Universidad de las Ciencias Informáticas

Facultad 8

***Título**: SWAP, Sistema Web para la verificación de autenticidad de los documentos en la Universidad de las Ciencias Informáticas.*

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autores:

Annia Louit Moreno.

Emilio Francisco Taillacq Trueba.

Tutor:

Ing. Yohandri Ril Gil

Co-tutor:

Ing. José Antonio Sánchez Imbert

Ciudad de La Habana, 23 de junio de 2009

"Año del 50 Aniversario de la Revolución"

DECLARACIÓN DE AUTORÍA

Declaro que soy el único autor de este trabajo y autorizo a la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmo la presente a los ___días del mes de _____del año _____.

Autores:

Annia Louit Moreno

_____

Emilio Francisco Taillacq Trueba

_____

Tutor:

Ing. Yohandri Ril Gil

_____

Co-tutor:

Ing. José Antonio Sánchez Imbert.

_____

Dedicatoria

A nuestro Padre, nuestro Amigo, nuestro Consolador.

A nuestros padres.

Agradecimientos comunes

Agradecemos a todos los que nos ayudaron y apoyaron el transcurso de estos meses de trabajo.

A Dios, por ser nuestro guía y nuestro apoyo, porque hasta aquí hemos llegado por Él. Al Padre por Su Amor, al Hijo por Su Gracia, al Espíritu Santo por ser nuestro mejor amigo.

A nuestro tutor, el profe Ril, por facilitarnos el tema de tesis y por sus oportunas sugerencias, por toda su ayuda. A nuestro tribunal por siempre corregirnos y ayudarnos en el desarrollo de toda la tesis.

A Lili y a tío José, por decir si cuando necesitamos su ayuda.

A Rosalba, Carlos Pupo, Adonys y Leandro, por su pronta ayuda en cuestiones de programación siempre que los necesitamos.

A Lidiana por ayudarnos a corregir los errores de ortografía y concordancia. Por leerse todo el documento de la tesis.

A nuestros padres, porque este trabajo es para ellos. A nuestros amigos, por estar siempre cuando los necesitamos. A nuestros hermanos, por sus oraciones y preocupación.

Agradecimientos

Emilio:

A mi Dios a quien agradezco de principio a fin cada paso de esta tesis.

A mi mamá y mi papá que siempre estuvieron apoyándome en cada momento.

A mis abuelos, tíos y primos que me siguieron en cada paso.

A mis amigos que sostuvieron mis manos cuando se me cansaron, comenzaron la carrera y llegaron al final, a la meta, conmigo.

A mis hermanos en la fe que me apoyaron con sus oraciones y sus palabras día tras día.

A mi pastor que se mantuvo intercediendo por mí.

A mi compañera de tesis Annia por soportarme todo un año haciendo este trabajo juntos.

A mis compañeros de grupo que desde primer año vienen conmigo.

A mis profesores que ayudaron a forjarme docentemente como ingeniero.

A todos aquellos, que son muchos, los cuales no daría abasto esta página para poner y que contribuyeron a formar mi vida y la realización de este trabajo.

Annia:

Gracias le doy a mi amigo el Espíritu Santo por ser el principal responsable de que esta tesis hoy esté terminada, por escucharme, consolarme, ayudarme, enseñarme y guiarme en todo.

A Dios es todo el mérito de este trabajo y toda la gloria es solo de Él. Gracias a Jesucristo, por ser mi Señor, mi Salvador, mi amigo, por ser el centro de mi vida y mi principal motivación, todo lo que hago es para agradarle a Él, y ese "todo" incluye, por supuesto, este trabajo. No tengo palabras para expresar mi agradecimiento a ti Señor.

Gracias a Dios por mis padres, que me han enseñado y guiado como han podido todos estos años de mi vida, y que se han sacrificado para que hoy yo pueda estar aquí.

A mi mamá, que la amo con el alma, es mi ejemplo a seguir como madre, pues se ha dado completa por sus hijas.

A mi papá, que lo amo con el alma, y sé que el deseo de su corazón es darme más.

A mi hermana Anais, que la amo con el alma, por aguantar mis pesadeces por casi 21 años, por estar siempre conmigo. Por velar por tanto por el desarrollo de mi tesis y llegar a ser más exigente que mi tutor y mi tribunal.

A mis abuelas, mis tíos y primos por su amor.

Agradecimientos

A Rosa, Orta, Indira y Julia, por acogerme desde el primer momento es su hogar, por su amor, comprensión, preocupación y ayuda incondicional.

A mis hermanos en la fe, por su apoyo, sus oraciones, por su amor, por quererme como soy, por compartir conmigo las preocupaciones de la tesis, por sus palabras de aliento. Porque Dios los puso ahí. Son unos de los regalos más lindos y preciados que me ha dado el Señor. Decir nombres no es necesario. Los amo con todo el corazón. Sigan brillando.

A Denise, por enseñarme tanto y tanto de tantas cosas. Por su amistad.

A mi compañero de tesis, Emilio, por ser instrumento para darme cuenta cuánta paciencia tengo y cuánta me falta aún. Gracias por escogerme como compañera de tesis.

A Yisel, Yadira y Lilianne (la china), por aguantarnos mutuamente desde la vocacional (y Yadira desde la secundaria). Gracias por no dejar morir la amistad.

A Katy, Zorilín, Rosalba, Grettell, Eduardo, Víctor Frank, Dago, Liana, Sandy, Enrique, Yusle, Ivette, por estar desde 1er año a mi lado o desde un poco después. Son muy especiales para mí, y aunque no entiendan dónde está la Annia que conocieron, les aseguro que esta los ama más.

A mis compañeros de grupo, a los viejos desde 1er año, a los que se quedaron en el camino, a los que se fueron incorporando y a los que vienen más atrás. Gracias por su compañía estos 5 años.

A mis profesores por contribuir a mi formación como profesional.

A Carlos y Ana por ser una gran bendición en mi vida.

Agradecimientos

A Adriana, Jesús, Kader, Tania y Lisset, por estar aunque no tan cerca (físicamente).

A Yurima por recordar nuestra amidad a pesar de estar 13 años sin vernos.

A Roxana y su familia, por ser tan buenos vecinos.

A todos los que han contribuido de una forma u otra a que pueda graduarme.

Muchas gracias.

Resumen

Los software antiplagio son utilizados en la mayoría de las universidades del mundo para combatir el creciente aumento de la copia ilegal de trabajos desde Internet por los estudiantes. Proveerse de información para realizar las tareas docentes no tiene nada de malo, lo negativo está en copiar textualmente documentos ajenos y hacerlos pasar por su propia autoría.

En la Universidad de las Ciencias Informáticas (UCI) se generan gran cantidad de documentos en las áreas de la docencia, la investigación y la producción, a los cuales no se les verifica la autenticidad, pues la universidad no cuenta con una estrategia para hacer dicha comprobación, ni con un software que compruebe si los diferentes documentos son copiados parcial o totalmente de otros documentos similares en la universidad o de sitios web en Internet.

El presente trabajo, para esta problemática, presenta una solución basada en un sistema Web que verifica si un documento tiene similitudes con otros, el cual podrá ser de gran utilidad para verificar la autenticidad de los documentos generados en la UCI; dicho sistema se nombra *SWAP*, siglas de Sistema Web Antiplagio, y será utilizado con este fin.

Para su desarrollo se utiliza la metodología de desarrollo ágil *Extreme Programming (XP)* por las facilidades que brinda para el trabajo en equipos de desarrollo pequeños y la producción de software con posibilidad de cambios en los requisitos en el futuro.

En este documento se hace un estudio del arte de algunas de las herramientas que a nivel mundial se utilizan con el mismo objetivo de la propuesta del sistema *SWAP*; además se describen las características de las herramientas utilizadas para realizar la aplicación y del ciclo de vida del software utilizando la metodología XP.

Índice

Introducción

A lo largo de toda la historia se registran algunos hechos delictivos que de una forma u otra llevan el apellido de “plagio”, la Real Academia de la Lengua Española define “plagiar” como “copiar en lo sustancial obras ajenas, dándolas como propias” [1]. Actualmente el plagio es un asunto importante, y hasta preocupante, en todos los países del mundo en todas las áreas, principalmente en la docencia; en universidades de los Estados Unidos de América, por ejemplo, le hacen firmar a los estudiantes un “contrato de honradez” como intento para evitar la copia ilegal, pero esto no es suficiente; varias conferencias y forum se han hecho también, con el mismo resultado ineficiente.

En Cuba esto no deja de ser un motivo de alarma en el área educacional. En la Universidad de las Ciencias Informáticas (UCI) coexisten una gran cantidad de estudiantes que reciben iguales materias docentes o que de una manera u otra interactúan con las mismas tareas, por lo que se generan diariamente múltiples documentos similares tales como: informes de asignaturas (seminarios, trabajos prácticos, talleres, tareas integradoras, trabajos investigativos), documentación de proyectos, tesis de grado, tesis de maestría, código fuente de programas, entre otros. Otro aspecto a tener en cuenta es que la gran mayoría de los estudiantes en la Universidad tienen acceso pleno a Internet, lo cual propicia que en muchas ocasiones los trabajos orientados como tareas investigativas extra clases en las diferentes disciplinas impartidas en la carrera sean simplemente copiados parcial o casi totalmente de la red de redes, aún cuando estos documentos consultados estén respaldados por leyes que prohíben su copia parcial o total, por lo que se incurre en el delito de plagio, no respetando la propiedad intelectual; además hay muchos sitios en la red que son poco fiables ya que son de libre acceso por lo que cualquier persona puede subir artículos a ellos, sin comprobarse la veracidad y autenticidad de los mismos, sitios tales como *http://www.monografias.com* o *http://www.rincondelvago.com* son algunos de los más visitados y utilizados en la realización de las tareas en nuestro centro. Más importante que detectar estudiantes copiando documentos es educarlos para que analicen las consecuencias negativas que trae consigo esta práctica ilegal en su capacidad de aprendizaje, pues retrasa y atrofia el desarrollo de habilidades tales como sintetizar, aprender a redactar, o interpretar textos, ya que realizan poco esfuerzo mental para realizar los trabajos orientados.

De acuerdo con el desarrollo tecnológico, y la evolución propia de las nuevas metodologías y herramientas en el proceso de enseñanza-aprendizaje, se hace necesario que la UCI se encuentre al tanto del desarrollo tecnológico, y proponga nuevos escenarios que le permitan utilizar las ventajas de herramientas antiplagio para comprobar la autenticidad de diferentes documentos generados en áreas de la docencia, la producción y la investigación.

Por tanto, el **problema científico** a resolver es: *¿Cómo determinar la autenticidad de los diferentes documentos científicos e investigativos generados en la Universidad de las Ciencias Informáticas para reducir el plagio existente en la actualidad en los trabajos docentes y productivos?*, planteando como **objetivo general** para darle solución a esta problemática, *desarrollar un sistema Web que permita la comparación de documentos generados en las áreas de la docencia y la producción para la detección de copias o reproducción ilegal en ellos.*

Como **objetivos específicos** se plantean**:**

1. *Realizar un estudio de los software antiplagio existentes.*
2. *Diseñar una aplicación informática que permita el análisis comparativo de documentos.*

La **idea a defender** de este trabajo es: *si se cuenta en la Universidad de las Ciencias Informáticas con un sistema que detecte el posible plagio en diferentes documentos, entonces se comprobará la autenticidad de los mismos y se elevará el nivel investigativo en las tareas docentes, productivas e investigativas.*

El **objeto de estudio** es el *proceso de validación de la autenticidad de documentos* y el **campo de acción** es la *autenticidad de los documentos de importancia generados en la docencia, la producción y la investigación en esta universidad.*

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

1. *Estudiar las herramientas antiplagio existentes.*
2. *Estudiar los algoritmos de segmentado y comparación de texto.*
3. *Analizar las principales plataformas de desarrollo Web.*
4. *Diseñar una base de datos para guardar los documentos.*
5. *Diseñar e implementar el sistema antiplagio.*

Capítulo 1. Fundamentación teórica.

1.1 Introducción.

En no pocas universidades del mundo se aplica para la revisión de los trabajos desarrollados por los estudiantes, software que detectan plagios en dichos documentos, *Turnitin, Eve2, Wcopyfind, Compilatio* o *My DropBox*, son ejemplos de aplicaciones que se usan con este fin en centros escolares de nivel superior de países tales como España, Alemania, Australia, Estados Unidos de América, Brasil, Chile, Francia y Reino Unido.

La Universidad de las Ciencias Informáticas carece de una herramienta que automatice la comparación de documentos en aras de validar la autenticidad de los trabajos que se generan en las diferentes asignaturas y en las actividades productivas e investigativas.

1.2 Software antiplagio en el mundo.

Existen varios software que internacionalmente se utilizan en la detección de plagio o copias en documentos, estos son mayormente utilizados en universidades debido a la gran cantidad de investigaciones que se generan en dichos centros educacionales.

a) ***AntiCutAndPaste:*** Herramienta desktop para detectar copia de código o fragmento de código en textos planos o códigos fuentes de programas tales como *C++, C#, Visual Basic, Delphi* y *Java*. Los algoritmos usados son muy rápidos y pueden manejar hasta tres millones de líneas de código de C en un minuto. A las modificaciones de menor importancia del código no se le hacen caso durante la búsqueda. Los informes son convenientemente ordenados por el tamaño total de todos los fragmentos similares y hay muchas opciones de personalización de los mismos. *AntiCutAndPaste* se integra fácilmente con *Microsoft Visual Studio* para que pueda saltar a la línea fuente haciendo un clic en una ventana de reporte con estilo de compilador. [2]

- Puede realizar hasta 50 mil comparaciones de líneas de código en un segundo. Pasando por alto las líneas convencionales.
- No tiene conexión a Internet.
- Requiere instalación en la computadora.
- Hay que predefinirle el origen del archivo así como el supuesto destino.

b) ***AntiPlagiarist:*** Herramienta desktop propietaria o privativa. [4]

- Puede comparar múltiples documentos o fragmentos de ellos.
- No compara código fuente, solo documentos de textos. Soporta los siguientes formatos: *HTML, DOC, TXT, WPD* y otros.
- Requiere instalación en la computadora.
- Hay que predefinirle el origen del archivo así como el supuesto destino.
- Sistemas operativos necesarios: *Win95/98/98SE/Me/2000/NT/XP/2003.*

c) ***Educared Antiplagio:*** Herramienta desktop para realizar búsquedas en Internet. Comprueba posibles plagios entre documentos introducidos en la propia aplicación, ofreciendo la posibilidad de organizarlos y archivarlos. Comprueba posibles plagios respecto a otras fuentes de documentos sea del tipo que sean en Internet. Otra utilidad de la aplicación, es poder comparar los apuntes desarrollados por un profesor, para una determinada asignatura, con los trabajos presentados por los alumnos de dicha asignatura. Además, los documentos que usted incorpore a la aplicación pasarán a formar parte de la base de datos de la herramienta para que de este modo no puedan volver a ser plagiados en otras circunstancias. [3]

- Presenta una búsqueda de *proxy* predefinida por ella, si no lo encuentra da error y no se puede inicializar el programa.
- Requiere instalación en la computadora.
- No se puede configurar manualmente el software ni el puerto.

d) ***Active File Compare:*** Herramienta desktop propietaria o privativa que permite comparar y sincronizar archivos de tipo texto así como buscar similitudes en código fuente de aplicaciones hechas en *C++, C#, Java, Visual Basic, Delphi, Object Pascal, SQL, Perl, Assembler, Fortran, Foxpro* y código *HTML, PHP, XML, INI* y archivos *BAT*. El algoritmo de comparación siempre localiza las diferencias correctamente tanto en archivos de texto pequeños como en archivos que contengan grandes cantidades de modificaciones. [5]

- Es utilizado por gran variedad de desarrolladores para verificar código en él.
- Permite sincronizar búsquedas automáticas.

- Presenta filtro de búsqueda.
- Es un software propietario.
- Requiere instalación en la computadora.

Estos son solo algunos de los software antiplagio más utilizados actualmente. Luego de estudiar sus características se concluye que de forma general todas las herramientas analizadas requieren instalación en la computadora y son privativas; además de que en su mayoría no pueden utilizarse en todos los sistemas operativos. No son sistemas centralizados sobre una base de datos, más bien son herramientas para comparar textos en un momento determinado. Se pueden usar los mecanismos de filtrado y comparación de texto utilizados en alguno de ellos.

1.3 Aplicación Web.

El software a desarrollar se implementará como una aplicación Web. ¿Qué es una aplicación Web?

Una aplicación Web es un conjunto de páginas Web estáticas y dinámicas. Estas contienen información específica de un tema en particular que es almacenada en algún sistema. Son basadas en el paradigma cliente-servidor, donde el servidor sabe cómo proporcionar un servicio y el cliente desea acceder a ese servicio. El cliente puede ser cualquier dispositivo de un navegador Web compatible. Este realiza peticiones al servidor y recibe la respuesta en un determinado formato, dibuja la página según el código recibido y ejecuta la respuesta a eventos. El servidor recibe las peticiones de los clientes, si le pide un recurso estático lo busca y lo devuelve y si le pide un recurso dinámico pasa la petición a un componente apropiado que generará el código y lo devolverá de vuelta al cliente.

La comunicación cliente-servidor se realiza por el protocolo *HTTP* (*HyperText Transfer Protocol*, en español, Protocolo de Transferencia de Hipertexto) que permite enviar peticiones (opcionalmente con parámetros) y recibir respuestas en un contenido específico. [6]

1.4 Fundamentación de las tecnologías en la que se basa la propuesta.

1.4.1 Lenguajes de programación.

PHP:

Es un lenguaje multiplataforma. Tiene capacidad de conexión con la mayoría de los manejadores de base de datos que se utilizan en la actualidad, hay que destacar su conectividad con *MySQL*. Además posee capacidad de expandir su potencial utilizando la enorme cantidad de módulos (llamados *ext's* o extensiones).

Todas las funciones del sistema están explicadas y ejemplificadas en un único archivo de ayuda. Es libre, por lo que se presenta como una alternativa de fácil acceso para todos. Permite las técnicas de Programación Orientada a Objetos. Tiene una biblioteca nativa de funciones sumamente amplia e incluida. No requiere definición de tipos de variables. Tiene manejo de excepciones.

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*), que permiten separar el tratamiento y acceso a los datos, la lógica de control y la interfaz de usuario en tres componentes independientes.

Java:

Java es un lenguaje orientado a objetos, de alto nivel, moderno, que permite la realización de programas profesionales. La tecnología Java ha cobrado gran importancia en el ámbito de Internet en la actualidad gracias a su plataforma *J2EE*. Está compuesta básicamente por dos elementos: el lenguaje Java y su máquina virtual (*Java Virtual Machine*).

Java proporciona una colección de clases para su uso en aplicaciones de red, que permiten establecer y aceptar conexiones con servidores o clientes remotos, lo que facilita la creación de aplicaciones distribuidas. El lenguaje Java fue diseñado para crear software altamente fiable para lo que proporciona numerosas comprobaciones en compilación y en tiempo de ejecución. Además está diseñado

para soportar aplicaciones que serán ejecutadas en variados entornos de red, desde *Unix* a *Windows NT*, pasando por *MAC* y estaciones de trabajo, sobre arquitecturas distintas y con sistemas operativos diversos.

La indiferencia a la arquitectura que tiene Java es sólo una parte de su portabilidad. Además, Java especifica los tamaños de sus tipos de datos básicos y el comportamiento de sus operadores aritméticos, por lo que los programas son iguales en todas las plataformas.

Java soporta sincronización de múltiples hilos de ejecución (*multithreading*) a nivel de lenguaje, muy útiles en la creación de aplicaciones de red distribuidas. Así, mientras un hilo se encarga de la comunicación, otro puede interactuar con el usuario mientras otro presenta una animación en pantalla y otro realiza cálculos.

El lenguaje Java y su sistema de ejecución en tiempo real son dinámicos en la fase de enlazado. Las clases sólo se enlazan a medida que son necesitadas. [11][12]

ASP.NET y Java:

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

Visual Basic .Net y *C#* son lenguajes orientados a objetos, al igual que Java, y en su diseño ha tenido mucha importancia la existencia de Internet. Desde la perspectiva de los desarrolladores, *J2EE* y *.Net* proporcionan las herramientas para crear Servicios Web. [14]

Ventajas de .Net frente a J2EE:

Una ventaja muy importante del entorno .Net frente a *J2EE* es la posibilidad de emplear múltiples lenguajes de programación, mientras que *J2EE* sólo trabaja con uno: Java. La alta diversidad de lenguajes es obligatoria por la misma variedad de las necesidades de los programadores. Un lenguaje moderno y orientado a objetos como Java puede resultar totalmente ineficaz (y hasta inadecuado) a la hora de abordar problemas que involucren cálculos matemáticos masivos y complejos, mientras que esos mismos cálculos pueden ser abordados mucho más adecuadamente con un lenguaje tan primitivo como

Fortran 77. Por otro lado, *.Net* posibilita así, que programadores de terceros lenguajes pasen a esta plataforma reduciendo el tiempo de aprendizaje y entrenamiento.

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

C# es un lenguaje interesante, fácil de aprender por los programadores de Java, el cual, en caso de estandarizarse podría resultar un lenguaje muy conveniente para ciertas tareas de programación en diferentes plataformas. *Microsoft* ha impulsado con gran energía los servicios Web y ha resaltado su importancia entre toda la comunidad de desarrolladores (utilicen o no los productos de esta compañía). La plataforma *.Net* se ha diseñado considerando los servicios Web (mientras que *J2EE* no) siendo estos servicios propios de la plataforma y ofrece una nueva versión de *ASP*, *ASP .Net*, que puede considerarse un entorno de programación "de verdad" en lugar de un entorno basado en scripts.

.Net fue construido para la integración a través de los servicios Web *XML* usando protocolos y formatos de ficheros como *SOAP* (Simple Object Access Protocol), WSDL (*Web Services Description Language*), y *UDDI* (*Universal Description, Discovery, and Integration*). Comparativamente, .Net va por delante con respecto a *J2EE* con respecto a servicios Web y estos servicios son propios de la plataforma, aunque *J2EE* respondió ya con el lanzamiento del *Java Web Services Developer Pack*. De todos modos, la facilidad, rapidez y sencillez con la que se pueden construir servicios Web con el Asistente de servicios Web de *Visual Studio .Net* son muy superiores a las de las herramientas para construir servicios Web dentro del entorno de *J2EE*. [14]

Ventajas de J2EE frente a .Net:

Las implementaciones de *J2EE* pueden adquirirse en distintas compañías, mientras que .Net solo puede comprarse en *Microsoft*. El hecho de que haya distintas organizaciones implementando *J2EE* ofrece mayor variedad para los usuarios finales y permite la existencia de una cierta competencia entre ellas para obtener mejores productos, lo cual no existe en el caso de *Microsoft* y su *.Net*.

Debido al proceso evolutivo de los productos de *Microsoft*, y en muchos casos, por motivos de compatibilidad, la seguridad frente a virus informáticos de los productos de *Microsoft* es menor que los basados en Java, pues desde un comienzo Java se fundamentó en un estricto modelo de seguridad.

Las aplicaciones Java pueden correr en una amplia gama de sistemas operativos (desde sistemas empresariales como *Windows 2000, OS/390, Solaris, HP-UX, IRIX* u otras versiones de *Unix* hasta en sistemas orientados más a ordenadores personales como *Mac OS*, *Windows 9x* o *Linux*, y en sistemas operativos para dispositivos móviles) y de arquitecturas *hardware*. Hasta la fecha, .*Net* corre solamente sobre sistemas operativos de *Microsoft*, siendo *J2EE* el único entorno de desarrollo que ofrece una independencia real de la plataforma.

La tecnología Java es una tecnología abierta y se basa en gran parte en estándares de organizaciones de normalización y estándares empresariales "de facto". Esto posibilita que los desarrolladores puedan conocer y entender completamente cómo hace las cosas Java y aprovecharlo para sus aplicaciones y, por otro lado, al basarse en estándares empresariales, simplifica la integración con productos de múltiples compañías. En contraposición, solo el código fuente del nuevo lenguaje *C#* de la plataforma .*Net* ha sido abierto al público general.

Aunque Java fue creado originalmente por una compañía: *Sun MicroSystems*, lo cierto es que *J2EE* es ahora el producto de la colaboración de más de 400 empresas y organizaciones de todo el mundo. La plataforma .*Net* es el producto de una sola compañía, que aunque haya implementado algunos estándares en .*Net* y esté intentando conseguir que ciertas tecnologías se conviertan en estándares "oficiales", no puede tener la misma aprobación que *J2EE*.

J2EE ha probado su eficacia en muchos entornos y situaciones empresariales distintas, mientras que .*Net* ha visto oficialmente la luz hace poco.

Concluyendo, los productos de *Microsoft* han sido los mejores en ambiente homogéneos, de redes “todos *Windows*”, mientras que la plataforma empresarial Java se desempeña bastante bien en ambientes heterogéneos. Si nosotros consideramos una red enteramente *Microsoft*, esto permite la integración del sistema y la utilización de las características de seguridad de .*Net* a su potencial más alto. En caso de un ambiente mixto, las características de seguridad “independiente de la plataforma” de Java pueden ser más útiles que aquellas de .*Net*. [14]

Se escogió Java luego de estudiar sus características de acuerdo a lo anterior expuesto y a estos aspectos mencionados a continuación:

a) Uso del servidor: Carga el *Servlet* una sola vez, así como la página que fue pedida y lo mantiene en memoria ayudando así a que el servidor esté menos sobrecargado. Es en gran medida orientado al servidor.
b) Respaldo de compañías: Es implementado y optimizado por *Sun, IBM* y *Microsoft.*
c) Facilidad para programar: Altamente complejo, se necesita más de un miembro de equipo para realizar esta labor.
d) Costo del Producto: Gratis.
e) Reusabilidad: Es una programación totalmente orientada a objetos, lo que nos permite crear y usar una enorme cantidad de librerías y otros recursos. Lo cual ayuda a que el código usado sea altamente reusable.
f) Arquitectura: Arquitectura 3 capas claramente definidas (base datos (*hibernate*), negocio (*spring*), interfaz (*jsf -jsp*)).
g) Portabilidad: Multiplataforma.
h) Documentación: No posee mucha información al respecto, la ayuda se encuentra en su mayoría en inglés, además no existen muchos ejemplos de funcionalidades. Su documentación está muy distribuida o muy explícita.

1.5 Fundamentación de la metodología a utilizar.

Hasta hace poco tiempo el proceso de desarrollo llevaba asociado un marcado énfasis en el control del proceso mediante una rigurosa definición de roles, actividades y artefactos, incluyendo modelado y documentación detallada. Este esquema "tradicional" para abordar el desarrollo de software ha demostrado ser efectivo y necesario en proyectos de gran tamaño (respecto a tiempo y recursos), donde por lo general se exige un alto grado de ceremonia en el proceso. Sin embargo, este enfoque no resulta ser el más adecuado para muchos de los proyectos actuales donde el entorno del sistema es muy cambiante, y en donde se exige reducir drásticamente los tiempos de desarrollo pero manteniendo una alta calidad. Ante las dificultades para utilizar metodologías tradicionales con estas restricciones de tiempo y flexibilidad, muchos equipos de desarrollo se resignan a prescindir del “buen hacer” de la ingeniería del software, asumiendo el riesgo que ello conlleva. En este escenario, las metodologías ágiles emergen como una posible respuesta para llenar ese vacío metodológico. Por estar especialmente orientadas para proyectos pequeños, las metodologías ágiles constituyen una solución a medida para ese entorno,

aportando una elevada simplificación que a pesar de ello no renuncia a las prácticas esenciales para asegurar la calidad del producto.

Las características de los proyectos para los cuales las metodologías ágiles han sido especialmente pensadas se ajustan a un amplio rango de proyectos industriales de desarrollo de software; aquellos en los cuales los equipos de desarrollo son pequeños, con plazos reducidos, requisitos volátiles, y/o basados en nuevas tecnologías.

Proceso Racional Unificado (RUP, en inglés *Rational Unified Process*)

Es un proceso para el desarrollo de un proyecto de un 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 que orientan el proyecto a la importancia para el usuario y lo que este quiere, 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 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.

Como filosofía, RUP maneja 6 principios claves:

- Adaptación del proceso

El proceso deberá adaptarse a las características propias de la organización. El tamaño del mismo, así como las regulaciones que lo condicionen, influirán en su diseño específico. También se deberá tener en cuenta el alcance del proyecto.

- Balancear prioridades

Los requerimientos de los diversos inversores pueden ser diferentes, contradictorios o incluso disputarse recursos limitados. Debe encontrarse un balance que satisfaga los deseos de todos.

- Colaboración entre equipos

El desarrollo de software no lo hace una única persona sino múltiples equipos. Debe haber una comunicación fluida para coordinar requerimientos, desarrollo, evaluaciones, planes, resultados.

- Demostrar valor iterativamente

Los proyectos se entregan, aunque sea de un modo interno, en etapas iteradas. En cada iteración se analiza la opinión de los inversores, la estabilidad y calidad del producto, y se refina la dirección del proyecto así como también los riesgos involucrados.

- Elevar el nivel de abstracción

Este principio dominante motiva el uso de conceptos reutilizables tales como patrón del software, lenguajes *4GL* o esquemas (*frameworks*) por nombrar algunos. Estos se pueden acompañar por las representaciones visuales de la arquitectura, por ejemplo con *UML*.

- Enfocarse en la calidad

El control de calidad no debe realizarse al final de cada iteración, sino en todos los aspectos de la producción.

El ciclo de vida de RUP

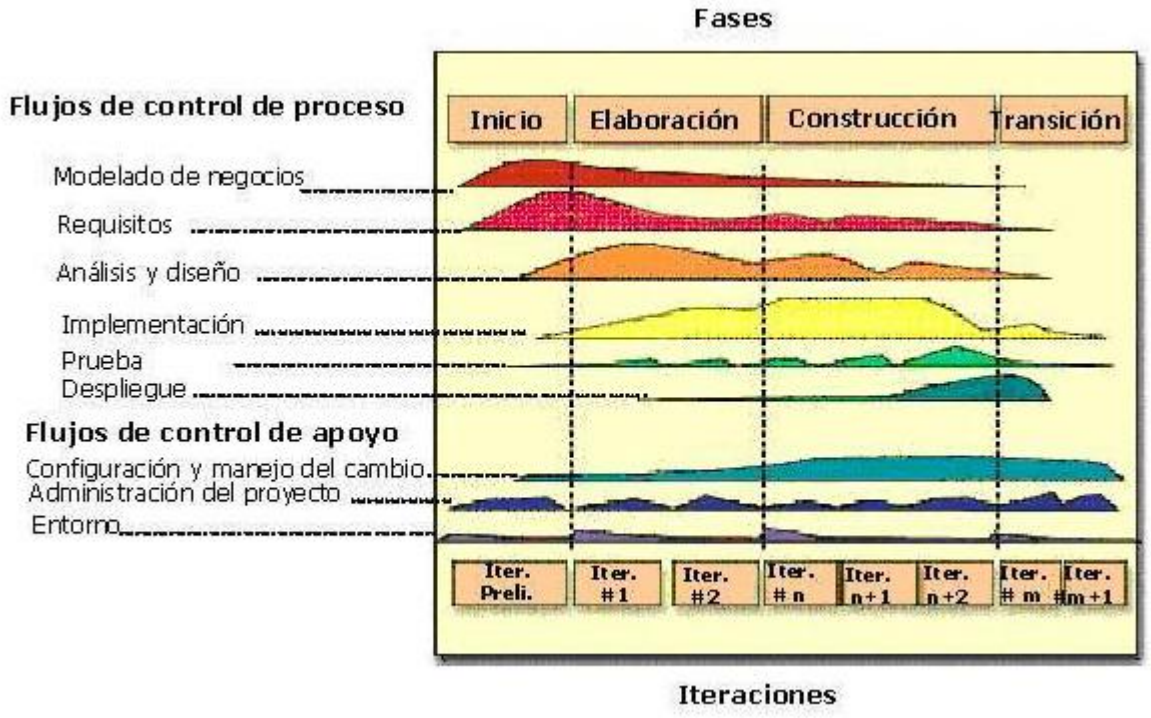

Fig. 1. Flujos y fases de RUP

RUP divide el proceso en cuatro fases, dentro de las cuales se realizan varias iteraciones en número variable según el proyecto y en las que se hace un mayor o menor hincapié en las distintas actividades. En las iteraciones de cada fase se hacen diferentes esfuerzos en cada una de las actividades.

- Inicio

Se hace un plan de fases, se identifican los principales casos de uso y se identifican los riesgos. Se define el alcance del proyecto.

- Elaboración

Se hace un plan de proyecto, se completan los casos de uso y se eliminan los riesgos

- Construcción

Se concentra en la elaboración de un producto totalmente operativo y eficiente, y el manual de usuario.

- Transición

Se instala el producto en el cliente y se entrena a los usuarios. Como consecuencia de esto suelen surgir nuevos requisitos a ser analizados. [13]

A continuación se describen brevemente las fases del RUP:

Fase de inicio

Durante la fase de inicio las iteraciones hacen mayor énfasis en actividades de modelado del negocio y de requisitos.

- Modelado del negocio

En esta fase el equipo se debe familiarizar más con el funcionamiento de la empresa, para conocer sus procesos. Entender la estructura y la dinámica de la organización para la cual el sistema va ser desarrollado. Entender el problema actual en la organización objetivo e identificar potenciales mejoras. Asegurar que clientes, usuarios finales y desarrolladores tengan un entendimiento común de la organización objetivo.

- Requisitos

En esta línea los requisitos son el contrato que se debe cumplir, de modo que los usuarios finales tienen que comprender y aceptar los requisitos que especifiquemos. Establecer y mantener un acuerdo entre clientes y otros *stakeholders* sobre lo que el sistema podría hacer. Proveer a los

desarrolladores un mejor entendimiento de los requisitos del sistema. Definir el ámbito del sistema. Proveer una base para estimar costos y tiempo de desarrollo del sistema. Definir una interfaz de usuario para el sistema, enfocada a las necesidades y metas del usuario.

Fase de elaboración

En la fase de elaboración, las iteraciones se orientan al desarrollo de la línea base de la arquitectura, abarcan más los flujos de trabajo de requerimientos, modelo de negocios (refinamiento), análisis, diseño y una parte de implementación orientado a la línea base de la arquitectura.

- Análisis y Diseño

En esta actividad se especifican los requerimientos y se describen sobre cómo se van a implementar en el sistema. Se transforman los requisitos al diseño del sistema. Se desarrolla una arquitectura para el sistema. Se adapta el diseño para que sea consistente con el entorno de implementación.

Fase de construcción:

- Implementación

Se implementan las clases y objetos en ficheros fuente, binarios, ejecutables y demás. El resultado final es un sistema ejecutable. Se planifican qué subsistemas deben ser implementados y en qué orden deben ser integrados, formando el Plan de Integración. Cada implementador decide en qué orden implementa los elementos del subsistema. Si encuentra errores de diseño, los notifica. Se integra el sistema siguiendo el plan.

- Pruebas

Este flujo de trabajo es el encargado de evaluar la calidad del producto que estamos desarrollando, pero no para aceptar o rechazar el producto al final del proceso de desarrollo, sino que debe ir integrado en todo el ciclo de vida. Se encuentran y documentan defectos en la calidad del software. Generalmente asesora sobre la calidad del software percibida. Provee la validación de los supuestos errores realizados en el diseño y especificación de requisitos por medio de demostraciones concretas. Se verifican las funciones del producto de software según lo diseñado y que los requisitos tengan su apropiada implementación.

- Despliegue

Esta actividad tiene como objetivo producir con éxito distribuciones del producto y distribuirlo a los usuarios. Las actividades implicadas incluyen: probar el producto en su entorno de ejecución final, empaquetar el software para su distribución, distribuir el software, instalar el software, proveer asistencia y ayuda a los usuarios, formar a los usuarios y al cuerpo de ventas, y migrar el software existente o convertir bases de datos.

Extreme Programming (XP)

"Un proceso ligero, de bajo riesgo, flexible, predecible, científico y divertido de desarrollar software." [21]

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, adaptación a los cambios, ya sean de tecnología o de metáforas dentro del negocio.

- Comunicación

Al desarrollar sistemas software con una u otra metodología, ya sea ágil o tradicional se determina que la comunicación es pieza importante para un buen desarrollo de los proyectos, intercambio de ideas entre jefes, personal que manejara el sistema, terceras personas que aportan, equipo de desarrollo y equipo de trabajo.

La mala comunicación se puede representar en ocasiones con el cliente, (requisitos mal comprendidos), procesos mal definidos o inexactos. También cuando el equipo de trabajo, "desarrolladores", comete errores en el manejo de la metáfora, en el manejo a nivel de código cuando no se comentan cambios entre los desarrolladores, partes inseguras o no se enseñan u orientan apuntes importantes sobre programación.

En la práctica se observa que la comunicación basada en metodologías ágiles potencia el desarrollo de los proyectos software, gracias a que el cliente casi siempre está en el sitio de trabajo, se genera una retroalimentación apoyada con reuniones diarias.

Al estar diariamente con el equipo cliente se aprende mucho sobre el manejo de la empresa, procesos, temas relacionados y se realizan conexiones o relaciones laborales importantes para ambas partes.

- Sencillez

Cuando se desarrolla con interfaces y lógica simple, los sistemas mejoran la portabilidad, manejo, mantenimiento, capacitación y rapidez, siendo un punto a favor para la adquisición o desarrollo de dicho sistema.

Cuando un programa tiene mucha sobrecarga o abuso de detalles se puede observar que estos generan retrasos en máquina, también no son usados del todo o interfieren en el manejo del programa, es recomendado ubicar en los proyectos la esencia y solo lo necesario para un óptimo manejo.

- Retroalimentación

Aumenta mucho la socialización y comunicación entre los distintos equipos relacionados en el proyecto, es recomendado manejar una retroalimentación constante y oportuna en todas las fases del proyecto de forma rápida y clara.

El cliente es el usuario final encargado de administrar, manejar y obtener el máximo provecho del sistema desarrollado, por lo tanto es recomendable que los aportes de este usuario tengan eco y sean primordiales en las fases: pruebas, entregas frecuentes, versionado e integración, para obtener como resultado un producto final óptimo y aprobado por el cliente.

- Valentía

Ser valientes en el desarrollo de proyectos software es asumir riesgos, enfrentarse a ellos y continuar al límite junto a esta actitud hasta finiquitar un sistema, un valor importante y arriesgado que gracias a la pertenencia que se maneja en XP permite la toma de decisiones importantes para el buen desarrollo de los proyectos software.

En la práctica se aplicaron decisiones de valentía al cambiar la base de datos estando esta en un 75% conformada y la interfaz de usuario para que su manejo fuera más ágil, obteniendo excelentes resultados.

Los grupos de trabajo de desarrolladores que se conforman solos, organizan autónomamente o auto conforman, funcionan en el desarrollo de proyectos con mejores resultados y obtienen mejores logros, ya que se conocen, forman un ambiente o entorno acorde para trabajo e interacciones con el

cliente. Los equipos que son conformados sistemáticamente enfrentan problemas desde el inicio, desarrollo y fin de un proyecto, ya que desconocen el manejo de trabajo en el grupo nuevo mientras se acoplan al equipo.

En la Programación Extrema se propone que además de las reuniones diarias los desarrollos de código sean efectuados por parejas de programadores en lapsos de tiempo de 40 horas semanales y entre estas prácticas un equipo no conocido podría no cumplir la ley de sinergia en la que todos los componentes del sistema tienen o trabajan con un mismo fin. Un equipo conocido trabaja con más ánimo, confianza y de forma agradable para el desarrollo de distintos proyectos. [8]

Comparación entre las metodologías ágiles y las metodologías tradicionales

Las metodologías ágiles son basadas en heurísticas provenientes de prácticas de producción de código, especialmente preparados para cambios durante el proyecto, realizan un proceso menos controlado, con pocos principios, no existe contrato tradicional o al menos es bastante flexible, el cliente es parte del equipo de desarrollo, grupos pequeños (menos de 10 integrantes) y trabajando en el mismo sitio, se generan pocos artefactos, hay pocos roles y se hace menos énfasis en la arquitectura del software. Mientras que las metodologías tradicionales son basadas en normas provenientes de estándares seguidos por el entorno de desarrollo, tienen cierta resistencia a los cambios, realizan procesos mucho más controlados con numerosas políticas y/o normas, existe un contrato prefijado, el cliente interactúa con el equipo de desarrollo mediante reuniones, los grupos de trabajo son grandes y posiblemente distribuidos, se generan más artefactos, hay más roles, la arquitectura del software es esencial y se expresa mediante modelos.

Aunque los creadores e impulsores de las metodologías ágiles más populares han suscrito el manifiesto ágil y coinciden con los principios enunciados anteriormente, cada metodología tiene características propias y hace hincapié en algunos aspectos más específicos. [7]

Por los motivos antes mencionados se escoge desarrollar el sistema *SWAP* con la metodología XP.

1.5 Herramientas a utilizar.

1.5.1 Sistemas Gestores de Bases de Datos.

Microsoft SQL Server:

Es un Sistema de Gestión de Bases de Datos Relacionales. Es capaz de poner a disposición de muchos usuarios grandes cantidades de datos de manera simultánea.

Entre sus ventajas se encuentra el soporte de transacciones, tiene gran escalabilidad, estabilidad, seguridad, también soporta procedimientos almacenados. Permite trabajar en modo cliente-servidor, la información está en el servidor y solo es accedida en modo lectura por los clientes. Administra información de más servidores de datos. Entre sus desventajas encontramos sus licencias con altos precios y su portabilidad (solo en *Windows*).

ADABAS (*Adaptable Database System*):

Base de Datos jerárquica de alto rendimiento, de lista invertida, no es relacional, se usa en aplicaciones con altos volúmenes de procesamiento de datos o de transacciones en línea. Entre sus ventajas se encuentran que los archivos (no contiene tablas) es la mayor unidad organizacional, los registros (no contiene filas) es la unidad contenedora dentro de la unidad organizacional, los campos (por las columnas) son los componentes de una unidad de contenido. Es muy efectivo a la hora de mantener la integridad de los datos. Entre sus desventajas está que no tiene un motor de SQL embebido, por lo que un mecanismo externo de consulta debe ser provisto, la lectura sucia es el modo estándar de operación, soporta tablas embebidas, por ejemplo en el caso de los archivos anidados.

Oracle:

Es un Sistema de Gestión de Bases de Datos Relacionales, es básicamente una herramienta cliente-servidor. Dentro de sus ventajas está el soporte de transacciones, la estabilidad, la escalabilidad, es multiplataforma, soporta diferentes cantidades de datos. Entre sus desventajas encontramos el alto precio de sus licencias, así como de soporte técnico, y que genera una gran cantidad de procesos en el procesador.

MySQL:

Sistema de Gestión de Bases de Datos Relacionales, licenciado. Tiene un diseño multihilo lo que le permite soportar una gran carga de forma muy eficiente. Presenta gran rapidez así como facilidad de uso, es de fácil configuración así como de instalación y la velocidad operacional es baja, consume poco de memoria, es poco probable que corrompa los datos. Entre las desventajas están que carece de soporte para transacciones, *rollback* y subconsultas, no maneja la integridad referencial, no es viable para su uso con grandes bases de datos, no implementa una buena escalabilidad.

PostgreSQL:

Sistema de Gestión de Bases de Datos orientadas a objetos, de software libre, por lo que no es manejado por una sola compañía, sino que es dirigido por una comunidad de desarrolladores y organizaciones comerciales que se esfuerzan en su desarrollo; esta comunidad se llama el PGDG (*PostgreSQL Global Development Group*). Es el sistema de gestión de base de datos más avanzado del mundo.

PostgreSQL es un sistema avanzado de administración de bases de datos objeto-relacionales (*ORDBMS*) derivado del sistema de administración de bases de datos Postgres de *Berkeley*. Es un sistema objeto-relacional (incluye herencia, tipos de datos, funciones, restricciones, disparadores, reglas e integridad transaccional).

Es el servidor de bases de datos más utilizado por los programadores de *servlets* de Java y, en general, por todos aquellos que realizan aplicaciones cliente-servidor complejas o críticas en el mundo *Linux/Unix*.

Entre sus principales ventajas está su alta concurrencia, su amplia variedad de tipos nativos, costo, estabilidad, confiabilidad, gran escalabilidad, funcionalidad (motor de bases de datos avanzado). Y sus desventajas radican en que consume bastantes recursos y carga con mucha facilidad el sistema y su velocidad de respuesta es un poco deficiente al gestionar bases de datos relativamente pequeñas, aunque esta misma velocidad la mantiene al gestionar bases de datos realmente grandes. [9]

Por las características descritas anteriormente del PostgreSQL se escoge el mismo como gestor de base de datos del sistema SWAP.

1.5.2 Plataforma de desarrollo

Red Hat Developer Studio:

Red Hat Developer Studio es un conjunto, basado en *Eclipse*, de herramientas de desarrollo pre-configuradas para plataformas *JBoss* y *Red Hat Enterprise* [10]. Está diseñado para maximizar la productividad del desarrollador sobre las soluciones de *Red Hat* y tiene arquitectura *open source*.

Developer Studio combina productos aportados a *Red Hat* por *Exadel: Exadel Studio Pro, RichFaces* y *Ajax4jsf,* con software de *middleware Jboss,* como *JBoss Seam* e *Hibernate,* dentro de un potente entorno de desarrollo para aplicaciones *SOA*, *Ajax* y Java empresariales. Aumenta y proporciona nuevas herramientas alrededor de *JBoss Seam* para construir aplicaciones de manera sencilla y consistente. Ofrece un modelo unificado y sencillo para desarrollar cualquier clase de aplicación, eliminando la necesidad de múltiples modelos de programación. También hace más sencilla la construcción de aplicaciones *Java EE*, con capacidades como *What You See Is What You Get (WYSIWYG)* y edición de *JavaServer Faces (JSF)* y páginas *Facelets*, asistencia de código dinámico y una paleta de componentes. Además, al incluir e integrar *JBoss Application Server*, *Developer Studio* simplifica el despliegue, la ejecución y la depuración de las aplicaciones *Java EE*. Es el primer entorno de desarrollo *open source* basado en Eclipse, que une el *runtime* con las herramientas. Esto elimina la necesidad de que el desarrollador improvise estructuras y componentes *open source* antes de que empiece a desarrollar. [25]

Tomcat:

Es el servidor Web con soporte de servlets y JSF y de aplicaciones del proyecto Yakarta, decimos que es servidor Web ya que gestiona solicitudes y respuestas HTTP (incluye el servidor Apache) gracias a sus conectores HTTP; además es servidor de aplicaciones o contenedor de Servlets/JSP (Catalina).

El motor de servlets de Tomcat a menudo se presenta en combinación con el servidor Web Apache. Puede funcionar como servidor Web por sí mismo. Es usado como servidor Web autónomo en entornos con alto nivel de tráfico y alta disponibilidad. Fue desarrollado sobre Java, o sea está completamente integrado a este lenguaje, ideal para usarlo cuando se programa en Java. [15]

1.6 Propuesta de solución.

Se tiene como propuesta de solución el desarrollo de un sistema Web implementado en el lenguaje Java por las ventajas y características expuestas anteriormente. Como ambiente de desarrollo integrado (*IDE*) se propone *Red Hat Developer Studio*. Como gestor de base de datos se propone el PostgreSQL por sus características y su integración a Java y el Tomcat como servidor Web.

1.7 Conclusiones.

En este capítulo se mencionaron algunos de los software antiplagio que se utilizan en muchas universidades del mundo. Además se describieron de manera sencilla las tecnologías, la metodología y las herramientas a utilizar en el desarrollo del sistema. Por último se elaboró la propuesta de solución para un mayor entendimiento en sentido general.

Capítulo 2. Descripción del sistema.

2.1 Introducción.

En este capítulo se describe de forma general el sistema a desarrollar, así como los requerimientos funcionales y no funcionales del mismo, las personas que se relacionan con el sistema y cómo interactúan con la aplicación, y se hace una descripción de las características de la arquitectura en capas, la cual se utilizará en el software.

2.2 Descripción del sistema.

El sistema que se desea desarrollar dará cumplimiento a las necesidades de reducir el plagio en los documentos docentes, investigativos y productivos que se generan en la Universidad de las Ciencias Informáticas.

Actualmente en la universidad no se utiliza ningún software antiplagio, debido entre otras cuestiones, a que casi todos los que están disponibles en la red son software propietarios o se necesita acceso a Internet para poder utilizarlos. Teniendo en cuenta estas razones y lo que se plantea en la situación problémica, se desea darle solución a este problema implementando un software antiplagio propio de nuestra universidad que pueda ser utilizado por todos los trabajadores docentes y que ayude a encontrar similitudes entre documentos para exigir más a los estudiantes en sus trabajos y tareas docentes.

No se describen los procesos elementales del negocio pues actualmente no se realiza detección de plagio en la universidad en ninguna área lo que se pudo corroborar mediante encuestas realizadas en los diferentes Departamentos Docentes y en la Dirección de Investigaciones. El software a desarrollar deberá dar solución al problema en cuestión, permitiendo que el usuario realice la solicitud al sistema para buscar plagio en un documento cualquiera y que el sistema realice el análisis devolviendo un reporte con los resultados.

2.3 Personas que interactúan con el sistema.

Las personas que interactúan con este sistema son aquellas que obtienen un resultado del mismo. Teniendo en cuenta las características del sistema podrán acceder a él cuatro tipos diferentes de usuarios, todos deben autentificarse con el usuario y la contraseña del dominio UCI.

1. **Estudiantes:** Pueden acceder a la aplicación y verificar la autenticidad de un solo documento. No pueden ver las estadísticas de plagio.
2. **Profesores:** Pueden acceder a la aplicación y verificar la autenticidad de varios documentos. No pueden ver las estadísticas de plagio.
3. **Profesores con privilegios:** Estos son los jefes de departamentos docentes o profesores autorizados, cuya autorización se le dará en la base de datos del sistema. Pueden acceder a la aplicación y verificar la autenticidad de varios documentos. Pueden ver las estadísticas de plagio.
4. **Administrador:** Se encarga de administrar el sistema, arreglar posibles fallos y gestionar que funcionen bien todas las conexiones con las bases de datos. Pueden acceder a la aplicación y verificar la autenticidad de varios documentos. Pueden ver las estadísticas de plagio. Tiene todos los privilegios.

2.4 Requerimientos del sistema.

Los requisitos funcionales son capacidades o condiciones que el sistema debe cumplir y los requisitos no funcionales son propiedades o cualidades que el producto debe tener, son las características que hacen al producto atractivo, usable, rápido y/o confiable.

De acuerdo a estas definiciones de requerimientos, se mencionan a continuación los requisitos que debe cumplir el sistema *SWAP*.

Requisitos funcionales:

R1. Autenticar usuario.

R1.1 Permitir entrar los datos requeridos para la autenticación (usuario y contraseña UCI).

R1.2 Validar los datos introducidos por el usuario.

R2. Subir documento al sistema.

R2.1 Permitir entrar datos del documento y guardarlos.

R2.2 Permitir cargar un documento en la aplicación.

R3. Comparar documentos.

R3.1 Permitir leer el texto del documento.

R3.2 Permitir comparar texto entre los documentos.

R4. Guardar documentos en la base de datos del sistema.

R4.1 Permitir guardar información del resultado de la comparación entre los documentos.

R4.2 Permitir guardar documento comparado.

R5. Generar reportes.

R5.1 Mostrar resultado de la comparación entre los documentos.

R5.2 Mostrar datos de los documentos entre los cuales hay similitud.

R6. Ver estadísticas de plagio por criterios.

R6.1 Mostrar los diferentes criterios para las estadísticas.

R6.2 Mostrar estadísticas según el/los criterios seleccionados.

Requisitos no funcionales:

- Requerimientos de Software:

  Multiplataforma, Internet Explorer versión 5.0 o mayor, Mozilla Firefox.

- Requerimientos de Hardware:

  Módem estándar o tarjeta de red. Memoria RAM de 256 Mb o más.

- Restricciones en el diseño y la implementación:

  - Estándares requeridos:

    Se utilizarán los estándares expuestos en *http://java.sun.com/docs/codeconv/CodeConventions.pdf*, como parte del estándar universal de java.sun.

- Lenguajes de programación a ser usados para la implementación:

  Lenguaje Java.

- Bibliotecas de clases:

  Java.io.FileInputStream; olrg.springframework.context.ApplicationContext; org.springframework.context.support.ClassPathXmlApplicationContext; java.util.Iterator; java.util.List; org.springframework.beans.factory.config.PropertyPlaceholderConfigurer; org.apache.commons.dbcp.BasicDataSource; org.springframework.orm.hibernate3.LocalSessionFactoryBean; org.springframework.orm.hibernate3.HibernateTransactionManager; org.springframework.orm.hibernate3.HibernateTemplate; com.etymon.; com.etymon.pj.Pdf; com.etymon.pj.object.*; etymon.*; etymon.pjx.PdfInput; etymon.pjx.PdfInputBuffer; etymon.pjx.PdfManager; etymon.pjx.PdfReader.*

- Requerimientos de Seguridad:

El sistema usará el Framework Acegi para validar aspectos de seguridad como son la confiabilidad e integridad de la información, se usará una base de datos caché en un servidor principal con conexión a la base de datos principal con el objetivo de asegurar los activos fijos, que en este caso serían los documentos.

- Requerimientos de Usabilidad:

El sistema debe permitir a usuarios con un nivel mínimo básico en conocimientos de informática, acceder a él y gestionar documentos. Este debe estar disponible para que las opciones principales de gestión de documentos estén visibles y accesibles.

2.5 Arquitectura candidata

Las técnicas metodológicas desarrolladas con el fin de facilitar la programación se engloban dentro de la llamada Arquitectura de Software o Arquitectura Lógica. Se refiere a un grupo de abstracciones y patrones que nos brindan un esquema de referencia útil para guiarnos en el desarrollo de software dentro de un sistema informático.

Así, los programadores, diseñadores, ingenieros y analistas pueden trabajar bajo una línea común que les posibilite la compatibilidad necesaria para lograr el objetivo deseado.

Algunos objetivos dentro de un esquema de Arquitectura de Software pueden ser: el software debe ser mantenible, esto es, fácilmente analizable, modificable, corregible; también puede ser un objetivo el nivel de interacción con otros sistemas informáticos, o su escalabilidad.

Estas arquitecturas están definidas muchas veces por el tipo de tecnología a la cual se enfrenta un programador o grupo de programadores, por lo cual algunos tipos de arquitectura son más recomendables que otras para ciertas tecnologías. [18]

Para el desarrollo de esta aplicación se escogió la arquitectura en tres capas que se describe a continuación.

Arquitectura en tres capas

La estrategia tradicional de utilizar aplicaciones compactas causa gran cantidad de problemas de integración en sistemas software complejos como pueden ser los sistemas de gestión de una empresa o los sistemas de información integrados consistentes en más de una aplicación. Estas aplicaciones suelen encontrarse con importantes problemas de escalabilidad, disponibilidad, seguridad, integración.

Para solventar estos problemas se ha generalizado la división de las aplicaciones en capas que normalmente serán tres: una capa que servirá para guardar los datos (base de datos), una capa para centralizar la lógica de negocio (modelo) y por último una interfaz gráfica que facilite al usuario el uso del sistema. [17]

La arquitectura de tres capas se refiere a un diseño reciente que introduce una capa intermedia al proceso. Cada capa es un proceso separado y bien definido corriendo en plataformas separadas.

En la arquitectura tradicional de tres capas, se instala una interfaz de usuario en la computadora del usuario final (el cliente). La arquitectura basada en WEB transforma la interfaz de búsqueda existente (el explorador de WEB), en la interfaz del usuario final. La parte funcional de la arquitectura de tres capas generalmente es conocida como la capa intermedia o el servidor de aplicaciones. Aquí es donde la mayoría de los procesos ocurren.

La tercera capa comúnmente es el sistema de administración de la base de datos. Es decir, donde los datos requeridos por la capa intermedia son almacenados. La tercera capa se localiza en un servidor separado conocido como el servidor de base de datos. [19]

- Presentación

Como su nombre indica, se limita a la navegabilidad y a gestionar todos aquellos aspectos relacionados con la lógica de presentación de la aplicación, como comprobación de datos de entrada, formatos de salida, internacionalización de la aplicación.

- Negocio o dominio

El resultado del análisis funcional de la aplicación viene a ser la identificación del conjunto de reglas de negocio que abstraen el problema real a tratar. Estas son las que realmente suponen el motor del sistema, dado que se basan en el funcionamiento del modelo real.

- Acceso a datos

Esta capa es la encargada de hacer persistir las entidades que se manejan en negocio, el acceso a los datos almacenados, la actualización, y otras operaciones sobre ellos, aunque puede ofrecer servicios relacionados con la persistencia o recuperación de información más complejos. [20]

Ventajas de una arquitectura de tres capas

Las llamadas de la interfaz del usuario, en la estación de trabajo, al servidor de capa intermedia, son más flexibles que en el diseño de dos capas, ya que la estación sólo necesita transferir parámetros a la capa intermedia. Con la arquitectura de tres capas, la interfaz del cliente no es requerida para comprender o comunicarse con el receptor de los datos. Por lo tanto, esa estructura de los datos puede ser modificada sin cambiar la interfaz del usuario en la computadora. El código de la capa intermedia puede ser reutilizado por múltiples aplicaciones si está diseñado en formato modular. Esto puede reducir los esfuerzos de desarrollo y mantenimiento, así como los costos de migración. La separación de roles en tres capas, hace más fácil reemplazar o modificar una capa sin afectar a los módulos restantes. Separando la aplicación de la base de datos, se hace más fácil utilizar nuevas tecnologías de agrupamiento y balance de cargas. Separando la interfaz de usuario de la aplicación, se libera de gran procesamiento a la estación de trabajo y permite que las actualizaciones de la aplicación sean centralizadas en el servidor de aplicaciones.

Desventajas de las arquitecturas de tres capas y basadas en WEB

Los ambientes de tres capas pueden incrementar el tráfico en la red y requerir más balance de carga y tolerancia a las fallas. Los exploradores actuales no son todos iguales. La estandarización entre diferentes proveedores ha sido lenta en desarrollarse. Muchas organizaciones son forzadas a escoger uno en lugar de otro, mientras que cada uno ofrece sus propias y distintas ventajas. [19]

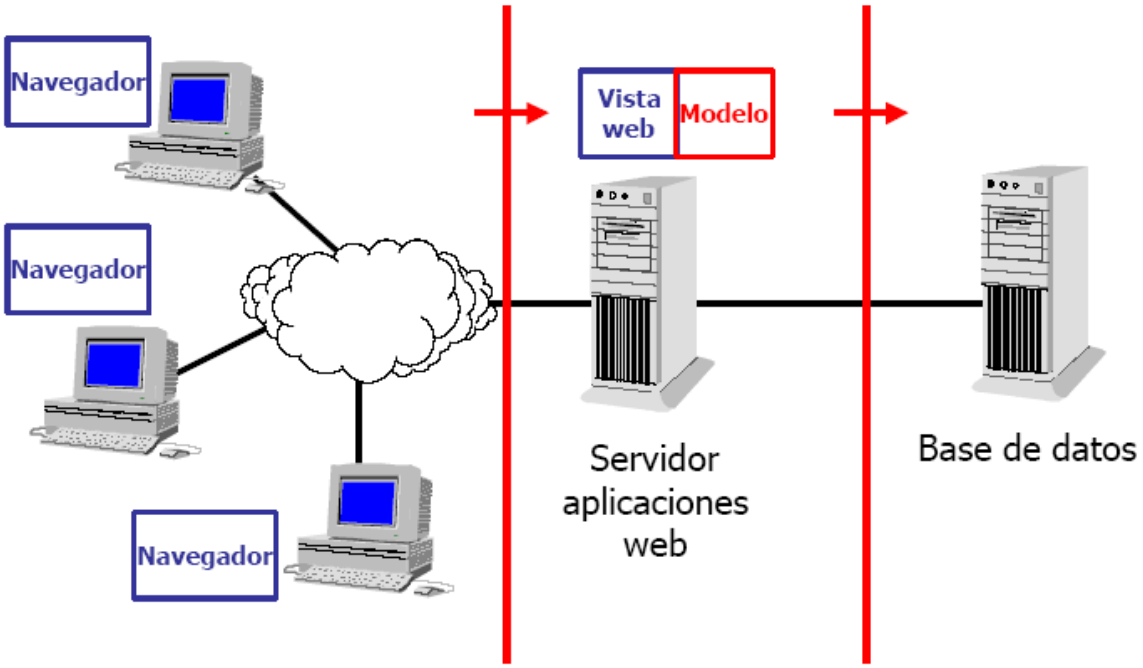

Fig. 2. Arquitectura Web en tres capas.

La arquitectura que se propone para este sistema es una arquitectura en tres capas, donde la capa de presentación será implementada con el *Framework JSF* (*Java Server Faces*), la capa de negocio con el *Framework Spring* y la capa de acceso a datos con el *Framework Hibernate*, todos los *Frameworks* antes mencionados son del lenguaje de programación Java, utilizado para la implementación del sistema.

2.6 Conclusiones.

En este capítulo se hizo una descripción del sistema a implementar. Se describen las funcionalidades que debe cumplir y los requerimientos funcionales, además se describe la arquitectura en tres capas, generalmente utilizada en proyectos implementados en el lenguaje Java. A partir de aquí se le empezará a dar solución al problema planteado, siguiendo la metodología XP como se ha propuesto antes.

Capítulo 3. Exploración y Planificación.

3.1 Introducción.

En este capítulo se describe el desarrollo del software, que dará solución al problema planteado, a través de las primeras fases de la metodología XP. Además, se generan los artefactos de estas etapas de desarrollo.

3.2 Fase 1: Exploración.

En esta fase, los clientes plantean a grandes rasgos las historias de usuario que son de interés para la primera entrega del producto. Al mismo tiempo el equipo de desarrollo se familiariza con las herramientas, tecnologías y prácticas que se utilizarán en el proyecto.

Se prueba la tecnología y se exploran las posibilidades de la arquitectura del sistema, construyendo un prototipo. La fase de exploración toma de pocas semanas a pocos meses, dependiendo del tamaño y familiaridad que tengan los programadores con la tecnología. [16]

3.2.1 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.

Estas deben proporcionar sólo el detalle suficiente como para poder hacer razonable la estimación de cuánto tiempo requiere la implementación de la historia, difiere de los casos de uso porque son escritos por el cliente, no por los programadores, empleando terminología del cliente. “Las historias de usuario son más “amigables” que los casos de uso formales”. [16]

Las Historias de Usuario tienen tres aspectos:

Tarjeta:

En ella se almacena suficiente información para identificar y detallar la historia.

Conversación:

Cliente y programadores discuten la historia para ampliar los detalles (verbalmente cuando sea posible, pero documentada cuando se requiera confirmación).

Pruebas de Aceptación:

Permite confirmar que la historia ha sido implementada correctamente.

Historia de Usuario	
Número:	**Usuario:**
Nombre historia:	
Prioridad en negocio: (Alta / Media / Baja)	**Riesgo en desarrollo:** (Alta / Media / Baja)
Puntos estimados:	**Iteración asignada:**
Programador responsable:	
Descripción:	
Observaciones:	

Tabla 3.1 Descripción de la Tarjeta de Historia de Usuario.

Campos de la tarjeta Historia de Usuario.

Número:

Índice de la historia de usuario. Es un número único que se le asigna a cada historia de usuario con el fin de lograr una mejor organización de éstas.

Nombre historia:

Nombre de la historia de usuario. Debe ser descriptivo, en la medida de las posibilidades, de lo que se implementará y no muy extenso.

Descripción:

Se describe el requerimiento que da origen a la historia de usuario. La descripción debe ser corta, precisa y dejar claro qué es lo que se desea hacer.

Observaciones:

Aspectos que se deben tener en cuenta para implementar la historia de usuario. Precondiciones o poscondiciones que se deben tener presentes al implementar la historia de usuario.

En la fase de exploración se identificaron seis historias de usuario, las que se describen a continuación.

1. Autenticar usuario.
2. Subir documento al sistema.
3. Comparar documentos.
4. Guardar documentos en la base de datos del sistema.
5. Generar reportes.
6. Ver estadísticas de plagio por criterios.

Historia de Usuario	
Número: 1	**Usuario:** Todos los usuarios del sistema.
Nombre historia: Autenticar usuario.	
Prioridad en el negocio: Alta.	**Riesgo en el desarrollo:** Bajo.
Puntos estimados: 1	**Iteración asignada:** 1
Descripción: El usuario podrá acceder al sistema según sus privilegios.	
Observaciones: Se ingresan los datos elementales: usuario y contraseña UCI. Los diferentes usuarios serán: Administrador, Profesor, Profesor con privilegios y Estudiante.	

Tabla 3.2 Representación de la Historia de Usuario Nro. 1

Historia de Usuario	
Número: 2	**Usuario:** Todos los usuarios del sistema.
Nombre historia: Subir documento al sistema.	
Prioridad en el negocio: Alta.	**Riesgo en el desarrollo:** Alto.
Puntos estimados: 2	**Iteración asignada:** 1
Descripción: El usuario escribe los datos que le pide la aplicación y sube el documento. La aplicación muestra un mensaje si el documento ha sido subido satisfactoriamente, de lo contrario, da la posibilidad de hacerlo nuevamente.	
Observaciones: El usuario debe de estar autenticado. Si es estudiante solo puede subir un documento para comparar.	

Tabla 3.3 Representación de la Historia de Usuario Nro. 2

Historia de Usuario	
Número: 3	**Usuario:** Todos los usuarios del sistema.
Nombre historia: Comparar documentos.	
Prioridad en el negocio: Alta.	**Riesgo en el desarrollo:** Alto.
Puntos estimados: 3	**Iteración asignada:** 1
Descripción: El sistema compara el texto del documento que se ha subido, con el texto de los documentos de la base de datos.	
Observaciones: Debe existir conexión con la base de datos que tiene guardados los documentos.	

Tabla 3.4 Representación de la Historia de Usuario Nro. 3

Historia de Usuario	
Número: 4	**Usuario:** Administrador, profesor o profesor privilegiado.
Nombre historia: Guardar documentos en la base de datos del sistema.	
Prioridad en el negocio: Alta.	**Riesgo en el desarrollo:** Alto.
Puntos estimados: 3	**Iteración asignada:** 1
Descripción: El sistema guarda en la base de datos, si el usuario lo desea, el documento subido para luego usar ese documento en la comparación con otros documentos nuevos.	
Observaciones: Debe existir conexión con la base de datos que tiene guardados los documentos.	

Tabla 3.5 Representación de la Historia de Usuario Nro. 4

Historia de Usuario	
Número: 5	**Usuario:** Todos los usuarios del sistema.
Nombre historia: Generar reportes.	
Prioridad en el negocio: Alta.	**Riesgo en el desarrollo:** Medio.
Puntos estimados: 1	**Iteración asignada:** 2
Descripción: El sistema le muestra al usuario si se ha detectado similitud con otros documentos y, en caso afirmativo, los datos de los documentos con los que tiene similitud.	
Observaciones: El usuario debe de estar autenticado y haber subido un documento al sistema.	

Tabla 3.6 Representación de la Historia de Usuario Nro. 5

Historia de Usuario	
Número: 6	**Usuario:** Administrador y Profesor con privilegios.
Nombre historia: Ver estadísticas de plagio por criterios.	
Prioridad en el negocio: Media	**Riesgo en el desarrollo:** Medio.
Puntos estimados: 1	**Iteración asignada:** 2
Descripción: El sistema le muestra al usuario las estadísticas de plagio detectado por el sistema según el criterio de búsqueda que dicho usuario hay escogido.	
Observaciones:	

Tabla 3.7 Representación de la Historia de Usuario Nro. 6

3.3 Fase 2: Planificación.

En esta fase se priorizan las historias de usuario y se acuerda el alcance del *release* del sistema. Los programadores estiman cuánto esfuerzo requiere cada historia y a partir de allí se define el cronograma. La fase de planeamiento toma un par de días. Se deben incluir varias iteraciones para lograr un *release*. El cronograma fijado en la etapa de planeamiento se realiza a un número de iteraciones, cada una toma de una a cuatro semanas en ejecución. La primera iteración crea un sistema con la arquitectura del sistema completo. Esto es alcanzado seleccionando las historias que harán cumplir la construcción de la estructura para el sistema completo. El cliente decide las historias que se seleccionarán para cada iteración. Las pruebas funcionales creadas por el cliente se ejecutan al final de cada iteración. Al final de la última iteración el sistema está listo para producción.

Las estimaciones de esfuerzo asociado a la implementación de las historias se establecen utilizando como medida el punto. Un punto, equivale a una semana ideal de programación. Las historias generalmente valen de 1 a 3 puntos. Por otra parte, se mantiene un registro de la “velocidad” de desarrollo, establecida en puntos por iteración, basándose principalmente en la suma de puntos correspondientes a las historias de usuario que fueron terminadas en la última iteración.

La planificación se puede realizar basándose en el tiempo o el alcance. La velocidad del proyecto es utilizada para establecer cuántas historias se pueden implementar antes de una fecha determinada o cuánto tiempo tomará implementar un conjunto de historias. Al planificar por tiempo, se multiplica el número de iteraciones por la velocidad del proyecto, determinándose cuántos puntos se pueden completar. Al planificar según el alcance del sistema, se divide la suma de puntos de las historias de usuario seleccionadas entre la velocidad del proyecto, obteniendo el número de iteraciones necesarias para su implementación. [17]

3.3.1 Estimación de esfuerzos por historias de usuario

Se realizó una estimación para cada una de las historias de usuario identificadas, para el buen desarrollo del sistema propuesto, llegando a los resultados que se muestran a continuación:

Historia de Usuario	Puntos de Estimación
Autenticar usuario.	1
Subir documento al sistema.	2
Comparar documentos.	3
Guardar documentos en la base de datos del sistema.	3
Generar reportes.	1
Ver estadísticas de plagio por criterios.	1

Tabla 3.8 Estimación de esfuerzo por historias de usuario

3.3.2 Plan de iteraciones

Después de detallar e identificar las historias de usuario y estimar el esfuerzo propuesto para la realización de estas, se procede a la planificación de la etapa de implementación del sistema. Este plan define exactamente cuáles historias de usuario serán implementadas para cada iteración del sistema y las posibles fechas para estas liberaciones. Se decide realizar el sistema en dos iteraciones.

Iteración # 1:

Se implementan de las historias de usuario que tienen prioridad alta. Al finalizar se contará con las funcionalidades descritas en las historias de usuario 1, 2, 3 y 4, las cuales dan respuesta a las funcionalidades de Autenticar usuario, Subir documentos al sistema, Comparar documentos y Guardar documentos en la base de datos del sistema.

Iteración # 2:

Aquí se implementan las restantes funcionalidades del sistema, todas las que quedan son de prioridad media o baja. Al terminar se tendrán implementadas las peticiones del cliente descritas en las historias de usuario 5 y 6 en las que se le da cumplimiento a las funcionalidades de Generar reportes y Ver estadísticas de plagio por criterios.

3.3.3 Plan de duración de las iteraciones

Todo proyecto que utiliza la Metodología XP, como parte de su ciclo de vida, crea un plan de duración de cada una de las iteraciones, en este caso se hace para el único equipo de desarrollo con el cual se cuenta. Este plan tiene como objetivo mostrar la duración de cada iteración, así como el orden en que serán implementadas las historias de usuario en cada una de las mismas.

Iteraciones	Orden de las historias usuario a implementar	Duración total de las iteraciones (semanas)
Iteración 1	1. Autenticar usuario. 2. Subir documento al sistema. 3. Comparar documentos. 4. Guardar documentos en la base de datos del sistema.	9
Iteración 2	1. Generar reportes. 2. Ver estadísticas de plagio por criterios.	2

Tabla 3.9 Plan de duración de las iteraciones.

3.3.4 Plan de entregas

En este plan se ajustan las funcionalidades de un mismo tema en módulos.

Módulos	Historias de usuario que abarca
Usuario	1. Autenticar usuario.
Análisis de documento	1. Subir documento al sistema. 2. Comparar documentos. 3. Guardar documentos en la base de datos del sistema.
Reporte	1. Generar reportes. 2. Ver estadísticas de plagio por criterios.

Tabla 3.10 Plan de entregas.

Módulo	1ra iteración (Finalizada 26 abril – 12 mayo)	2da iteración (Finalizada 20 mayo – 5 junio)
Usuario	1.0	Finalizada
Análisis de documento	0.1	Finalizada
Reporte	----	Finalizada

Tabla 3.11 Plan de duración de entregas.

3.4 Conclusiones.

En este capítulo se hace una descripción de las fases de Exploración y Planificación de la metodología de desarrollo de software utilizada en este sistema y se describen todos los artefactos que en ellas se generan para posibilitar el paso a la siguiente fase de desarrollo.

Capítulo 4. Implementación y pruebas del sistema.

4.1 Introducción.

En este capítulo se describen los artefactos generados en la implementación del sistema y las pruebas hechas al mismo. Al comienzo de una iteración cada historia de usuario que se implementará se fragmenta en diferentes tareas de programación. Posteriormente se elaboran los casos de prueba.

4.2 Tareas de programación.

Las tareas de programación 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 las tareas de programación se derivan directamente de las historias de usuario. [21]

Existen dos tipos de tareas de programación las que provienen de las historias de usuario y las técnicas.

Cada historia de usuario puede ser dividida en varias tareas de programación sencillas. Para determinar las tareas de programación que componen a una historia de usuario se propone hacer una reunión con todos los miembros del equipo de desarrollo y obtener una buena lista con todas. [22]

Las tareas de programación técnicas son aquellas que no son resultado del análisis de ninguna historia de usuario pero deben ser realizadas para que el sistema funcione. [22]

Cada tarea de programación será comprobada a través de los casos de prueba. Las tareas de programación no tienen por qué ser comprendidas por el cliente. [21]

Tarea de programación	
Número tarea:	**Número historia:**
Nombre tarea:	
Tipo de tarea : Desarrollo / Corrección / Mejora / Otra (especificar)	**Puntos estimados:**
Fecha inicio:	**Fecha fin:**
Programador responsable:	
Descripción:	

Tabla 4.1 Descripción de los campos de una tarjeta de Tarea de programación.

Campos de la tarjeta Tarea.

Número historia de usuario:

Número de la historia de usuario a la que corresponde. Índice de la historia de usuario a la que se corresponde esta tarea de programación.

Número tarea:

Índice de la tarea de programación. Es un número único que se le asigna a cada tarea de programación que pertenece a una historia de usuario determinada con el fin de lograr una mejor organización de éstas.

Nombre tarea:

Nombre de la tarea de programación. Debe ser descriptivo, en la medida de las posibilidades, de lo que se realizará y no muy extenso.

Tipo de tarea:

Informa el tipo de tarea a realizar. Las tareas pueden ser:

Desarrollo: Tarea que se realizará por primera vez.

Corrección: Tarea que se realiza a partir de una anterior que no se realizó correctamente, es decir no pasó todos los casos de prueba que le corresponden correctamente.

Mejora: Tarea que se realiza a partir de una anterior que se realizó correctamente pero se incorporan nuevos requerimientos para la misma, ahora tendrá que ser modificada para pasar satisfactoriamente los nuevos casos de prueba adicionados.

Otra: Tarea que no corresponde con ninguna de las anteriormente descritas.

Fecha inicio:

Fecha en que se inicia la implementación de la tarea de programación.

Fecha fin:

Fecha en que concluye la implementación de la tarea de programación.

Programador responsable:

Programador responsable de implementar la tarea de programación.

Descripción:

Se describe qué es lo que se desea realizar. La descripción debe ser corta y precisa.

Puntos estimados:

Tiempo estimado que demorará la implementación de la tarea de programación. Tiempo ideal en que se estima se implementará la tarea de programación.

Historias de Usuarios divididas en tareas.

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

Historia de Usuario	Tareas de programación
Autenticar usuario.	1. Introducir usuario y contraseña UCI. 2. Comprobar datos de usuario en la base de datos de los usuarios de la UCI. 3. Permitir o denegar el acceso a la aplicación.
Subir documento al sistema.	1. Pedir datos del documento. (Nombre, Tipo, Asignatura, Autores, Profesor, Facultad y Grupo) 2. Cargar el documento que se desea comparar.
Comparar documentos.	1. Leer documento cargado. 2. Abrir documento de la base de datos que se va a comparar con este. 3. Comparar el texto escrito en ambos documentos.
Guardar documentos en la base de datos del sistema.	1. Guardar los datos del documento en la base de datos. 2. Guardar el documento comparado en la base de datos del sistema. 3. Guardar información de la comparación de dicho documento en la base de datos.
Generar reportes.	1. Mostrar resultados del análisis del documento.
Ver estadísticas de plagio por criterios.	1. Permitir seleccionar criterio para ver las estadísticas. 2. Mostrar las estadísticas del criterio seleccionado.

Tabla 4.2 Descripción de las Tareas de Programación por cada Historia de Usuario.

Descripción de las Tareas de programación por Historias de Usuario.

Tarea de programación	
Número tarea: HU1_1	**Número historia:** 1
Nombre tarea: Introducir usuario y contraseña UCI.	
Tipo de tarea: Desarrollo	**Puntos estimados:**
Fecha inicio:	**Fecha fin:**
Descripción: Se muestra una página solicitando los datos del usuario para entrar a la aplicación. El usuario introduce los datos necesarios (usuario y contraseña UCI) para acceder a la aplicación.	

Tabla 4.3 Descripción de la Tarea de Programación HU1_1.

Tarea de programación	
Número tarea: HU1_2	**Número historia:** 1
Nombre tarea: Comprobar datos de usuario en la base de datos de los usuarios de la UCI.	
Tipo de tarea: Desarrollo	**Puntos estimados:**
Fecha inicio:	**Fecha fin:**
Descripción: Comprobar que exista conexión con la base de datos UCI, de no ocurrir la operación, notificar al usuario. Luego se comprueba que los datos introducidos por el usuario sean correctos.	

Tabla 4.4 Descripción de la Tarea de Programación HU1_2.

Tarea de programación	
Número tarea: HU1_3	**Número historia:** 1
Nombre tarea: Permitir o denegar el acceso a la aplicación.	
Tipo de tarea: Desarrollo	**Puntos estimados:**
Fecha inicio:	**Fecha fin:**
Descripción: Si los datos de autenticación son incorrectos se le informa al usuario y se le permite volver a insertar los datos requeridos (usuario y contraseña UCI). Si los datos son correctos se le muestra al usuario una página con los menús correspondientes a los privilegios de ese usuario. Los privilegios de usuario son los siguientes: si el usuario es estudiante solo puede comparar un documento, si es profesor puede comparar todos los documentos que desee, si es profesor privilegiado puede ver las estadísticas de plagio, el administrador tiene acceso a todo.	

Tabla 4.5 Descripción de la Tarea de Programación HU1_3.

Tarea de programación	
Número tarea: HU2_1	**Número historia:** 2
Nombre tarea: Pedir datos del documento.	
Tipo de tarea: Desarrollo.	**Puntos estimados:**
Fecha inicio:	**Fecha fin:**
Descripción: Se muestra una página en la que se piden los datos del documento y un botón “Examinar” que permitirá subir el documento al sistema.	

Tabla 4.6 Descripción de la Tarea de Programación HU2_1.

Tarea de programación	
Número tarea: HU2_2	**Número historia:** 2
Nombre tarea: Cargar el documento que se desea comparar.	
Tipo de tarea: Desarrollo.	**Puntos estimados:**
Fecha inicio:	**Fecha fin:**
Descripción: El documento se guarda en un lugar definido por los programadores para ser leído con posterioridad. De no ocurrir la operación, notificar al usuario.	

Tabla 4.7 Descripción de la Tarea de Programación HU2_2.

Tarea de programación	
Número tarea: HU3_1	**Número historia:** 3
Nombre tarea: Leer documento cargado.	
Tipo de tarea: Desarrollo.	**Puntos estimados:**
Fecha inicio:	**Fecha fin:**
Descripción: Se abre el fichero y se lee el texto con ayuda de librerías del lenguaje utilizado. De no ocurrir la operación, notificar al usuario.	

Tabla 4.8 Descripción de la Tarea de Programación HU3_1.

Tarea de programación	
Número tarea: HU3_2	**Número historia:** 3
Nombre tarea: Abrir documento de la base de datos que se va a comparar con este.	
Tipo de tarea: Desarrollo.	**Puntos estimados:**
Fecha inicio:	**Fecha fin:**
Programador responsable:	
Descripción: Se comprueba que exista conexión con la base de datos de los documentos. Se abre el documento correspondiente y se lee el texto de este con ayuda de librerías del lenguaje utilizado. De no ocurrir la operación, notificar al usuario.	

Tabla 4.9 Descripción de la Tarea de Programación HU3_2.

Tarea de programación	
Número tarea: HU3_3	**Número historia:** 3
Nombre tarea: Comparar el texto escrito en ambos documentos.	
Tipo de tarea: Desarrollo.	**Puntos estimados:**
Fecha inicio:	**Fecha fin:**
Descripción: Se va leyendo el texto de ambos documentos que se estén comparando empleando el algoritmo implementado por el programador. Comparar el documento subido con todos los documentos que están en la base de datos. De no ocurrir la operación, notificar al usuario.	

Tabla 4.10 Descripción de la Tarea de Programación HU3_3.

Tarea de programación	
Número tarea: HU4_1	**Número historia:** 4
Nombre tarea: Guardar los datos del documento en la base de datos.	
Tipo de tarea: Desarrollo.	**Puntos estimados:**
Fecha inicio:	**Fecha fin:**
Descripción: Si el usuario desea guardar el documento subido al sistema en la base de datos, y el documento no está plagiado, entonces se guardan los datos que describen a dicho documento en la base de datos del sistema.	

Tabla 4.11 Descripción de la Tarea de Programación HU4_1.

Tarea de programación	
Número tarea: HU4_2	**Número historia:** 4
Nombre tarea: Guardar el documento comparado en la base de datos del sistema.	
Tipo de tarea: Desarrollo.	**Puntos estimados:**
Fecha inicio:	**Fecha fin:**
Descripción: Luego que se ha terminado de comparar el documento subido al sistema, si el usuario desea guardar el documento subido en la base de datos del sistema, y el documento no está plagiado, se guarda dicho documento en la base de datos para ser utilizado nuevamente en otra operación de comparar. De no ocurrir la operación, notificar al usuario.	

Tabla 4.12 Descripción de la Tarea de Programación HU4_2.

Tarea de programación	
Número tarea: HU4_3	**Número historia:** 4
Nombre tarea: Guardar información de la comparación de dicho documento en la base de datos.	
Tipo de tarea: Desarrollo.	**Puntos estimados:**
Fecha inicio:	**Fecha fin:**
Descripción: El resultado de comparar el documento subido al sistema con cada uno de los documentos de la base de datos del sistema se debe guardar en la base de datos. De no ocurrir la operación, notificar al usuario.	

Tabla 4.13 Descripción de la Tarea de Programación HU4_3.

Tarea de programación	
Número tarea: HU5_1	**Número historia:** 5
Nombre tarea: Mostrar resultados del análisis del documento.	
Tipo de tarea: Desarrollo.	**Puntos estimados:**
Fecha inicio:	**Fecha fin:**
Descripción: Luego de efectuada la comparación, se le debe mostrar al usuario si existe similitud entre el documento y los documentos de la base de datos. En caso afirmativo se deben mostrar además los datos de los documentos con algún tipo de similitud.	

Tabla 4.14 Descripción de la Tarea de Programación HU5_1.

Tarea de programación	
Número tarea: HU6_1	**Número historia:** 6
Nombre tarea: Permitir seleccionar criterio para ver las estadísticas.	
Tipo de tarea: Desarrollo.	**Puntos estimados:**
Fecha inicio:	**Fecha fin:**
Descripción: Se muestra una lista con los diferentes criterios para las estadísticas y el usuario escoge el criterio que desea consultar. Esto solo lo pueden hacer los profesores privilegiados y el administrador del sistema.	

Tabla 4.15 Descripción de la Tarea de Programación HU6_1.

Tarea de programación	
Número tarea: HU6_2	**Número historia:** 6
Nombre tarea: Mostrar las estadísticas del criterio seleccionado.	
Tipo de tarea: Desarrollo.	**Puntos estimados:**
Fecha inicio:	**Fecha fin:**
Descripción: Se muestran organizadamente las estadísticas de plagio agrupadas por el criterio seleccionado.	

Tabla 4.16 Descripción de la Tarea de Programación HU6_2.

Descripción de las tareas de programación técnicas.

Tarea de programación	
Número tarea: T1	**Número historia:**
Nombre tarea: Diseñar de interfaz de las páginas de la aplicación.	
Tipo de tarea: Desarrollo.	**Puntos estimados:**
Fecha inicio:	**Fecha fin:**
Descripción: Diseñar cada una de las páginas del sistema, de manera que se cumplan todos los requerimientos. Las páginas deben ser de interfaz amigable para el usuario.	

Tabla 4.17 Descripción de la Tarea de Programación T1.

Tarea de programación	
Número tarea: T2	**Número historia:**
Nombre tarea: Crear la base de datos de datos de los documentos y la de guardar los documentos.	
Tipo de tarea: Desarrollo.	**Puntos estimados:**
Fecha inicio:	**Fecha fin:**
Descripción: Diseñar y crear las bases de datos de los documentos y de los datos de estos.	

Tabla 4.18 Descripción de la Tarea de Programación T2.

4.3 Diseño del sistema.

En XP se sigue una estrategia de diseño que consiste en siempre tener el diseño más sencillo posible que satisfaga los casos de pruebas. Un diseño sencillo es aquel que posea la menor cantidad de clases posible, las cuales deben poseer la menor cantidad de métodos posible. No debe existir código duplicado, y debe cumplir con todos los requisitos del software.

Los cuatro valores de la metodología XP: comunicación, simplicidad, retroalimentación y coraje, inciden directamente sobre la estrategia de diseño. Se debe crear un diseño simple donde cada elemento existente represente aspectos importantes del software a implementar, fácil de comunicar y confiar en que solo con lo diseñado es suficiente en ese mismo instante; que en el momento que se desee y sea necesario se puede incrementar el diseño.

Desarrollar software con los requerimientos imprecisos o cambiantes, es uno de los objetivos de XP, y para ello rompe con un hábito de la mayoría de los programadores: diseñar anticipándose a los problemas. El diseño debe ser inicialmente simple, progresivamente se le adicionará cuanto sea necesario y se refinará. Con ello se evita que el diseño adquiera una complejidad innecesaria, lo cual se traduce en pérdida de tiempo, debido a que, generalmente, lo que se supone nunca ocurre.

XP se basa en el principio de que el sistema debe ser diseñado tan sencillo como sea posible en cada momento; y sugiere eliminar las complejidades que no se necesiten tan pronto como sean encontradas.

Cada pieza del diseño de un sistema debe justificar su existencia. No se recomienda diseñar para un futuro incierto. Se recomienda diseñar solo lo necesario cuando sea necesario. [21]

Para llevar a cabo el diseño en XP se puede elaborar un diseño Clase-Responsabilidad-Colaborador (CRC) o utilizar Lenguaje de Modelado Unificado (UML).

Llevar a cabo la sesión de diseño no es obligatorio en cada iteración, esta se puede omitir si, por ejemplo, se tiene certeza de lo que se va a hacer. [23]

Por las características propias del sistema se decide no realizar ningún tipo de diseño, ya que la implementación a realizar se puede desarrollar satisfactoriamente sin la necesidad de diseñar las tareas a programar. Se considera innecesario el diseño de la aplicación porque es simple entender, por los programadores, las funcionalidades que la aplicación requiere.

4.4 Pruebas

Los casos de prueba son pruebas funcionales o unitarias que se realizan al sistema para comprobar su funcionamiento.

Las pruebas funcionales no son más que pruebas escritas desde la perspectiva del cliente, y las pruebas unitarias son pruebas escritas desde la perspectiva del programador. 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 cuando una historia de usuario está lista. [21]

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 historia de usuario y se verifique que cumpla con todos los requerimientos. [24]

XP anima a probar constantemente tanto como sea posible. Esto permite aumentar la calidad de los sistemas reduciendo el número de errores no detectados y disminuyendo el tiempo transcurrido entre la aparición de un error y su detección. También permite aumentar la seguridad de evitar efectos colaterales no deseados a la hora de realizar modificaciones y refactorizaciones.

Esta metodología divide las pruebas del sistema en dos grupos: pruebas unitarias, encargadas de verificar el código y diseñada por los programadores, y pruebas de aceptación o pruebas funcionales destinadas a evaluar si al final de una iteración se consiguió la funcionalidad requerida diseñadas por el cliente final.

Las pruebas del sistema tienen como objetivo verificar la funcionalidad del sistema a través de sus interfaces externas comprobando que dicha funcionalidad sea la esperada en función de los requisitos del sistema. Generalmente las pruebas del sistema son desarrolladas por los programadores para verificar que su sistema se comporta de la manera esperada, por lo que podrían encajar dentro de la definición de pruebas unitarias que propone XP.

Sin embargo, las pruebas del sistema tienen como objetivo verificar que el sistema cumple los requisitos establecidos por el usuario por lo que también pueden encajar dentro de la categoría de pruebas de aceptación.

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.

El cliente es la persona más indicada para escribir las pruebas funcionales porque no existe nadie que tenga una visión de lo que desea mejor que él. [24]

Una historia de usuario puede tener todas las pruebas de aceptación que necesite para asegurar su correcto funcionamiento. El objetivo final de estas es garantizar que los requerimientos han sido cumplidos y que el sistema es aceptable. [21]

Caso de Prueba de Aceptación	
Código:	**Historia de Usuario (Nro. y Nombre):**
Nombre:	
Descripción:	
Condiciones de Ejecución:	
Entrada / Pasos de ejecución:	
Resultado Esperado:	
Evaluación de la Prueba:	

Tabla 4.19 Descripción de los campos de un Caso de Prueba de Aceptación

Campos de la tarjeta Caso de Prueba

Número historia de usuario:

Número de la historia de usuario a la que corresponde. Índice de la historia de usuario a la que se le desea comprobar este aspecto.

Número:

Índice del caso de prueba. Es un número único que se le asigna a cada caso de prueba que pertenece a una historia de usuario determinada con el fin de lograr una mejor organización de estos.

Nombre:

Nombre del caso de prueba. Debe ser descriptivo, en la medida de las posibilidades, de lo que se comprobará y no muy extenso.

Descripción:

Se describe qué es lo que se desea probar. La descripción debe ser corta y precisa.

Fecha:

Fecha en que se realiza el caso de prueba. En caso de realizarse varias veces siempre se actualiza con la última.

Tipo de prueba:

Informa si la prueba que se realiza es funcional o unitaria.

Condiciones de ejecución:

Condiciones especiales que deben tenerse en cuenta para ejecutar el caso de prueba.

Entradas:

Entradas al caso de prueba en caso de necesitarlas.

Resultado esperado:

Resultado que se desea tenga el caso de prueba. Descripción breve de lo que debe suceder.

Evaluación:

Se evalúa si el caso de prueba tuvo éxito o no. En caso de ser exitoso se asigna un resultado de satisfactorio, en caso contrario insatisfactorio.

Casos de prueba del sistema

Caso de Prueba de Aceptación	
Código: 1	**Historia de Usuario (Nro. y Nombre):** HU #1 Autenticar usuario.
Nombre: Comprobar autenticar usuario.	
Descripción: Prueba para la funcionalidad de autenticar usuario del dominio UCI.	
Condiciones de Ejecución: Entrar al sistema el usuario y la contraseña UCI.	
Entrada / Pasos de ejecución: Se intenta introducir el usuario y la contraseña para entrar a la aplicación.	
Resultado Esperado: El usuario entra satisfactoriamente.	
Evaluación de la Prueba: Prueba satisfactoria.	

Tabla 4.20 Descripción del Caso de Prueba de Aceptación 1.

Caso de Prueba de Aceptación	
Código: 2	**Historia de Usuario (Nro. y Nombre):** HU #2 Subir documento al sistema.
Nombre: Comprobar que los datos del documento se inserten correctamente en el sistema.	
Descripción: Prueba para la funcionalidad de insertar los datos del documento en el sistema.	
Condiciones de Ejecución: El usuario tiene que estar autenticado.	
Entrada / Pasos de ejecución: Se intentan entrar los datos del documento al sistema.	
Resultado Esperado: Los datos se insertan satisfactoriamente al sistema.	
Evaluación de la Prueba: Prueba satisfactoria.	

Tabla 4.21 Descripción del Caso de Prueba de Aceptación 2.

Caso de Prueba de Aceptación	
Código: 3	**Historia de Usuario (Nro. y Nombre):** HU #2 Subir documento al sistema.
Nombre: Comprobar que el documento sea subido satisfactoriamente.	
Descripción: Prueba para la funcionalidad de subir un documento al sistema.	
Condiciones de Ejecución: El usuario tiene que estar autenticado.	
Entrada / Pasos de ejecución: Se intenta subir un documento al sistema.	
Resultado Esperado: El documento se sube al sistema sin generar error.	
Evaluación de la Prueba: Prueba satisfactoria.	

Tabla 4.22 Descripción del Caso de Prueba de Aceptación 3.

Caso de Prueba de Aceptación	
Código: 4	**Historia de Usuario (Nro. y Nombre):** HU #3 Comparar documentos.
Nombre: Comprobar que los documentos se comparan satisfactoriamente.	
Descripción: Prueba para la funcionalidad de comparar texto de los documentos.	
Condiciones de Ejecución: El usuario tiene que estar autenticado. Se debe haber subido un documento al sistema.	
Entrada / Pasos de ejecución: Se sube el documento al sistema. Se selecciona la opción de compararlo y se empieza a comparar este documento con los de su tipo en la base de datos.	
Resultado Esperado: El documento se compara con los demás documentos de su tipo que se encuentran almacenados.	
Evaluación de la Prueba: Prueba satisfactoria.	

Tabla 4.23 Descripción del Caso de Prueba de Aceptación 4

Caso de Prueba de Aceptación	
Código: 5	**Historia de Usuario (Nro. y Nombre):** HU #5 Generar reportes.
Nombre: Mostrar reportes de posibles plagios.	
Descripción: Prueba para la funcionalidad de generar reportes.	
Condiciones de Ejecución: El usuario tiene que estar autenticado. Se debe haber subido un documento al sistema. Se debe de haber comparado este documento con los demás documentos de su tipo.	
Entrada / Pasos de ejecución: Se compara el documento subido al sistema. Los resultados de la comparación se le muestran al usuario.	
Resultado Esperado: Se muestra el resultado de la comparación de los documentos.	
Evaluación de la Prueba: Prueba satisfactoria.	

Tabla 4.24 Descripción del Caso de Prueba de Aceptación 5.

Caso de Prueba de Aceptación	
Código: 6	**Historia de Usuario (Nro. y Nombre):** HU #6 Ver estadísticas de plagio por criterios.
Nombre: Mostrar estadísticas de los plagios.	
Descripción: Prueba para la funcionalidad de generar reportes.	
Condiciones de Ejecución: El usuario tiene que estar autenticado. Debe ser administrador o profesor con privilegios.	
Entrada / Pasos de ejecución: Se selecciona la opción de ver estadísticas y se escoge el criterio para mostrar los resultados.	
Resultado Esperado: Se muestran las estadísticas de plagio por el criterio seleccionado.	
Evaluación de la Prueba: Prueba satisfactoria.	

Tabla 4.25 Descripción del Caso de Prueba de Aceptación 6.

4.5 Conclusiones

En este capítulo se especificaron los artefactos necesarios para la fase de Implementación del sistema. Se describieron las tareas de programación necesarias para implementar las historias de usuario y las otras necesarias para desarrollar el sistema. Se detallaron además, las pruebas de aceptación con las que el cliente comprueba que la solución obtenida es la deseada y que cumple con la seguridad requerida. Aquí se da por terminada la propuesta de este trabajo de diploma.

Conclusiones generales

Durante el desarrollo de este trabajo se hizo un estudio de las diferentes herramientas antiplagio que existen actualmente y que se utilizan en las diferentes universidades del mundo para tratar de eliminar el llamado ciber plagio académico que ha cobrado auge debido al creciente uso de las tecnologías de la información y las comunicaciones (TIC) en la educación, sobre todo Internet, lo que facilita esta práctica éticamente reprobable y académicamente incorrecta.

La Universidad de las Ciencias Informáticas no está exenta de dicha práctica negativa para el desarrollo de las capacidades cognitivas de los estudiantes. Este Sistema Web Antiplagio, propuesto en el trabajo de diploma, motivará a que en la universidad se empiece a velar por la autenticidad de los documentos generados en la docencia, la producción y la investigación.

Dicho software, fue realizado con la metodología de desarrollo XP, lo que permitió que se realizara en un plazo de tiempo más corto que el que se hubiera empleado utilizando otra metodología tradicional, ganándose en rendimiento y organización.

El sistema, además, tiene como ventaja que no necesita ser instalado en la computadora y puede ser utilizado en cualquier sistema operativo.

Recomendaciones

Luego del desarrollo del Sistema Web Antiplagio se recomienda:

1. Hacer un estudio más detallado del algoritmo de comparación de texto utilizado en la implementación del sistema e incorporar otros algoritmos más rápidos y eficientes en futuras versiones del software.

2. Darle seguimiento al trabajo de diploma implementando un módulo de comparación de los documentos buscando en diferentes sitios de Internet.

3. Usar este sistema en la Universidad de las Ciencias Informáticas como método de corrección y búsqueda de posibles plagios en los documentos de cualquier índole generados por los estudiantes, por lo que se recomienda realizar en la universidad una base de datos general donde se almacenen los documentos generados por los estudiantes en las diferentes asignaturas para poder verificar su autenticidad.

4. Vincular este sistema al sitio http://teleformacion.uci.cu donde se encuentran todos los materiales docentes de la universidad.

Referencias bibliográficas

[1]. Diccionario Real Academia Española. [En línea] [Citado el: 09 de 02 de 2009.] http://buscon.rae.es/draeI/.

[2]. AntiCutAndPaste. [En línea] [Citado el: 30 de 01 de 2009.] www.fileheaven.com/descargar/anticutandpaste/28951.htm.

[3]. Educared. [En línea] [Citado el: 30 de 01 de 2009.] www.educared.net/mespana_intercampus/home_49_658_esp_1__.html.

[4]. Antiplagiarist. [En línea] [Citado el: 30 de 01 de 2009.] www.antiplagiarist.softonic.com.

[5]. ActiveFileCompare. [En línea] [Citado el: 30 de 01 de 2009.] www.freedownloadscenter.com/es/Programacion/Editores/Active_File_Compare.html.

[6]. **Alvarez, Miguel Angel.** AplicaciónWeb. [En línea] [Citado el: 03 de 02 de 2009.] www.desarrolloweb.com/articulos/332.php.

[7]. **Letelier, Patricio y Penadés, Mª Carmen.** *Metodologías ágiles para el desarrollo de software: eXtreme Programming (XP).* Universidad Politécnica de Valencia : s.n., 2007.

[8]. **Plaza Marín, Edison Arley y Anaya Villegas, Adrian.** *Desarrollo de software bajo metodologías ágiles (Agile Methods XP) en la práctica.*

[9]. PostgreSQL. [En línea] [Citado el: 03 de 02 de 2009.] www.escomposlinux.org/lfs-es/blfs-es-6.0/content/postgresql.html.

[10]. *La flecha, Diario de Ciencia y Tecnología*. (27 de agosto de 2007). [Citado el: 2009 de junio de 9] http://www.laflecha.net/canales/softlibre/red-hat-presenta-developer-studio-basado-en-eclipse/

[11]. **Martra, Pere.** UML. [En línea] [Citado el: 03 de 02 de 2009.] www.programacion.com/tutorial/uml/1/.

[12]. Java. [En línea] [Citado el: 03 de 02 de 2009.] www.lab.dit.upm.es/~fprg/asignatura/java.htm.

[13]. **Gómez Gallego, Juan Pablo y Galves, Ing Jorge.** *Fundamentos de la metodología RUP.* Universidad Técnológica de Pereira : s.n., 16/09/2007. (http://www.scribd.com/doc/297224/RUP)

[14]. Comparación ASP y Java. [En línea] [Citado el: 25 de 3 de 2009.] http://cek.bitacoras.com.

[15]. Tomcat. [En línea] [Citado el: 25 de 3 de 2009.] http://www.proactiva-calidad.com/java/herramientas/tomcat/index.html .

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

[17]. **Sánchez González, Carlos.** *Proyecto de fin de carrera de Ingeniería Informática.* Universidad de Coruña, Facultad de Informática - Departamento de Tecnologías de la Información y las Comunicaciones : s.n., Septiembre de 2004.

[18]. **Marcos Guglielmetti.** Definición de Arquitectura Software . [En línea] http://www.mastermagazine.info/termino/3916.php.

[19]. Aplicaciones basadas en Web. [En línea] http://www.janium.com/page2/page1/page6/page7/page7.html.

[20]. **Daniel Fernández Lanvin.** Definición de una arquitectura software para el diseño de aplicaciones web basadas en tecnología Java-J2EE. *Universidad de Oviedo - Estudios de Doctorado Avances en Informática - Curso Tecnologías WEB.* [En línea] http://pdimagenes.iespana.es/carlos/ArquitecturaJ2EE.PDF.

[21]. **Beck., K.** *Extreme Programming Explained. Embrace Change.* s.l. : Addison-Wesley, 1999.

[22]. **Beck, K.** *Martin Fowler Planning Extreme Programming.* s.l. : First Edition ed, 2000.

[23]. Jeffries, R., Anderson, A., & Hendrickson, C. (2000). *Extreme Programming.*

[24]. Cohn, M. (2004). *User Stories Applied: For Agile Software Development.* Addison.

[25]. *Viva Linux*. (16 de agosto de 2007). [Citado el: 10 de junio de 2009] http://www.vivalinux.com.ar/soft/redhat-developer-studio-1.0-beta1.html

Bibliografía

1. *Ciber-Plagio Académico. Una aproximación al estado de los conocimientos.* **Comas, Rubén y Sureda, Jaume.** 10. Temática Variada., s.l. : Revista TEXTOS de la CiberSociedad, 2007.

2. **Canós, José H., Letelier, Patricio y Penadés, Mª Carmen.** *Metodologías Ágiles para el desarrollo de software.* Universidad Politécnica de Valencia : s.n.

3. *Metodologías Ágiles en el Desarrollo de Software.* Alicante – España : s.n., 2003.

4. **Beck, Kent.** *Extreme Programming Explained: Embrace Change.* s.l. : Addison Wesley Longman, 2000.

5. *Embracing Change with Extreme Programming.* **Beck, Kent.** 10, s.l. : Revista de la IEEE Computer Society, 1999, Vol. 32.

6. **Hernán, Schenone Marcelo.** *Diseño de una Metodología Ágil de Desarrollo de Software.*

7. **Sánchez, Emilio A., Letelier, Patricio y Canós, José H.** *Mejorando la gestión de historias de usuario en eXtreme Programming.*

8. **Sarkela, J, McDonough, P y Caster, D.** *Embracing Change with Squeak: Extreme Programming.*

9. **Sánchez González, Carlos.** *ONess: un proyecto open source para el negocio textil mayorista desarrollado con tecnologías open source innovadoras. Proyecto de FIn de carrera de Ingeniería Informática.* s.l. : Universidade da Coruña - Facultad de Informática, 2004.

10. **Anaya Villegas, Adrián.** *A propósito de programación extrema XP (eXtreme Programming).*

11. **Wake, William C.** *Extreme Programming Explored.* 2000.

12. **Fernández Escribano, Gerardo.** *Introducción a Extreme Programming.* 2002.

13. **Weltman, Rob y Dahbura, Tony.** *LDAP programming with Java.* s.l. : Addison Wesley.

14. **Schildt, Herbert.** *Java 2: The Complete Reference.*

15. **Becerril C., Francisco.** *Java a su alcance.* s.l. : McGraw-Hill Interamericana Editore.

16. **Yuan, Michael y Heute, Thomas.** *JBoss Seam Simplicity and Power Beyond Java EE.* s.l. : Prentice Hall.

17. **Armstrong, Eric, y otros.** *Java Web Services Tutorial (2002).* s.l. : Addison Wesley.

18. **Jewell, Tyler y Chappell, David.** *Java Web Services.*

19. Cómo puedo hacer Upload de ficheros con JSF. [En línea] mayo de 2008. www.autentia.com.

20. **Salinas Caro, Patricio y Histchfeld K., Nancy.** UML. [En línea] [Citado el: 03 de 02 de 2009.] www.dcc.uchile.cl/~psalinas/uml/introduccion.html.

Glosario de términos

SWAP: Sistema Web Antiplagio.

Software: Es la suma total de los programas de cómputo, procedimientos, reglas, documentación y datos asociados que forman parte de las operaciones de un sistema de cómputo.

Software Antiplagio: Software que permite localizar e identificar documentos plagiados a partir de su análisis. Identifica a través de palabras claves si existe o no plagio en algún texto.

Ingeniería de Software: Es una tecnología multicapa en la que, se pueden identificar: los métodos (indican cómo construir técnicamente el software), el proceso (es el fundamento de la Ingeniería de Software, es la unión que mantiene juntas las capas de la tecnología) y las herramientas (soporte automático o semiautomático para el proceso y los métodos).

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.

Metodologías de Desarrollo: Se define como un conjunto de filosofías, etapas, procedimientos, reglas, técnicas, herramientas, documentación y aspectos de formación para los desarrolladores de sistemas de información.

Metodología Ágil: Constituyen un nuevo enfoque en el desarrollo de software, mejor aceptado por los desarrolladores de proyectos que las metodologías convencionales debido a la simplicidad de sus reglas y prácticas, su orientación a equipos de desarrollo de pequeño tamaño, su flexibilidad ante los cambios y su ideología de colaboración.

TIC: Tecnologías de la Información y las Comunicaciones.

UML: Lenguaje Unificado de Modelado (UML, por sus siglas en inglés, *Unified Modeling Language*) es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad; está respaldado por el OMG (*Object Management Group*). Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema de software.

***Framework*:** Herramienta cuyo fin es el de facilitarle a los programadores y diseñadores el desarrollo de determinado software, posee una estructura de soporte bien organizada y definida, permitiéndole al usuario del mismo no lidiar con detalles de bajo nivel.

***Web services*:** Es un conjunto de estándares y protocolos que sirven para el intercambio de datos entre aplicaciones.

***Servlet*:** Objeto que se ejecuta dentro de un contexto determinado ya sea de un servidor (Tomcat) o un contenedor (*JEE - Java Enterprise Edition* -) y cuya función es la de ofrecer contenido dinámicamente desde un servidor Web. Extienden su funcionalidad.

***Mozilla Firefox*:** Navegador Web multiplataforma, y de código abierto descendiente.

RUP: El Proceso Unificado Racional o RUP (*Rational Unified Process*), es un proceso desarrollo de software y junto con el Lenguaje Unificado de Modelado UML, constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos. Se caracteriza por ser iterativo e incremental, estar centrado en la arquitectura y guiado por los casos de uso. Incluye artefactos y roles.

Programación Extrema (XP): Es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo**.** Se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico.

***Release*:** Nueva versión de una aplicación informática. Se refiere a un producto final, preparado para lanzarse como versión definitiva a menos que aparezcan errores que lo impidan. En esta fase el producto implementa todas las funciones del diseño y se encuentra libre de cualquier error que suponga un punto muerto en el desarrollo.
Cliente: Rol que se encarga de escribir las historias de usuario, asignarle prioridad y escribir las pruebas funcionales.

Desarrollador: Miembro del equipo de desarrollo que puede encarar cualquiera de los roles de XP.
Programador: Rol encargado de diseñar, hacer pruebas, programar e integrar el sistema.

Equipo de desarrollo: Grupo de desarrolladores que implementa un software determinado, en el que se pueden encontrar diferentes roles.

Fase: Cada uno de los distintos estados sucesivos por los que puede pasar un software en su ciclo de vida.

Historia de usuario (HU): Tarjetas de papel donde se especifican los requisitos del software a implementar.

IDE: Un entorno de desarrollo integrado o en inglés *Integrated Development Environment* ('IDE') es un programa compuesto por un conjunto de herramientas para un programador.

Iteración: Período de 1 a 4 semanas en el que se implementa una o varias historias de usuarios.

Tareas de programación: Grupo de funcionalidades que los programadores conocen que el sistema debe hacer.