la barra de navegación.
Nombre:	load_grid_libs()
Descripción:	Carga los estados de grupo docente en un grid.
Nombre:	load_js()
Descripción:	Carga la barra de botones.

Tabla 10. Descripción de la clase vista listar estado de grupo docente.

Nombre: modificar_estado_grupo_docente

Tipo de clase: vista	
Atributo	**Tipo**
Para cada responsabilidad:	
Nombre:	load_navbar_libs()
Descripción:	Carga la barra de navegación.
Nombre:	load_js()
Descripción:	Carga la barra de botones.
Nombre:	load_js()
Descripción:	Carga el js para validar los datos de entrada.
Nombre:	site_url()
Descripción:	Concatena al URL base.

Tabla 11. Descripción de la clase vista modificar estado de grupo docente.

Nombre: registrar_estado_grupo_docente	
Tipo de clase: vista	
Atributo	**Tipo**
Para cada responsabilidad:	
Nombre:	load_navbar_libs()
Descripción:	Carga la barra de navegación.
Nombre:	load_js()
Descripción:	Carga la barra de botones.
Nombre:	load_js()
Descripción:	Carga el js para validar los datos de entrada.
Nombre:	site_url()
Descripción:	Concatena al URL base.

Tabla 12. Descripción de la clase vista registrar estado de grupo docente.

2.4 Estilos de programación

Los aspectos que normalmente se denominan "estilo" son aspectos relacionados a los lenguajes como medio de comunicación entre personas, y que usualmente no influyen en la comunicación humano-máquina.

Las reglas de estilo son flexibles. Esto no significa que uno va escribiendo y cambiando de estilo. Es muy importante dentro de un mismo proyecto mantener siempre las mismas reglas rígidas, aunque estas sean distintas a las que uno usa en otros proyectos.

Incluso, cuando se trabaja sobre un proyecto escrito por otro, es mejor adaptarse al estilo en que está escrito en vez de mezclarlos.

No basta con escribir un programa que funcione. El código tiene que estar bien escrito. El problema del estilo es muy recurrente en el desarrollo de *software*. Muchas veces se escribe el código pensando que la única persona que lo modificará es el mismo programador, y cuando llega alguien más, y comienza a revisar el código, comienzan los problemas. Peor aún es cuando se mezclan estilos de programación.

Así, la meta final del programador es construir programas. Y el ideal es construir "buenos" programas. Hay diversas cualidades generalmente aceptadas de lo que es un programa "bueno", y cualquier herramienta, técnica o método que ayude a mejorar esas cualidades es aceptado.

Las cualidades que se ven beneficiadas de forma más directa por un buen estilo son:

- **Extensibilidad:** facilidad con que se adapta el *software* a cambios de especificación. Un buen estilo de código fomenta programas que no sólo resuelven el problema, sino que también reflejan claramente la relación problema/solución. Esto tiene como efecto que muchos cambios simples en el problema reflejen de forma clara los cambios a hacer en el programa.
- **Verificabilidad:** facilidad con que pueden comprobarse propiedades de un sistema. Si el estilo de código hace obvia la estructura del programa, eso ayuda a verificar que el comportamiento sea el esperado.
- **Reparabilidad:** la posibilidad de corregir errores sin demasiado esfuerzo.
- **Capacidad de evolución:** la capacidad de adaptarse a nuevas necesidades.
- **Comprensibilidad:** la facilidad con que el programa puede ser comprendido.

2.5 Estándares de codificación

2.5.1 Identación, llaves de apertura y cierre, y tamaño de las líneas

En el desarrollo del sistema de Gestión Universitaria, se ha establecido un estándar de codificación, como usar una identación* sin tabulaciones, con un equivalente a 4 espacios, para mantener integridad en las revisiones svn*. El uso de las llaves "{}" es en una nueva línea. La longitud de las líneas de código es aproximadamente de 75-80 caracteres para mantener la legibilidad del código.

Ejemplo:

```
1 ....$a = $b;
2
3 ....function ejemplo()
4 ....{
5     ....//BI
6 ....}
```

Figura 7 Identación y llaves

2.5.2 Conversión de nomenclatura

Las variables se rigen por la nomenclatura camelCase. Siempre comienzan con minúscula y en caso de nombres compuestos la primera letra de cada palabra comienza con mayúscula.

Ejemplo:

```
1 ....$variable
2 ....$variableNombreCompuesto
```

Figura 8 Variables

Las constantes son escritas en mayúsculas, con caracteres de subrayado "_" para separar palabras en caso de nombres compuestos.

Ejemplo:

* Ver Glosario de términos

```
1 ....define(CONSTANTE,valor);
2 ....define(CONSTANTE_COMPUESTO,valor);
```

Figura 9 Constantes

Los nombres de las clases siempre comienzan con mayúscula, en caso de nombre compuesto las palabras se separan con el carácter subrayado "_" y el resto en minúscula.

Ejemplo:

```
1 ....class Clase
2 ....{
3     ....//BI
4 ....}
5
6 ....class Clase_nombre_compuesto
7 ....{
8     ....//BI
9 ....}
```

Figura 10 Clases

Las funciones se rigen por la nomenclatura camelCase. Siempre comienzan con minúscula y en caso de nombres compuestos la primera letra de cada palabra comienza con mayúscula. Los parámetros son separados por espacio luego de la coma que los separa.

Ejemplo:

```
1 ....function funcion($parametro1,.$parametro2)
2 ....{
3     ....//BI
4 ....}
5
6 ....function funcionNombreCompuesto($parametro1,.$parametro2)
7 ....{
8     ....//BI
9 ....}
```

Figura 11 Funciones y parámetros

Los ficheros siempre se escriben con minúscula y en caso de nombres compuestos se usa el carácter subrayado "_".

- **Vistas:** intuitivo y relacionado con el formulario y/o vista que representa.
- **Modelos:** con el mismo nombre de la clase que representa que contiene en el nombre el sufijo _mdl o _base en caso de ser modelos base.
- **Librerías:** con el mismo nombre de la clase que representa que contiene en el nombre el sufijo: _lib.
- **Controladoras:** con el mismo nombre de la clase que representa.
- **Manager:** con el mismo nombre de la clase que representa que contiene en el nombre el sufijo: _mng.

2.5.3 Estructuras de control

Se incluye un espacio entre las estructuras de control (*if, for, foreach, while, switch*) y los paréntesis. Se recomienda utilizar siempre llaves de apertura y cierre, incluso en situaciones en las que técnicamente son opcionales. Esto aumenta la legibilidad y disminuye la probabilidad de errores lógicos.

Ejemplo:

```
1 ....if.(condicion)
2 ....{
3     ....//BI
4 ....}
5 ....elseif(condicion)
6 ....{
7     ....//BI
8 ...}
9 ....else
```

```
10....{
11    ....//BI
12....}
13
14....switch.(valor)
15....{
16    ....case valor1:
17        ....//BI para valor1
18    ....break;
19    ....case valor2:
20        ....//BI para valor2
21    ....break;
22    ....default:
23        ....//BI por defecto
24....}
```

Figura 12 Estructuras de control

Si las condiciones son muy largas que sobrepasan el tamaño de la línea, éstas se dividen en varias líneas.

Ejemplo:

```
1 ....if.(condicion1
2 ....|| condicion2)
3 ....||(condicion3
4 ....&& condicion4))
5 ....{
6     ....//BI
7 ....}
```

Figura 13 Condiciones en nuevas líneas

En caso de que la condición sea muy extensa, se puede dividir en variables y compararlas dentro de la estructura de control.

Ejemplo:

```
1 ....$variableCondicion1 = condicion1 || condicion2;
2 ....$variableCondicion2 = condicion3 && condicion4;
3
4 ....if.($variableCondicion1 || $variableCondicion2)
5 ....{
6     ....//BI
7 ....}
```

Figura 14 Condiciones usando variables

2.5.4 Documentación

Todos los archivos deben de tener la documentación asociada al mismo. Para esto se debe cumplir con el siguiente bloque al principio de cada clase.

Clase:

```
1 /**
2 *Breve descripción de la clase
3 *
4 *PHP versión #
5 *
6 *@category Categoría de la clase implementada "Libreria,
7 * Controladora, Modelo"
8 *@package  Nombre del paquete o módulo al que pertence
9 *@author  Nombre y Apellidos del autor y  correo electrónico
10*/
```

Figura 15 Descripción de clases

Funciones:

```
1 /**
2 *Breve descripción de la función
3 *
4 *@param tipo y nombre del paremetro(por cada parametro que
5 * recibe la función)
6 *@return  tipo que retorna
7 *@author  Nombre y Apellidos del autor y  correo electrónico
8 */
```

Figura 16 Descripción de funciones

2.5.5 Buenas prácticas

Los valores booleanos y nulos siempre se escriben con mayúscula, para facilitar la legibilidad del código usar una línea en blanco antes de las estructuras de control y definición de las funciones.

```
1 ....$variableBooleana = FALSE;
2 ....$variableNula = NULL;
3 ....
4 ....if(condicion)
5 ....{
6     ....//BI
7 ....}
```

Figura 17 Buenas prácticas

2.5.6 Pautas para la implementación de las HU

El desarrollo en la metodología utilizada por el sistema de Gestión Universitaria está guiado por las HU y a su vez el desarrollo de estas HU estará guiado y dirigido por las Tareas de Ingeniería que se generan de una HU.

Para la implementación de las Tareas de Ingeniería se definen algunos patrones a seguir:

1. Regirse por el estándar de código definido para la implementación del proyecto.
2. Todas las interfaces interactúan mediante AJAX.
3. Las acciones se dividen generalmente en dos métodos en las controladoras para lograr una mayor legibilidad del código:
    - Para obtener los datos necesarios en caso de que lo necesite y mostrar la vista. El nombre usado para estos métodos se escribe en infinitivo, ejemplo: crear, listar, modificar, detallar, asociar.
    - Para obtener y validar los datos recibidos desde la vista, interactúa con la librería asociada y crear el mensaje que se envía a la vista. El nombre usado es el mismo infinitivo usado en el otro método con el o los elementos afectados en la acción ejemplo: crearGrupo, modificarGrupo, asociarTrabajadorGrupo, en el caso de que no sea necesario mostrar una vista, éste se escribe en infinitivo ejemplo: eliminar, activar.
4. En los métodos implementados en las clases modelo, las consultas no se hacen con código SQL directamente, se realiza utilizando el *active record* de CodeIgniter.

Ejemplo:

```
1 ....$this->db->select('sq_esquema.tb_ddatos.id_datos');
3 ....$this->db->where ($key, $value);
4 ....$this->db->get ('sq_esquema.tb_ddatos')->result();
```

Figura 18 Consultas

5. Para la creación de las interfaces se utilizarán las funciones del *helper form* de CodeIgniter para garantizar homogeneidad en el html, ejemplo: form_open, form_dropdown, form_input.

6. Los mensajes de error en la vista se lanzan con un *throw Exception* (“mensaje”).

7. No se utiliza el .load de jQuery pues el envío de los datos los hace por *get* y para garantizar un poco de seguridad se deben enviar por *post* siempre, para esto hay que utilizar $.AJAX.

8. En los métodos de las controladoras que serán encuestados mediante AJAX, se utiliza *echo** y no *die** para mostrar los datos o la vista cargada.

9. Todas las implementaciones se realizan en español. (Dirección de Calidad de la Infraestructura Productiva, 2010)

2.5.6.1 Tareas de Ingeniería

Las Tareas de Ingeniería (TI) son actividades que los programadores conocen que el sistema debe hacer. Deben ser estimables, y poder ser implementadas entre uno y tres días ideales. La mayoría de estas tareas se derivan directamente de las HU.

Existen dos tipos de TI, las que provienen de las Historias de Usuario y las técnicas.

Cada Historia de Usuario puede ser dividida en varias Tareas de Ingeniería sencillas. Para determinar las tareas que componen a una HU se propone hacer una reunión con todos los miembros del equipo de desarrollo y obtener una buena lista con todas.

Las TI técnicas son aquellas que no son resultado del análisis de ninguna HU pero deben ser realizadas para que el sistema funcione.

Cada tarea de ingeniería será comprobada a través de los casos de prueba y no tienen por qué ser comprendidas por el cliente.

Cada una de estas HU se transformará en tareas que serán desarrolladas por programadores, dentro del equipo de desarrollo.

Descripción de las Tareas de Ingeniería de Gestionar nomencladores

Tarea de Ingeniería	
Número tarea: HU1-1	Número historia: 1

Nombre tarea: Desarrollar los formularios que permitan crear el nomenclador estado de asistencia.	
Tipo de tarea: Desarrollo	Puntos estimados: 1 semana.
Fecha de inicio: 1/04/2010	Fecha de fin: 7/04/2010
Programador responsable: Lidiana Hernández Legrá y Yudileidis Peña Carballosa.	
Descripción: Crear el código que permita crear el nomenclador estado de asistencia. Se especifica el estado y la descripción.	

Tabla 13. Tarea de Ingeniería HU Gestionar nomenclador.

Tarea de Ingeniería	
Número tarea: HU1-2	Número historia: 1
Nombre tarea: Desarrollar los formularios que permitan mostrar nomenclador estado de asistencia.	
Tipo de tarea: Desarrollo	Puntos estimados:1 semana
Fecha de inicio: 8/04/2010	Fecha de fin: 14/04/2010
Programador responsable: Lidiana Hernández Legrá y Yudileidis Peña Carballosa.	
Descripción: Crear un grid que permita mostrar el nomenclador estado de asistencia.	

Tabla 14. Tarea de Ingeniería HU Gestionar nomenclador.

Tarea de Ingeniería	
Número tarea: HU1-3	Número historia: 1
Nombre tarea: Desarrollar los formularios que permitan modificar el nomenclador estado de asistencia.	
Tipo de tarea: Desarrollo	Puntos estimados:
Fecha de inicio: 15/04/2010	Fecha de fin: 21/04/2010
Programador responsable: Lidiana Hernández Legrá y Yudileidis Peña Carballosa.	
Descripción: Implementar el código que modifique el nomenclador estado de	

asistencia. Se especifica el estado y la descripción.

Tabla 15. Tarea de Ingeniería HU Gestionar nomenclador.

Tarea de Ingeniería	
Número tarea: HU1-4	Número historia: 1
Nombre tarea: Desarrollar los formularios que permitan crear el nomenclador estado de grupo docente.	
Tipo de tarea: Desarrollo	Puntos estimados: 1 semana.
Fecha de inicio: 22/04/2010	Fecha de fin: 28/04/2010
Programador responsable: Lidiana Hernández Legrá y Yudileidis Peña Carballosa.	
Descripción: Implementar el código que permita crear el nomenclador estado de grupo docente. Se especifica el estado y la descripción.	

Tabla 16. Tarea de Ingeniería HU Gestionar nomenclador.

Tarea de Ingeniería	
Número tarea: HU1-5	Número historia: 1
Nombre tarea: Desarrollar los formularios que permitan mostrar el nomenclador estado de grupo docente.	
Tipo de tarea: Desarrollo	Puntos estimados: 1 semana.
Fecha de inicio: 29/04/2010	Fecha de fin: 5/05/2010
Programador responsable: Lidiana Hernández Legrá y Yudileidis Peña Carballosa.	
Descripción: Crear un grid que muestre el nomenclador estado de grupo docente.	

Tabla 17. Tarea de Ingeniería HU Gestionar nomenclador.

Tarea de Ingeniería	
Número tarea: HU1-3	Número historia: 1
Nombre tarea: Desarrollar los formularios que permitan modificar el nomenclador estado de grupo docente.	

Tipo de tarea: Desarrollo	Puntos estimados: 1 semana.
Fecha de inicio: 6/05/2010	Fecha de fin: 12/05/2010
Programador responsable: Lidiana Hernández Legrá y Yudileidis Peña Carballosa.	
Descripción: Implementar el código que modifique el nomenclador estado de grupo docente. Se especifica el estado y la descripción.	

Tabla 18. Tarea de Ingeniería HU Gestionar nomenclador.

2.6 Conclusión parcial

Al finalizar este capítulo queda implementado el sistema y descritas todas las clases utilizadas, cumpliendo con los objetivos específicos planteados y gran parte de las tareas propuestas para la elaboración del sistema. Se describen brevemente las funcionalidades implementadas, el estándar de codificación definido y las Tareas de Ingeniería que guían al programador a la hora de implementar. De esta manera se da paso a la etapa de pruebas para asegurar que el sistema esté libre de no conformidades y con la debida calidad para ser entregado al cliente.

Capítulo 3. Validación de la solución propuesta

3.1 Introducción

Para asegurar que una HU está terminada y que cumple con las necesidades del usuario final, es necesario probarla y verificar si los resultados son los esperados. Existen variadas formas de probar una aplicación, depende en gran medida, de la metodología de desarrollo por la que se rija el proyecto.

3.2 Pruebas de unidad que permiten validar la solución propuesta

Las pruebas son actividades en las cuales un sistema o componente es ejecutado bajo condiciones o requerimientos específicos. Los resultados son observados, registrados y se realiza una evaluación del sistema o componente, verificando de esta forma los resultados de la implementación del sistema.

Uno de los pilares fundamentales que propone la unión de las metodologías Scrum-XP es el proceso de pruebas, donde se anima a los desarrolladores a probar constantemente tanto como sea posible. Mediante esta filosofía se reduce el número de errores no detectados así como el tiempo entre la introducción de éste en el sistema y su detección. Todo esto contribuye a elevar la calidad de los productos desarrollados y a la seguridad de los programadores a la hora de introducir cambios o modificaciones. La metodología divide las pruebas en dos grupos: pruebas unitarias, desarrolladas por los programadores, encargadas de verificar el código de forma automática y las pruebas de aceptación, destinadas a evaluar si al final de una iteración se obtuvo la funcionalidad requerida, además de comprobar que dicha funcionalidad sea la esperada por el cliente. (Bartumeu, y otros, 2009)

Las pruebas de aceptación son más importantes que las pruebas unitarias dado que significan la satisfacción del cliente con el producto desarrollado y el final de una iteración y el comienzo de la siguiente.

Mientras un código no haya sido probado no existe. Estos pueden ser adicionados o eliminados en cualquier momento. El objetivo es tener una forma de que el cliente conozca cuándo una HU está lista. Los casos de prueba se deben escribir antes de comenzar la implementación, pero siempre que sea necesario se puede incluir uno nuevo. No existe restricción de cantidad para estos, se deben

escribir casos de prueba hasta que quede claro el objetivo de la HU y se verifique que cumpla con todos los requerimientos. (Louit Moreno, y otros, 2009)

3.3 Objetivos de las pruebas

- Encontrar y documentar los defectos que puedan afectar la calidad del *software*.
- Verificar que el *software* trabaje como fue diseñado.
- Validar y probar los requisitos que debe cumplir el *software*.
- Validar que los requisitos fueron implementados correctamente.

3.4 Pruebas unitarias

Desarrolladas por los programadores, las pruebas unitarias consisten en comprobaciones manuales o automatizadas realizadas para verificar que el código correspondiente a un módulo funciona de acuerdo con los requisitos del sistema, son realizadas a pequeña escala. Este tipo de pruebas facilita que ante un error el código sea cambiado por el programador para mejorar su estructura. Permite seguir realizando pruebas sobre los cambios para asegurarse de que no se han introducido nuevos errores y, por consiguiente, reduce los efectos secundarios de éstos.

3.5 Pruebas de caja negra

Las pruebas de caja negra se refieren a las pruebas que se llevan a cabo sobre la interfaz del *software*, las cuales permiten obtener un conjunto de condiciones de entrada que ejerciten completamente todos los requisitos funcionales del programa.

El objetivo de realizar estas pruebas al sistema es para revelar el incorrecto o incompleto funcionamiento de éste, así como los errores de interfaz, rendimiento y errores de inicialización y terminación.

El proceso se centra principalmente en los requisitos funcionales del *software* para verificar el comportamiento de la unidad observable externamente y la calidad funcional.

Se llevan a cabo sobre la interfaz del *software*, y son completamente indiferentes al comportamiento interno y la estructura del programa. Cada uno de los casos de prueba de caja negra pretenden demostrar que:

- Las funciones del *software* son operativas.
- La entrada se acepta de forma adecuada.
- Se produce una salida correcta.

- La integridad de la información externa se mantiene.

E intentan encontrar errores de las siguientes categorías:

- Funciones incorrectas o ausentes.
- Errores de interfaz.
- Errores en estructuras de datos o en accesos a bases de datos externas.
- Errores de rendimiento.

3.6 Pruebas de aceptación

Las pruebas de aceptación son pruebas de caja negra que se crean a partir de las HU. Destinadas a evaluar si al final de una iteración se obtuvo la funcionalidad requerida, además de comprobar que dicha funcionalidad sea la esperada por el cliente.

Durante las iteraciones, las HU seleccionadas serán traducidas a pruebas de aceptación, en ellas se especifican, desde la perspectiva del cliente (ya que son elaboradas conjuntamente con éste), los escenarios para probar que las mismas han sido implementadas correctamente. Una HU puede tener todas las pruebas de aceptación que necesite para asegurar su correcto funcionamiento y no se considera completa hasta que no ha pasado por sus pruebas de aceptación.

3.7 Casos de prueba

En la metodología, las pruebas de funcionalidad se realizan a través de los casos de prueba y se escriben para cada HU que deba validarse. Estas pruebas son ejecutadas y sus resultados son difundidos por el equipo de desarrollo.

3.7.1 Escenario “Crear estado de asistencia”

Caso de prueba
Identificación de la HU a probar: CP1-HU1-Gestionar nomencladores
Descripción de la funcionalidad: Pruebas para crear el nomenclador estado de asistencia.
Condiciones de ejecución: Se deben introducir los campos Estado, Descripción y se seleccionar la opción de estado Activo. Solo tiene acceso a realizar esta acción el administrador del sistema. Estado de la asistencia: Presente, Justificado, Injustificado.
Flujo central: Opción del menú “Crear estado de asistencia” / Área de iconos flotantes “Crear”.

Tabla 19. Descripción del caso de prueba crear estado de asistencia.

Clases válidas	Resultado esperado	Resultado de la prueba	Observaciones
Introducción correcta del estado de la asistencia. “Estado de asistencia: Presente” “Letra de identificación: P” “Descripción: Prueba” **“**Activo: True**”**	1. Tras la introducción de un estado de asistencia, si el proceso ha sido correcto, en la base de datos aparecerán los datos del nuevo estado de asistencia del estudiante. 2. Se muestra un mensaje que indica que la acción ha sido realizada satisfactoriamente.	Satisfactoria	
Clases inválidas	**Resultado esperado**	**Resultado de la prueba**	**Observaciones**
Introducción incorrecta del estado de asistencia. “Estado de asistencia: Pro·$%&/()=?123358” “Letra de identificación: rt1” “Descripción: Prueba” “Activo: True”	1. Mostrará un mensaje indicando que la información del estado de asistencia no ha sido introducida correctamente. 2. El estado de la asistencia introducida incorrectamente no es insertada en la base de datos.	Satisfactoria	
Clases inválidas	**Resultado esperado**	**Resultado de la prueba**	**Observaciones**
Introducción del estado de asistencia ya insertado.	1. Se muestra un mensaje indicando que el estado de asistencia ya ha sido introducido. 2. El estado de	Satisfactoria	

	asistencia no vuelve a ser incluido en la base de datos.		
Clases inválidas	**Resultado esperado**	**Resultado de la prueba**	**Observaciones**
Introducción de los campos obligatorios de estado de asistencia sin llenar. “Estado:*” “Letra de identificación:*” “Descripción: Prueba” “Activo: True”	1. Se muestra un mensaje indicando que no se han introducido todos los campos obligatorios. 2. El estado de asistencia no es introducido en la base de datos.	Satisfactoria	
Evaluación de la prueba:	Satisfactoria		

Tabla 20. Validación del caso de prueba crear estado de asistencia.

3.7.2 Escenario “Modificar estado de asistencia”

Caso de prueba
Identificación de la HU a probar: CP2-HU1-Gestionar nomencladores
Descripción de la funcionalidad: Pruebas para modificar el nomenclador estado de asistencia.
Condiciones de ejecución: Se deben introducir los campos Estado, Descripción y seleccionar la opción de estado Activo. Sólo tiene acceso a realizar esta acción el administrador del sistema. Estado de la asistencia: Presente, Justificado, Injustificado.
Flujo central: opción del menú “Modificar estado de asistencia” / Área de íconos flotantes “Modificar”.

Tabla 21. Descripción del caso de prueba modificar estado de asistencia.

Clases válidas	Resultado esperado	Resultado de la prueba	Observaciones
Introducción correcta del estado de la asistencia. “Estado de asistencia: Presente” “Letra de identificación: P” “Descripción: Prueba” **“**Activo: True**”**	1. Tras la introducción del nuevo estado de asistencia, cambio de Injustificado a Presente, si el proceso ha sido correcto, en la base de datos cambia Injustificado a Presente. 2. Se muestra un mensaje que indica que la acción ha sido realizada satisfactoriamente.	Satisfactorio	
Clases inválidas	**Resultado esperado**	**Resultado de la prueba**	**Observaciones**
Introducción incorrecta del estado de asistencia. “Estado de asistencia: Pro·$%&/()=?123358” “Letra de identificación: rt1” “Descripción: Prueba” “Activo: True”	1. Mostrará un mensaje indicando que la información del estado de asistencia no ha sido introducida correctamente. 2. El estado de asistencia que fue insertado incorrectamente no es introducido en la base de datos.	Satisfactorio	
Clases inválidas	**Resultado esperado**	**Resultado de la prueba**	**Observaciones**
Introducción del estado de asistencia ya insertado.	1. Se muestra un mensaje indicando que el estado ha sido introducido previamente.	Satisfactorio	

	2. El estado de asistencia no vuelve a ser introducido en la base de datos.		
Clases inválidas	**Resultado esperado**	**Resultado de la prueba**	**Observaciones**
Introducción de campos obligatorios de estado de asistencia sin llenar. “Estado:*” “Letra de identificación:*” “Descripción: Prueba” “Activo: True”	1. Se muestra un mensaje indicando que no se han introducido todos los campos obligatorios. 2. El estado de asistencia no fue introducido en la base de datos.	Satisfactorio	
Evaluación de la prueba:	Satisfactoria		

Tabla 22. Validación del caso de prueba modificar estado de asistencia.

3.7.3 Escenario “Crear estado de grupo docente”

Caso de prueba
Identificación de la HU a probar: CP3-HU1-Gestionar nomencladores
Descripción de la funcionalidad: Pruebas para crear el nomenclador estado de grupo docente.
Condiciones de ejecución: Se deben introducir los campos Estado de grupo docente, Descripción y seleccionar la opción de estado Activo. Sólo tiene acceso a realizar esta acción el administrador del sistema. Estado de grupo docente: Listo, Crítico y Cerrado.
Flujo central: opción del menú “Crear estado de grupo docente” / Área de íconos flotantes “Crear”.

Tabla 23. Descripción del caso de prueba crear estado de grupo docente.

Clases válidas	Resultado esperado	Resultado de la prueba	Observaciones
Introducción correcta de estado de grupo docente. "Estado de grupo docente: Listo" "Descripción: Prueba" "Activo: True"	1. Tras la introducción de un estado de grupo docente si el proceso ha sido correcto, en la base de datos aparecerán los datos del nuevo estado de grupo docente. 2. Se muestra un mensaje que indica que la acción ha sido realizada satisfactoriamente.	Satisfactorio	
Clases inválidas	**Resultado esperado**	**Resultado de la prueba**	**Observaciones**
Introducción incorrecta del estado de grupo docente. "Estado grupo docente: $%&/()==??14587_9 " "Descripción: Prueba" "Activo: True"	1. Se mostrará un mensaje indicando que el elemento no ha sido introducido incorrectamente. 2. El estado de grupo docente que fue insertado incorrectamente no es introducido en la base de datos.	Satisfactorio	
Clases inválidas	**Resultado esperado**	**Resultado de la prueba**	**Observaciones**
Introducción de un estado de grupo docente ya insertado.	1. Se muestra un mensaje indicando que el estado de grupo docente ya ha sido introducido. 2. El estado de grupo docente no vuelve a introducirse en la base de datos.	Satisfactorio	
Clases inválidas	**Resultado esperado**	**Resultado de la**	**Observaciones**

		prueba	
Omitir campos obligatorios del estado de grupo docente. “Estado de grupo:*” “Descripción: Prueba” “Activo: True”	1. Se muestra un mensaje indicando que no se han introducido todos los campos obligatorios. 2. El estado de grupo docente no es introducido en la base de datos.	Satisfactorio	
Evaluación de la prueba:	Satisfactoria		

Tabla 24. Validación del caso de prueba crear estado de grupo docente.

3.7.4 Escenario “Modificar estado de grupo docente”

Caso de prueba
Identificación de la HU a probar: CP4-HU1-Gestionar nomencladores
Descripción de la funcionalidad: Pruebas para modificar el nomenclador estado de grupo docente.
Condiciones de ejecución: Se deben introducir los campos Estado, Descripción y seleccionar la opción de estado Activo. Sólo tiene acceso a realizar esta acción el administrador del sistema. Estado de la asistencia: Listo, Crítico y Cerrado.
Flujo central: opción del menú “Modificar estado de grupo docente” / Área de íconos flotantes “Modificar”.

Tabla 25. Descripción del caso de prueba modificar estado de grupo docente.

Clases válidas	Resultado esperado	Resultado de la prueba	Observaciones
Introducción correcta del estado del grupo docente.	1. Tras la introducción del nuevo estado de grupo docente, cambio de Listo a	Satisfactorio	

“Estado de grupo docente: Crítico” “Descripción: Prueba” **“**Activo: True**”**	Crítico, si el proceso ha sido correcto, en la base de datos se cambia de Listo a Crítico. 2. Se muestra un mensaje que indica que la acción ha sido realizada satisfactoriamente.		
Clases inválidas	**Resultado esperado**	**Resultado de la prueba**	**Observaciones**
Introducción incorrecta del estado de grupo docente. “Estado de grupo docente: Pro·$%&/()=?123358” “Descripción: Prueba” “Activo: True”	1. Mostrará un mensaje indicando que la información del estado de grupo docente no ha sido introducida correctamente. 2. El estado de grupo docente que fue introducido incorrectamente no es insertado en la base de datos.	Satisfactorio	
Clases inválidas	**Resultado esperado**	**Resultado de la prueba**	**Observaciones**
Introducción del estado de grupo docente ya insertado.	1. Se muestra un mensaje indicando que el estado ha sido introducido previamente. 2. El estado de grupo docente no vuelve a ser insertado en la base de datos.	Satisfactorio	
Clases inválidas	**Resultado esperado**	**Resultado de la prueba**	**Observaciones**
Introducción de campos obligatorios de estado de	1. Se muestra un mensaje indicando que no se han	Satisfactorio	

grupo docente sin llenar. “Estado:*” “Descripción: Prueba” “Activo: True”	introducido todos los campos obligatorios. 2. El estado de grupo docente no fue insertado en la base de datos.		
Evaluación de la prueba:	Satisfactoria		

Tabla 26. Validación del caso de prueba modificar estado de grupo docente.

3.8 Conclusión parcial

Con las pruebas realizadas, se garantiza que el módulo Registro y Control Docente del sistema de Gestión Universitaria queda libre de errores y de no conformidades, ya que el resultado de los casos de prueba fue satisfactorio, asegurando la calidad del sistema desarrollado y la aceptación del cliente.

Conclusiones generales

Con la implementación del módulo Registro y Control Docente utilizando tecnologías y herramientas de código abierto se logró:

- Gestionar las bonificaciones que se le otorgan a un estudiante en todo el transcurso de la carrera.
- Gestionar el registro de asistencia perteneciente a un grupo docente por asignatura.
- Gestionar el registro de evaluaciones perteneciente a un grupo docente por asignatura, examen final de disciplina o trabajo de diploma.
- Gestionar los grupos docentes pertenecientes a una facultad, una carrera y una modalidad de estudio.
- Gestionar los nomencladores estado de asistencia y estado de grupos docentes.

Cumpliéndose de esta manera el objetivo principal de la investigación y dando solución a la situación existente con el uso de Akademos, habiéndose mejorado la gestión de los procesos de control docente en la universidad.

Recomendaciones

Una vez vencidos los objetivos de esta investigación, teniendo en cuenta las experiencias obtenidas a lo largo de su desarrollo, se recomienda:

- Realizar refactorización al código programado.
- Mejorar las funcionalidades del *framework* para llegar a una mejor comunicación entre módulos y subsistemas.
- Continuar el trabajo de implementación del módulo en nuevas iteraciones.

Bibliografía

Alvarez, Miguel Angel. 2010. desarrolloweb.com. [En línea] 2010. http://www.desarrolloweb.com/articulos/2477.php.

Álvarez, Miguel Angel. 2009. desarrolloweb.com. [En línea] 25 de Marzo de 2009. http://www.desarrolloweb.com/articulos/introduccion-jquery.html.

Anaya Villegas, Adrian. 2009. *A propósito de programación extrema XP (eXtreme Programming).* 2009.

Babylon Programa de Traducción. 2009. Diccionario babylon. [En línea] 2009.

Bartumeu, Carlos Monzón y Pérez, Ortega Adrian. 2009. *Implementación de la Intranet de la Facultad 1. Módulo Docencia Autores.* 2009.

Bello del Pino, Daily y Fernández Fernández, Susana Alicia. 2009. *Desarrollo del Sistema para la Reservación de Tiempos de Máquina de los Laboratorios en la Universidad de las Ciencias Informática.* 2009.

de Pereda, Jose María. 2007. JMPereda´s Weblog. [En línea] 24 de Agosto de 2007. http://jmpereda.wordpress.com/2007/08/24/definiendo-la-plantilla/.

Departamento de Técnicas de Programación. 2010. Entorno Virtual de Aprendizaje. [En línea] 2010. http://eva.uci.cu/mod/resource/view.php?id=20103.

Díaz, J. F. 2010. galeon.com (hispavista). *NeoProgramadores.* [En línea] 2010. http://www.galeon.com/neoprogramadores/t_dis_sw.htm#Monolitica.

Dirección de Calidad de la Infraestructura Productiva. 2010. *Estándar de código CENIA.* 2010.

Drake, José M. 2008. *PROGRAMACION CONCURRENTE, Recursos para la concurrencia.* 2008.

Ellis, Rick y Burdick, Paul. 2009. MANUAL DE CodeIgniter. [En línea] 2009. http://lax.franhp.net/CodeIgniter_Spanish_UserGuide.pdf.

Ferriol Ortiz, Acralys y Azahares Reyes, Enmanuel. 2009. *Tesis de Análisis y Diseño del módulo Registro y Control Docente para Akademos v2.0.* 2009.

González Castellanos, Ma. Arnenis y Rojas Pabón, Wilson. 2005. *TRABAJO DE GRADO: Comparación entre sistemas de gestión de bases de datos (SGBD) bajo licenciamiento libre y comercial (pdf).* 2005.

González, Dr. Brambila y Beatriz, Silvia. 2009. Departamento de Sistemas. [En línea] 2009. http://delfosis.uam.mx/~sgb/docs/INTRO.pdf.

HispaNetwork Publicidad y Servicios. 2007. glosario.net. [En línea] 16 de Marzo de 2007. http://tecnologia.glosario.net/terminos-viricos/lenguaje-de-programaci%F3n-9768.html.

2009. Lenguajes de Programación. [En línea] 2009. http://www.lenguajes-de-programacion.com/programacion-orientada-a-objetos.shtml.

Letelier, Patricio y Penadés, Mª Carmen. 2010. WillyDev. [En línea] 2010. http://www.willydev.net/descargas/masyxp.pdf..

Louit Moreno, Annia y Trueba Taillacq, Emilio Francisco. 2009. *SWAP, Sistema Web para la verificación de autenticidad de los documentos en la Universidad de las Ciencias Informáticas.* 2009.

Lucas, Salvador. 2009. Programación Funcional. [En línea] 2009. http://users.dsic.upv.es/asignaturas/facultad/prg/prf.html.

Luix. 2009. debug_mode=ON. [En línea] 2009. http://es.debugmodeon.com/articulo/scrum-una-metodologia-agil-ii..

Makarov, Alexander. 2009. SMASHING MAGAZINE. [En línea] 11 de Febrero de 2009. http://www.smashingmagazine.com/2009/02/11/the-big-php-ides-test-why-use-oneand-which-to-choose/.

Martínez, Rafael. 2001. *Manual de PHP. Grupo de documentación de PHP.* 2001.

Microsoft Corporation. 2010. MSDN. [En línea] 2010. http://msdn.microsoft.com/es-es/library/z165t2xk.

Moraga, Hernán. 2005. Energy Market. [En línea] 16 de Diciembre de 2005. http://www.e-market.cl/dir/umayor/ingsw/Apoyo/GRUPO_1_PROGRAMACION_AGIL.ppt.

Peña Cruz, Susel y Núñez Rizo, Anaisa. 2008. *Propuesta de una Línea de Producción de Software para el proyecto Nova Linux.* 2008.

Pérez Valdés, Damián. 2007. maestros del web. [En línea] 2007. http://www.maestrosdelweb.com/principiantes/los-diferentes-lenguajes-de-programacion-para-la-web/.

PymeCrunch. 2008. PymeCrunch. [En línea] 2 de Abril de 2008. http://pymecrunch.com/scrum-metodologia-agil-para-tus-proyectos.

Quintero, Antonia Mª Reina. 2000. *POA.* 2000.

Rodríguez Moreno, Iran Roberto y Garcia Calderón, José Alejandro. 2009. *Sistema Integrado de Transportación (SIT): Implementación de los Módulos Administración - Configuración y Seguridad.* 2009.

Rossel, Dr. Gerardo. 2007. Programación lógica. [En línea] 2007. http://www.amzi.com/articles/code07_whitepaper.pdf.

Sotomayor Basilio, Borja. 2010. The Department of Computer Science, The University of Chicago. [En línea] 2010. http://people.cs.uchicago.edu/~borja/pubs/revistaeside2002.pdf.

Sun. 2009. Sun. [En línea] 14 de Diciembre de 2009. http://es.sun.com/sunnews/press/2009/20091214.jsp.

Universidad de Chile, Departamento de Ciencias de la Computación. 2009. Universidad de Chile, Departamento de Ciencias de la Computación. [En línea] 2009. http://www.dcc.uchile.cl/~lmateu/Java/Transparencias/.java/all.htm.

Universidad de Huelva. 2005. Universidad de Huelva. [En línea] 2005. http://www.uhu.es/18214/temas/pd-tema5.pdf.

WHMCompleteSolution. 2010. HostingSolutions. [En línea] 2010. http://www.hostingsolutions.com.ve/compra/knowledgebase.php?action=displayarticle&id=14.

Zaballa Coca, Roilán. 2009. *"Personalización de distribuciones basadas en la familia SUSE Linux.".* 2009.

Zaninotto, François y Potencier, Fabien. 2009. librosweb.es. [En línea] 25 de Agosto de 2009. http://librosweb.es/symfony_1_2/capitulo1/symfony_en_pocas_palabras.html.

Glosario de términos

A

AJAX
Javascript asíncrono y XML, es un nuevo avance de la tecnología con el objetivo de incrementar el intercambio de datos con el servidor, 22.

B

Bit
Acrónimo de *Binary digit*. (dígito binario). Un *bit* es un dígito del sistema de numeración binario, 21.

Bottom-up
Estrategia de procesamiento de información características de las ciencias de la información, especialmente en lo relativo al *software*. Por extensión se aplican también a otras ciencias humanas y científicas, 8.

Buffer
Ubicación de la memoria en una computadora o en un instrumento digital reservada para el almacenamiento temporal de información digital, mientras que está esperando ser procesada, 20.

Bugs
Error o un defecto en el *software* o *hardware* que hace que un programa funcione incorrectamente, 16.

C

Caché
Conjunto de datos duplicados de otros originales, con la propiedad de que los datos originales son costosos de acceder, normalmente en tiempo, respecto a la copia en la caché, 20.

CVS
Aplicación informática que implementa un sistema de control de versiones mantiene el registro de todo el trabajo y los cambios en los ficheros que forman un proyecto y permite que distintos desarrolladores colaboren, 26.

D

Die
Muestra un mensaje y finaliza el *script* actual en el punto en el que se encuentra. No devuelve valor alguno, 51.

DOM
El *Document Object Model* es una interfaz de programación de aplicaciones que proporciona un conjunto estándar de objetos para representar documentos HTML y XML, un modelo estándar sobre cómo pueden combinarse dichos objetos, y una interfaz estándar para acceder a ellos y manipularlos, 16.

E

Echo
Muestra una o más cadenas de caracteres, 51.

F

Framework
Una estructura para elaboración de proyectos donde se parte de un esqueleto de apoyo utilizado como base para lo que se está construyendo, 2.

FTP
Protocolo de red para la transferencia de archivos entre sistemas conectados a una red TCP (*Transmission Control Protocol*), basado en la arquitectura cliente-servidor, 23.

G

GNU
Es un acrónimo recursivo que significa GNU No es Unix (*GNU is Not Unix*). En español, se recomienda pronunciarlo ñu como el antílope africano o fonéticamente, 27.

Grid
Es una matriz o tabla donde se muestran los datos, 37.

H

Hardware
Partes físicas y tangibles de una computadora, 14.

Helpers
Son una de las dos bibliotecas de código disponibles en CodeIgniter y la más sencilla de manejar en un principio, puesto que son funciones que están a disposición sin depender de ningún objeto, 22.

HTTP
HyperText Transfer Protocol (Protocolo de transferencia de hipertexto) es el método más común de intercambio de información en la *world wide web*, el método mediante el cual se transfieren las páginas *web* a un ordenador, 24.

I

Identación
Anglicismo (de la palabra inglesa *indentation*) de uso común en informática, que significa mover un bloque de texto hacia la derecha insertando espacios o tabuladores para separarlo del texto adyacente, lo que en el ámbito de la imprenta se ha denominado siempre como sangrado o sangría, 45.

IIS
Es una serie de servicios para los ordenadores que funcionan con Windows. Los servicios que ofrece son FTP, SMTP, NNTP y HTTP/HTTPS, 17.

Innodb

Es una tecnología de almacenamiento de datos de código abierto para la base de datos MySQL, incluido como formato de tabla estándar en todas las distribuciones de MySQL AB a partir de las versiones 4.0, 20.

M

Metodologías
Se refiere a los métodos de investigación en una ciencia. Se entiende como la parte del proceso de investigación que permite sistematizar los métodos y las técnicas necesarios para llevarla a cabo. Define quién debe hacer, qué, cuándo y cómo debe hacerlo, 2.

P

Paradigma
Representa un enfoque particular o filosofía para la construcción del software, 5.

Parchear
Un parche consta de cambios que se aplican a un programa, para corregir errores, agregarle funcionalidad y actualizarlo, 1.

Plug-in
Módulo de *hardware* o *software* que añade una característica o un servicio específico a un sistema más grande, 15.

S

Servlets
Los *servlets*, son objetos que corren dentro del contexto de un contenedor de *servlets* y extienden su funcionalidad. También podrían correr dentro de un servidor de aplicaciones, que además de contenedor para *servlet*, será contenedor para objetos más avanzados, como son los EJB (Tomcat sólo es un contenedor de *servlets*), 17.

Software
Todos los componentes intangibles de una computadora. Es el conjunto de programas necesarios para hacer posible la realización de una tarea específica, 2.

SVN
Subversion es un *software* de sistema de control de versiones, se le conoce como svn por ser ese el nombre de la herramienta utilizada en la línea de órdenes, 45.

T

Top-down
Estrategia de procesamiento de información características de las ciencias de la información, especialmente en lo relativo al *software*. Por extensión se aplican también a otras ciencias humanas y científicas, 6.

Trigger
Procedimiento que se ejecuta cuando se cumple una condición establecida al realizar una operación de inserción (INSERT), actualización (UPDATE) o borrado (DELETE), 21.

W

Web
Se utiliza para denominar uno de los servicios más importantes de la red Internet. Son páginas que utilizan lenguaje HTML, permitiendo presentar en la pantalla texto y gráficos en el formato deseado. Estas páginas contienen referencias o enlaces que permiten acceder a otras páginas., 4

Z

Zip
Formato de almacenamiento sin pérdida, muy utilizado para la compresión de datos como imágenes, programas o documentos, 24