Universidad de las Ciencias Informáticas
FACULTAD 6

Título: "Sistema para la visualización y construcción de caché de mapas basado en tecnologías ágiles"

Trabajo de Diploma para optar por el Título de

Ingeniero en Ciencias Informáticas

Autor: Manuel Alejandro Pérez Rosabal.

Tutor: Alain Leon Companioni.

La Habana, Junio 2012.

"Año 54 de la Revolución"

Declaración de Autoría

Ciudad de La Habana, Julio, 2012.

“Año 54 de la Revolución”

Yo: Manuel Alejandro Pérez Rosabal declaro ser el único autor de este trabajo y autorizo a la Universidad de las Ciencias Informáticas (UCI) a hacer uso del mismo en su beneficio.

Para que así conste firmo la presente a los _____ días del mes de _____ del 2012.

_____	_____
Firma del Autor	Firma del Tutor
Manuel Alejandro Pérez Rosabal	Alain Leon Companioni

Datos del Contacto

Autor

Nombre: Manuel Alejandro

Apellidos: Pérez Rosabal

Teléfono: 58180327

Tutor

Nombre: Alain

Apellidos: León Companioni

Correo Electrónico: acompanioni@uci.cu

Teléfono: 53546653

Ingeniero en Ciencias Informáticas

Dedicatoria

A mi madre y mi padre que son todo para mí, por su amor, confianza, dedicación y entrega. Sin ellos no hubiese logrado ser lo que soy, mi triunfo es de ustedes.

A mi familia por ayudarme en los momentos difíciles, por su preocupación, consejos y palabras de aliento, por cada página que escribieron en mi vida.

Agradecimientos

A Dios por haberme permitido lograr mis objetivos, además de su infinita bondad y amor.

A mi madre Maira y a mi padre Manuel, no alcanzaría la vida para agradecerles todo lo han hecho y siguen haciendo por mí, a ustedes todo lo que soy. Por su perseverancia, por el valor mostrado para salir adelante, por ser mis compañeros y amigos de siempre y por el amor que me han brindado siempre. ¡Gracias!

A mis abuelos Mario, Nélida y a los que ya no están presentes por enseñarme valores como el honor, la responsabilidad, a ser fuerte y ayudarme en todo momento.

A mi familia mis parientes, tíos, tías, mis primos y más que primos mi hermanos, Yubi, Greisis y Leisis por quererme y por compartir conmigo como la gran familia que somos.

A todos los profesores que han impulsado mi formación profesional.

A mis amigos presentes y los que ya no están en la Universidad que me dejaron muchas enseñanzas y experiencias. A Pedro mi amigo incondicional gracias por sus consejos y sinceridad. A mi hermanita Yasnary ayudarme y haberme defendido en muchas ocasiones. A Yannia por ser parte de mi familia en Venezuela, por compartir conmigo momentos importantes y darme su apoyo siempre.

A mi tutor Alain Companioni y al tribunal por sus consejos, sugerencias y críticas constructivas que contribuyeron a realizar este trabajo con mejor calidad.

A la Revolución y a todas aquellas personas que de una u otra forma ayudaron en la elaboración de este trabajo de tesis…

Muchas Gracias…

Resumen

En la actualidad existe una consolidación de los Sistemas de Información Geográfica (SIG) como industria, donde los nuevos campos de innovación se encuentran en la integración de sistemas de soporte de decisiones para la divulgación de cartografías e información geográfica. Estas expectativas están presentes en Cuba y con el objetivo de ampliar su uso por otros usuarios, así como diversificar dichos sistemas, la Universidad de las Ciencias Informáticas ha reconocido el papel fundamental que desempeñan, creando varias soluciones en entornos de desarrollo de código abierto. Pero el elevado grupo de datos que se maneja en las soluciones creadas, aparejado al tiempo que consume el proceso de visualización de los mapas, genera la necesidad de utilizar una herramienta que permita la construcción de caché de dichos mapas para responder adecuadamente a las peticiones realizadas por los usuarios sobre la información geográfica. Por lo que, la presente investigación es el resultado del ciclo completo de desarrollo de *software* que propone la metodología *XP*, que basado en los principios de *software* libre y de tecnologías ágiles que utiliza la implementación *TileCache* para la construcción de servicio de caché de mapas que permita a las soluciones SIG creadas mejorar la velocidad de visualización de la información geográfica.

Palabras claves: Sistemas de Información Geográfica (SIG), Universidad de las Ciencias Informáticas, renderización, XP, *TileCache*, *software*.

Abstract

Nowadays there is a consolidation of Geographic Information Systems as an industry, where new fields of innovation are found in the integration of decision support systems for the dissemination of maps and geographic information. These expectations are present in Cuba and in order to expand its use by other users and diversify these systems, the University of Informatic Sciences has recognized the vital role by creating multiple solutions in environments of open source development. But the large set of data that is handled in the solutions created, coupled to the time taken to process map display, generates the need for a tool that allows the construction of such maps cache to respond appropriately to requests made by users about geographic information. So, this research is the result of the full cycle of software development proposed by the XP methodology, that based on the principles of open source and agile technologies that uses the implementation TileCache to build map cache service that enables GIS solutions created to improve the speed display geographic information..

Keywords: Geographic Information Systems (GIS), University of Informatic Sciences, rendering, extreme Programming (XP), TileCache.

Índice

Índice

Índice

Índice

Índice de tablas.

Índice

Índice

Índice de Figuras.

Introducción.

En la actualidad es muy común ver un mapa e identificar en él rasgos del territorio que son conocidos. En los cuales se maneja un conjunto de información alfanumérica: las características de un lugar, junto con suposición espacial (¿dónde está?) y sus relaciones espaciales (¿qué tan cerca está o qué caminos conectan?) por lo que la habilidad y la necesidad de los mapas son universales (Barroso, y otros, 1997).

La utilización de estos mapas impresos se hace cada vez más obsoleta y se ve desplazada por nuevas tecnologías, que por su sencillez y facilidad de interpretación facilitan la ubicación de puntos sobre la superficie de la tierra. Estas tecnologías, con el desarrollo de los programas de computadoras, han ayudado al hombre desde materias como la milicia o la explotación de recursos naturales hasta el salvamento de personas víctimas de accidentes o desastres naturales. Por lo que en este terreno, como en tantos otros, la utilización de herramientas informáticas ha proporcionado nuevos medios para abordar el problema: los Sistemas de Información Geográficos (SIG). (Barroso, y otros, 1997)

En la actualidad existe una consolidación del SIG como industria; caracterizado por una progresiva integración de sistemas ráster y vectoriales, además por el aumento de la importancia de las comunicaciones entre sistemas y la interfaz de usuario, así como por el uso de herramientas de programación tipo visual basadas en la metodología de orientación a objetos geográficos. Los nuevos campos de innovación de los SIG son la integración en sistemas de soporte de decisiones para divulgación de la cartografía y de la Información Geográfica. (Bravo, 2000)

Las expectativas creadas sobre los SIG están también presentes en Cuba con sus correspondientes limitaciones y singularidades, para superar la visión sectorial y consolidar una comprensión integral del territorio, mediante la interacción de las dimensiones ambiental, cultural, económica, social, espacial, entre otras. Los usuarios de los SIG en Cuba, la mayoría son geólogos, cartógrafos, geógrafos, desarrolladores, arquitectos e ingenieros, quienes conocen y operan Sistemas de Información Geográfica en sus investigaciones y proyectos. Muchos de estos especialistas han aprendido a manejar los SIG de forma autodidacta, en organizaciones oficiales del Gobierno, tales como GEOCUBA, Ministerio de Ciencia, Tecnología y Medio Ambiente, Instituto de Planificación Física, Ministerio de las Fuerzas Armadas y las Universidades. (Silva, 2005)

Con el objetivo de ampliar su uso por otros usuarios y diversificar dichos sistemas, la Universidad de las Ciencias Informáticas (UCI), se quiere convertir en prototipo de ciudad universitaria digital y de

sociedad de información en Cuba, reconociendo el papel fundamental que desempeñan los SIG. En la actualidad se han creado varias soluciones dentro del centro de Desarrollo y Producción de *Software*[i] Geoinformática y Sistemas Digitales GEySED, del Departamento Geoinformática, específicamente el proyecto Aplicativos SIG, perteneciente a la Facultad 6, como la Plataforma GeneSIG, creada a partir de tecnologías libres, entre ellas *MapServer*. Este último es un entorno de desarrollo en código abierto para la creación de aplicaciones SIG con el fin de visualizar, consultar y analizar información geográfica a través de la Web mediante la tecnología Internet *MapServer*. (Open Source Web Mapping, 2011)

Un área de trabajo del mencionado proyecto persigue desarrollar aplicaciones SIG sobre tecnologías libres que permitan proporcionar servicios de acceso a la información geográfica, para su consulta, análisis y visualización, a usuarios de cualquier tipo. Integrar información socioeconómica existente con la información geográfica asociada a cualquier negocio y facilitar solucionar problemas de cualquier índole, además permita aumentar la escalabilidad para poder hacer frente a las peticiones de los usuarios que acceden a dichos servicios, así como la elevar velocidad de la representación geoespacial de la información asociada a cualquier negocio.

Actualmente *MapServer* realiza con eficacia la visualización de la información, pero es notablemente lento debido al elevado grupo de datos a representar, además el proceso de generar los mapas consume mucho tiempo lo cual no permite al servicio responder adecuadamente a las peticiones realizadas por el usuario.

Por lo todo lo anteriormente expuesto, se puede identificar el siguiente **problema a resolver**: ¿Cómo mejorar la velocidad de respuesta en la visualización de los mapas para las soluciones SIG del Departamento Geoinformática?

Definiendo como **objeto de estudio**: el proceso de visualización de los mapas en los Sistemas de Información Geográfica. Donde el **campo de acción** lo constituyen las formas de visualización de los mapas utilizando tecnologías ágiles en la Universidad de las Ciencias Informáticas. Teniendo como **objetivo general**: Crear un sistema para la visualización y construcción de caché de mapas basado en tecnologías ágiles para el Departamento de Geoinformática.

Se defiende la siguiente **idea**: Con la creación de un sistema para la visualización y construcción de caché de mapas basado en tecnologías ágiles permitirá mejorar la velocidad de respuesta en la visualización de los mapas para las soluciones SIG del Departamento Geoinformática.

Para ello se especificaron las siguientes tareas:

1. Caracterizar el proceso de desarrollo de los Sistemas de Información Geográfica.
2. Seleccionar y argumentar las tendencias y tecnologías actuales a utilizar en el proceso.
3. Confeccionar las historias de usuario.
4. Confeccionar las tarjetas CRC.
5. Confeccionar las tareas de la ingeniería.
6. Implementar las historias de usuario.
7. Ejecutar las pruebas y validaciones de la solución propuesta.

Los métodos científicos que se utilizarán en el desarrollo de la investigación son los siguientes:

Métodos teóricos.

- Análisis y síntesis: Se usó para el estudio de la bibliografía utilizada y para realizar una recopilación de la misma.
- Modelación: Se utilizó para la modelación de los artefactos generados, representar el proceso de desarrollo y propiciar un mejor entendimiento de la solución a implementar.

Métodos empíricos.

- Observación: Se usó para el diagnóstico del problema a investigar, fue de gran utilidad en el diseño de la investigación y selección de aquellos aspectos que contribuyeron a la demostración de la hipótesis.

Al concluir la investigación se esperan los siguientes resultados:

- Un sistema que permita la visualización y construcción de caché de mapas basado en tecnologías ágiles.
- Documentación de la arquitectura base y de las tecnologías usadas.

El siguiente trabajo de diploma estará dividido en 4 capítulos fundamentales:

Capítulo 1: Fundamentación teórica. Se describen brevemente los conceptos fundamentales relacionados con el dominio del problema. Se realizará además un estudio acerca de las soluciones existentes, nacional e internacionalmente referentes al tema en cuestión.

Capítulo 2: Características del sistema. Se explican las principales tecnologías, lenguajes de programación y herramientas que se utilizarán para la construcción de la solución propuesta, así como las ventajas de utilizarla. Así como la metodología de desarrollo de *software* que se utilizará para la implementación del sistema.

Capítulo 3: Análisis y diseño de la solución propuesta. Aborda las dos primeras fases de la Programación Extrema (*XP*). En la fase de exploración se define las historias de usuario, para un mejor entendimiento del *software* y en la fase de planificación se realiza la estimación de esfuerzos por historia de usuario.

Capítulo 4: Implementación y prueba. Desarrolla las restantes fases de la Programación Extrema (XP), construcción. Donde se describen las tarjetas CRC (Contenido, Responsabilidad y Colaboración) y se detallan las iteraciones llevadas a cabo durante la etapa de construcción de la aplicación, así como las tareas generadas por cada historia de usuario y las pruebas de aceptación efectuadas sobre el sistema. Por último, se pone a prueba la solución implementada para que cumpla con los requisitos de calidad, diseño e implementación.

1. Capítulo I. Fundamentación Teórica.

1.1. Introducción.

En el siguiente capítulo se describen los conceptos asociados al dominio del problema, así como la descripción del objeto de estudio, referente al desarrollo de Sistemas de Información Geográfica utilizando tecnologías como *TileCache*[ii], haciendo uso de conceptos y definiciones, además de plantear y profundizar sobre la situación problémica, desde un punto de vista teórico para una mayor comprensión del problema que se aborda y se analizan algunas soluciones existentes nacionales e internacionales.

1.2. Conceptos asociados al dominio del problema.

La **información** es un conjunto de datos acerca de algún suceso, hecho o fenómeno, que organizados en un contexto determinado tienen su significado, cuyo propósito puede ser el de reducir la incertidumbre o incrementar el conocimiento acerca de algo. (Thompson, 2008)

De esta forma, la **información geográfica** es un conjunto de datos espaciales, los cuales brindan una información de algún hecho o fenómeno. (Gianfelici, 2008)

Entonces un **sistema de información** es un conjunto organizado de elementos que interactúan entre sí para procesar información. (WordPress, 2008)

Un **Sistema de Información Geográfica** se puede definir como un método o técnica de tratamiento de la información geográfica que permite combinar eficazmente información básica para obtener información derivada. Para ello, se cuenta tanto con las fuentes de información como con un conjunto de herramientas informáticas que facilitan esta tarea; todo ello enmarcado dentro de un proyecto que habrá sido definido por un conjunto de personas. En definitiva, un SIG es una herramienta capaz de combinar información gráfica y alfanumérica para obtener una información derivada sobre el espacio. (Bravo, 2000)

1.2.1. Mapa.

El mapa es una representación gráfica y métrica de una porción de territorio sobre una superficie bidimensional, que por lo general suele ser plana, aunque también puede ser esférica como en el caso de los globos terráqueos. El mismo puede ser representado en formato ráster o vertorial. (WordPress, 2008)

Gracias a sus propiedades métricas, un mapa permite tomar medidas de distancias, superficies y ángulos, con resultados casi exactos. Por eso, los mapas resultan una importante fuente de información y permiten desarrollar diversas actividades humanas que se basan en los datos que proporcionan.

1.2.2. Cartografía digital.

La cartografía digital es el procedimiento que transforma la información geográfica de los mapas de papel a coordenadas digitales, es decir, mapas digitales. Cuando hablamos de mapas que tienen un soporte digital, debemos diferenciar inmediatamente si se trata de tipo vectorial o de tipo ráster. (Cartografía digital y Espeleología, 2000)

El tipo vectorial utilizan el método de gráficos vectoriales. Es decir, cada objeto del mapa como puede ser una curva de nivel, un símbolo o texto guarda la definición geométrica y atributos del objeto que permiten generar la figura. Esta definición geométrica la representa mediante vectores y los atributos como son el grosor o el color. (Cartografía digital y Espeleología, 2000)

El tipo ráster utilizan el método grafico de mapas de bits, es decir, cada píxel del grafico está identificado con una posición y un color. (Cartografía digital y Espeleología, 2000)

1.2.3. WMS.

El servicio *Web Map Service* (Servicio de Mapas en la Web) definido por el *OGC* (*Open Geospatial Consortium*) produce mapas de datos referenciados espacialmente, de forma dinámica a partir de información geográfica. Este estándar internacional define un mapa como una representación de la información geográfica en forma de un archivo de imagen digital. Los mapas producidos por *WMS* se generan normalmente en un formato de imagen como PNG, GIF o JPEG y opcionalmente como gráficos vectoriales en formato *SVG*[iii] o WebCGM[iv]. (Open Geospatial Consortium, 1994-2010)

1.2.4. MapServer.

MapServer es un servidor de mapas *Open Source* (de código abierto), desarrollado actualmente por mapTools.org. Es un entorno sencillo que permite el desarrollo de aplicaciones SIG para Internet (o cualquier otra red donde pueda instalarse este servidor). Permite visualizar, consultar y analizar información geográfica a través de la red mediante la tecnología *Web*. Este puede ser utilizado como una aplicación *CGI*[v] o a través del acceso a la *API*[vi] de *MapServer* que proveen las librerías *Mapscript*. (Gianfelici, 2008)

1.2.5. Mapscript.

Mapscript constituye la vía de comunicación de las aplicaciones SIG con el servidor de mapas *Mapserver* y por otra parte rompe en cierta medida esta rigidez de la representación de mapas a través de los ficheros ".map", dichos ficheros definen los recursos que serán utilizados usados en la aplicación CGI, también contiene información acerca de cómo se debe dibujar el mapa, la leyenda y el resultado de realizar una consulta. *Mapscript* además permite modificar los mismos en tiempo de ejecución, facilitando de esta forma, la creación de aplicaciones con un grado de personalización mayor, no alcanzado así con aplicaciones de *MapServer* en modo *CGI*. (Espinosa, 2000)

1.2.6. TileCache.

TileCache[vii] es una implementación de un *WMS-C*[viii] disponible bajo la licencia *BSD*[ix] por *MetaCarta*. *TileCache* proporciona una base *WMS-C/TMS*[x] sobre un servidor compatible para *Python*[xi], con los mecanismos de almacenamiento en cachés acoplables y de representación. En el caso de uso más simple, *TileCache* solo requiere acceso de escritura al disco, la capacidad de ejecutar scripts CGI de *Python* y un WMS que se desea almacenar en caché. Con estos recursos, se puede crear una cache local de cualquier servidor *WMS* y utilizar el resultado en cualquier *WMS-C* de lado del cliente, como *OpenLayers*. (Meta Carta Labs, 2010)

1.3. Objeto de estudio.

1.3.1. Descripción general.

En la actualidad, los SIG plantean la solución para la eficiente administración y control de los recursos. Aunque los SIG se empezaron a generalizar a partir de la década de los 80, su gestación y desarrollo se remonta dos décadas atrás. Entre los años 1960 y 1964 se desarrolló el *Canadian Geographic Information System* (C.G.I.S.), con el objeto de gestionar los bosques y superficies marginales de Canadá. Este sistema ha ido evolucionando y sigue en uso en la actualidad. (Bravo, 2000)

Desde entonces hasta la actualidad, los SIG han alcanzado un gran desarrollo y expansión gracias fundamentalmente a la evolución y ampliación de las capacidades de los ordenadores, el desarrollo de los lenguajes de programación y el avance del tratamiento gráfico. (Bosque Sendra, 1997)

A continuación se visualizan algunas de las funcionalidades de los componentes de un SIG.

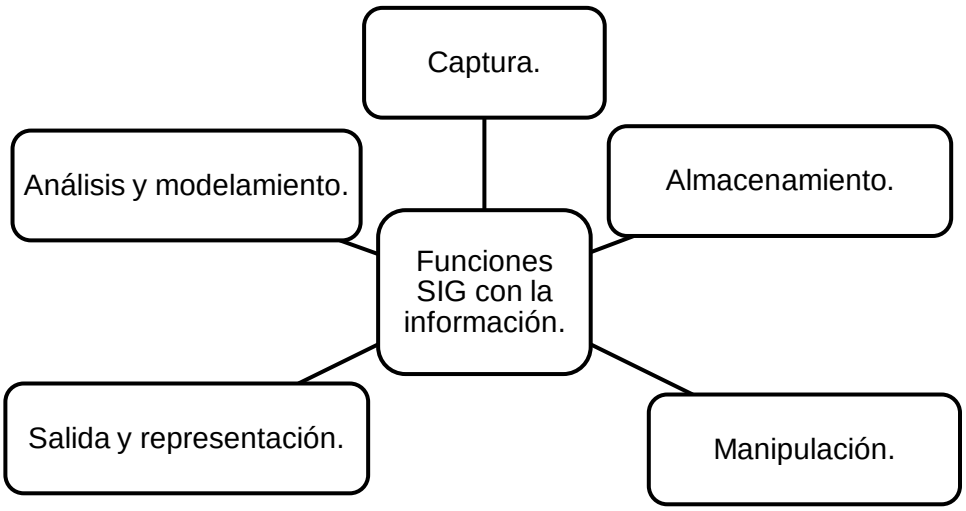

Figura 1. Funcionalidades de los SIG (Carmona, y otros, 1999).

La salida de información de un SIG puede ser de tipo textual o de tipo gráfico. Ambos tipos de información pueden ser presentados en forma digital o analógica.

El medio analógico es el que se presenta al usuario como respuesta a una interrogante de sí mismo. La información textual analógica consiste normalmente en un conjunto de tablas que representan la información almacenada en la base de datos o representan el resultado de algún tipo de análisis efectuado sobre esta. La información analógica gráfica consiste en mapas, gráficos o diagramas. El sistema debe tener la capacidad de complementar la información gráfica, antes de su presentación definitiva, por medio de una simbología adecuada y manejar la posibilidad de adicionar elementos geométricos que permitan una calidad y una visualización fáciles de entender por el usuario.

Para una mejor representación de la información geográfica surge la idea de tener las imágenes pre generadas para eliminar el coste de renderización[xii] basándose en un modelo de *tiles*[xiii]. Este modelo consiste en dividir el mapa cartográfico, ofrecido por el servidor, en una pirámide de mallas donde cada nivel de la pirámide corresponde con una resolución o escala del mapa y todas las celdas de la pirámide tienen un tamaño fijo en pixel[xiv].

Figura 2. Pirámide de tiles (Ramírez, 2008)**.**

Este nuevo servicio, con el modelo de tiles, requiere limitar las posibilidades de configuración en la petición *HTTP*[xv] frente a lasque ofrece el servicio *WMS* y definir una especificación donde describir la comunicación entre cliente / servidor.

Para tal fin se han desarrollado especificaciones como *WMS Tile Caching* (conocida como *WMS-C*) y de sus implementaciones se destaca el proyecto *TileCache* creado en *Meta Carta Labs*, para la gestión de una solicitud de un cliente, pedir a un servicio *Web Map Service* para la respuesta y la memoria caché, así que para las solicitudes junto a los mismos datos que el propio *proxy* podría utilizar su copia en caché y evitar las peticiones al servidor Web de mapas. (Corti, 2008)

El proyecto *TileCache* está desarrollado en *Python* y funciona como proxy entre el cliente y el servidor *WMS*. Este *proxy* se encarga de atender las peticiones WMS y de almacenar las *tiles* en caché (el almacenamiento en caché sigue una estructura de carpetas denominadas *TileCache*). La ventaja de este proyecto es la generación de *tiles* de forma dinámica. Si se realiza una petición a un *tile* no almacenado, le pasa la petición al servidor *WMS* y guarda (cachea) la respuesta de este para una posterior petición.

A continuación se muestra los distintos componentes que intervienen en el proyecto.

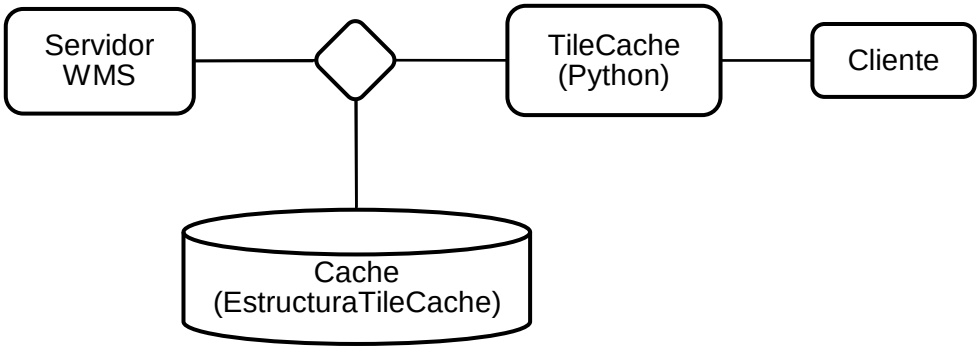

Figura 3. Estructura de *TileCache* (Ramírez, 2008).

1.4. Descripción del dominio del problema.

La Universidad de las Ciencias Informáticas ha adquirido gran experiencia en el desarrollo de proyectos contribuyendo de una forma u otra en la formación de profesionales con conocimientos en áreas de la informática. Con la creación del proyecto Aplicativos SIG del Departamento de GeoInformática, se persigue fortalecer la experiencia en la realización de Sistemas de Información Geográfica y abrir un espacio sólido en el mercado de aplicaciones de esta rama, reutilizando los componentes y funcionalidades para personalizar los productos en cualquier negocio que lo requiera. (Zaldívar, 2008)

Las soluciones creadas aspiran a tener o añadir los siguientes beneficios funcionales:

- Representación geoespacial de la información asociada a negocios específicos.
- Servicios de acceso a la información geográfica, para su consulta, análisis y visualización, mediante una interfaz de usuario sencilla y de fácil manejo que pueda ser utilizada por usuarios no especializados en tecnología SIG.
- Integración con la información ráster existente (imágenes de satélite, orto-fotos o mapas escaneados) con información vectorial.
- Publicar en Internet servicios *WMS* de *OGC* (*Open GIS Consortium*).

1.5. Situación problemática.

Aunque *MapServer* mejora la respuesta de una aplicación *Web*, todavía puede tomar tiempo para que el servidor pueda responder y visualizara cada pedido que el cliente realiza. Además, su rendimiento o capacidad de respuesta disminuye si aumenta la demanda de usuarios que debe atender.

La especificación *WMS* fue desarrollada para servir mapas cartográficos de forma interoperable y uno de sus principales objetivos es la flexibilidad. Sin embargo, debido a sus numerosos grados de libertad, los mapas necesitan generarse al instante. Dibujar un mapa es una tarea de gran carga computacional que dificulta que el servicio pueda responder adecuadamente a un número elevado de peticiones simultáneas.

La forma de lograr este objetivo no es comprar un *hardware*[xvi] muy potente para su servicio, sino simplemente convertir los datos publicados por el propio servicio, de una manera que sean más rápido de manejar.

Para afrontar este problema se pueden reducir los dominios continuos de los parámetros a un conjunto discreto de valores. La zona geográfica queda dividida en una rejilla compuesta por elementos de geometría predefinida (*tiles*) e identificables mediante índices enteros. De esta forma, es posible la actuación de mecanismos de caché entre el cliente y el servidor *WMS* (Ver Ilustración3) o incluso la prestación del servicio mediante una colección de imágenes pre-generadas. En este último caso se reduce la carga computacional al mínimo mejorando el tiempo de respuesta y la escalabilidad.

Es necesaria la implementación de un modelo de *tiles* para mejorar los servicios de mapas, de esta forma, se consigue aumentar la escalabilidad y así poder hacer frente a las peticiones de los muchos usuarios que acceden a los servicios de mapas. Actualmente se pueden implantar proyectos que resuelven esta problemática, como *TileCache* de *MetaCarta Labs*.

1.6. Soluciones existentes.

Hoy día se pueden encontrar muchas soluciones de proyectos que usen *TileCache* o procedimientos similares. A continuación se describen algunas de las recomendaciones y especificaciones actuales para el servicio de *tiles* y algunas de las implementaciones más usadas por la comunidad de SIG.

1.6.1. Especificaciones.

Se han desarrollado las especificaciones *WMS Tile Caching* (conocida como *WMS-C*), *Tile Map Service* de *OSGeo* y *Web Map Tiling Service* del *Open Geospatial Consortium*, de las cuales la primera es la más extendida.

1.6.1.1. WMS Tile Caching.

El objetivo de la propuesta *WMS Tile Caching* (*WMS-C*), es encontrar una manera de optimizar la entrega de las imágenes de mapa a través de Internet. La propuesta tiene que ofrecer idealmente algún medio por el cual los clientes puedan acceder a los *tiles* de los servidores existentes, de tal manera que las imágenes se pueden almacenar en caché en el servidor o en un lugar intermedio, o incluso ser completamente pre-generados, si se desea. (OSGeo, 2010)

Esta recomendación define una extensión del servicio WMS como mejora para aumentar la escalabilidad. Esta recomendación se basa en los siguientes criterios:

- Limitar las capas con *tiles* a configuraciones fijas (escalas, proyecciones, tamaño de *tiles*)
- Restringir los parámetros del *WMS* a los mínimos requeridos.
- La idea básica es que, a diferencia de *WMS*, dos solicitudes diferentes para un determinado *tile WMS-C* debe ser la misma solicitud *HTTP GET*[xvii]. Esto invita a una serie de limitaciones sobre las peticiones *WMS GetMap*[xviii].
- Argumentos mínimos en la cadena de consulta (es decir, sin argumentos opcionales permitidos).
- Argumento fijo cadena de consulta de pedidos y casos.
- Rango fijo de posibles dimensiones de las cajas de coordenadas, calculado a partir de los parámetros del perfil *WMS-C*.
- Tamaño de los tiles fijo en píxeles.
- Nombre fijo de la capa.
- Estilo fijo.
- Formato de salida fijo.

El almacenamiento en caché de los tiles *WMS* implica escala o de niveles de *zoom* (aumento) fijos. Normalmente, cada nivel de la escala válida sería la mitad de la escala inmediatamente superior. Sería un valor de referencia del código escrito para ayudar a los desarrolladores de averiguar cuáles tiles se deben cargar para cubrir una caja de coordenadas determinada a una escala determinada.

1.6.2. Implementaciones.

De entre las implementaciones de *WMS-C* del lado del servidor existente destacan *GeoWebCache* y *TileCache*.

1.6.2.1. GeoWebCache.

GeoWebCache es un caché para servicios *WMS* implementados en *Java*. Optimiza el renderizado de WMS, guardando los mapas renderizados para cuando sean solicitados. De manera que actúa como un *proxy* entre el cliente (*OpenLayers* o *GoogleMaps*) y el servidor (*GeoServer* u otro servidor *WMS*). En la medida en que nuevos mapas son solicitados, *GeoWeb*Cache intercepta estas llamadas y envía los *tiles* pre-renderizados, si están almacenados, o llama al servidor para que los renderice si es necesario. De esta manera, una vez que un mapa está guardado, la velocidad de renderizado se incrementa varias veces, mejorando la experiencia del usuario.

Tiene la ventaja de que al ser una aplicación *J2EE* (*Java* 2 *Plataform Enterprise Edition*) es fácilmente integrable en entornos empresariales donde se utilice la pila de aplicaciones Java. Actualmente *GeoWebCache* se distribuye tanto como aplicación Web independiente como integrada con *GeoServer*. (Salinas, y otros, 2009)

Características.

- *OGC* compatibles *Web Map Tiling Service* (*WMTS*), *Web Map Service-Caching* (*WMS-C*), *Tiled Map Service* (*TMS*).
- Capacidad de servir como un *WMS* compatible mediante la recombinación de re-muestreo y *tiles* (mosaicos) para responder a peticiones arbitrarias de imagen.
- Salida nativa para los servicios de mapas vía satélite *Google Maps* (incluyendo *Google* para móviles), *Google Earthsuper* superposiciones (vectorial y ráster), *Bing Maps* y *Yahoo Maps*.
- Buen control sobre la expiración de los *tiles* por *API REST*[xix] o notificación *GeoRSS*[xx].
- Las cuotas de disco con menos de uso frecuente y menos usados recientemente, algoritmos para gestionar con eficacia el espacio en disco.
- Maneja peticiones *WMS* con el tiempo, la elevación, los estilos alternativos y filtros.

1.7. Comparación de soluciones existentes.

Existen múltiples estudios comparativos acerca del rendimiento de servidores de mapas. Sin embargo no se han constatado estudios de rendimiento comparativo entre sistemas de caché de mapas.

La propuesta *WMS Tailing Caching* (*WMS-C*) surge ante la necesidad de disponer de una solución más escalable al actual servicio Web de mapas. Mediante la limitación de los parámetros de las peticiones a un conjunto discreto de valores, el servidor de mapas puede servir imágenes pre-generadas (*tiles*) a gran velocidad. (Martín, y otros)

A continuación se presentan algunas características comparativas entre las implementaciones existentes.

1.7.1. GeoWebCache.

- Se implementa utilizando *Java* en lugar de *Python*. Se distribuye bajo la *GNU General Public License*.
- Como una aplicación *Java*, *GeoWebCache* requiere un tiempo de ejecución de *Java* (v1.5 o v1.6) y *Java Servlet Server* (por ejemplo, *Apache Tomcat*) para ser instalado. *GeoWebCache* afirma ser una caché rápida, pero es capaz de saturar una conexión de 100 Mbps con solo un "*hardware* modesto".
- A diferencia de *TileCache*, *GeoWebCache* normalmente solo se ejecuta en una configuración independiente, lectura de archivos *WMS*, servido, por un servidor *WMS* compatible. Sin embargo, también se suministra con *GeoServer* en una forma que es más fácil de usar con *GeoServer* y tiene un menor consumo de memoria.

1.7.2. TileCache.

- Soporta dos modalidades de funcionamiento. El primero hace el *tile* usando *MapServer* o *Mapnick*[xxi] como una copia de la representación final. El segundo pude almacenar en caché *tiles WMS* descargados desde un servicio *WMS* remoto.
- Soporta dos mecanismos de caché diferentes. El mecanismo de almacenamiento *DiskCache*[xxii] almacena en memoria tiles como archivos de imagen. Además, el mecanismo *MemoryCache*[xxiii] que almacena datos en una instancia *memcache*[xxiv] o clúster[xxv.]
- Está distribuido bajo licencia *BSD*. Se requiere *Python* para ser instalado en el servidor *host*.
- Fácil configuración.

Por lo antes presentado se puede apreciar que el *TileCache* ofrece el mejor rendimiento, comodidad y facilidad de uso en las implementaciones probadas.

En general se puede lograr:

- Aumento de la capacidad de respuesta a la sociedad cubana por parte de estas entidades y a naciones interesadas en esta tecnología.
- Mayor satisfacción en los usuarios.
- Respuesta rápida en la toma de decisiones de otras áreas económicas, políticas y/o sociales.

De manera que el objetivo fundamental es lograr la representación geoespacial de la información asociada a negocios específicos, además debe permitir realizar análisis sobre dicha información, que se realiza actualmente por medio de *MapServer*.

1.8. Conclusiones parciales.

En la elaboración de este capítulo se abordaron conceptos generales que permitieron entender, así como desarrollar el marco conceptual de la investigación. Además, se realizó una caracterización de las soluciones existentes sobre los sistemas de construcción de caché, para analizar y comparar detalladamente la situación problemática antes planteada. De la cual se seleccionó la implementación *TileCache*, de manera que permita mejorar la velocidad de visualización (renderización) de la información geográfica, pues esta última constituye la fuente principal para la gestión y manipulación de datos geográficos. Una vez realizada la definición todos los aspectos teóricos-conceptuales necesarios para la realización de la investigación propuesta, puede proceder a seleccionar las tecnologías necesarias para el desarrollo del componente de *software*.

2. Capítulo II. Características del sistema.

2.1. Introducción.

En el presente capítulo se explicará mediante una descripción breve y detallada las principales tecnologías y herramientas utilizadas en la investigación, así como las características de cada una de ellas, para alcanzar un eficiente desarrollo de la aplicación. Además, se describe brevemente la propuesta de la metodología de desarrollo de *software* a utilizar para la implementación de la aplicación.

2.2. Metodologías de desarrollo de software.

Debido a que el desarrollo de *software* no es tarea fácil, existen numerosas propuestas metodológicas que inciden en distintas dimensiones del proceso de desarrollo. Por una parte tenemos aquellas propuestas más tradicionales que se centran especialmente en el control del proceso, estableciendo rigurosamente las actividades involucradas, los artefactos que se deben producir, las herramientas y notaciones que se usarán. Estas propuestas han demostrado ser efectivas y necesarias en un gran número de proyectos, pero también han presentado problemas en otros muchos. Debido a que el resultado final es un proceso de desarrollo más complejo que puede incluso limitar la propia habilidad del equipo para llevar a cabo el proyecto. (Canós, y otros, 2003)

Otra aproximación es centrarse en otras dimensiones, como por ejemplo el factor humano o el producto *software*. Esta es la filosofía de las metodologías ágiles, las cuales dan mayor valor al individuo, a la colaboración con el cliente y al desarrollo incremental del *software* con iteraciones muy cortas. Este enfoque está mostrando su efectividad en proyectos con requisitos muy cambiantes y cuando se exige reducir drásticamente los tiempos de desarrollo pero manteniendo una alta calidad. (Canós, y otros, 2003)

Las metodologías ágiles están revolucionando la manera de producir *software* y a la vez generando un amplio debate entre sus seguidores y quienes por escepticismo o convencimiento no las ven como alternativa para las metodologías tradicionales. (Canós, y otros, 2003)

La siguiente tabla recoge esquemáticamente las principales diferencias de las metodologías ágiles con respecto a las tradicionales. Estas diferencias que destinan no solo al proceso en sí, sino también al contexto del equipo así como a su organización. (Canós, y otros, 2003)

Metodología Ágil	Metodología No Ágil (Tradicional)
Pocos artefactos.	Más artefactos.
Pocos roles.	Más roles.
No existe un contrato tradicional o al menos es bastante flexible.	Existe un contrato prefijado.
El cliente es parte del equipo de desarrollo.	El cliente interactúa con el equipo de desarrollo mediante reuniones.
Grupos pequeños (menos de 10 integrantes) y trabajando en el mismo sitio.	Grupos grandes.
Menos énfasis en la arquitectura.	La arquitectura es esencial.

Tabla 1. Comparación entre las metodologías ágiles y tradicionales.

2.3. Programación extrema.

La Programación Extrema (PX), mejor conocida por su nombre en inglés *Extreme Programming* (*XP*), es una de las llamadas Metodologías ágiles de desarrollo de *software* más exitosas de los tiempos recientes formulada por Kent Beck. Consiste en una programación rápida o extrema, cuya particularidad es tener como parte del equipo, al usuario final, pues es uno de los requisitos para llegar al éxito del proyecto. *XP* se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, donde existe un alto riesgo técnico. (José H. Canós)

Tiene un conjunto de **prácticas** que lo hacen una metodología muy potente. El mayor beneficio de las prácticas se consigue con su aplicación conjunta y equilibrada puesto que se apoyan unas en otras. (José H. Canós)

- El juego de la planificación. Hay una comunicación frecuente el cliente y los programadores. El equipo técnico realiza una estimación del esfuerzo requerido para la implementación de las historias de usuario y los clientes deciden sobre el ámbito y tiempo de las entregas y de cada iteración.
- Entregas pequeñas. Producir rápidamente versiones del sistema que sean operativas, aunque no cuenten con toda la funcionalidad del sistema. Esta versión ya constituye un resultado de valor para el negocio.

- Metáfora. Es una historia compartida que describe cómo debería funcionar el sistema (conjunto de nombres que actúen como vocabulario para hablar sobre el dominio del problema, ayudando a la nomenclatura de clases y métodos del sistema).
- Diseño simple. Se debe diseñar la solución más simple que pueda funcionar y ser implementada en un momento determinado del proyecto.
- Pruebas. La producción de código está dirigida por las pruebas unitarias. Éstas son establecidas por el cliente antes de escribirse el código y son ejecutadas constantemente ante cada modificación del sistema.
- Refactorización. Es una actividad constante de reestructuración del código con el objetivo de remover duplicación de código, mejorar su legibilidad, simplificarlo y hacerlo más flexible para facilitar los posteriores cambios. Se mejora la estructura interna del código sin alterar su comportamiento externo.
- Integración continua. Cada pieza de código es integrada en el sistema una vez que esté lista. Así, el sistema puede llegar a ser integrado y construido varias veces en un mismo día.
- Estándares de programación. XP enfatiza que la comunicación de los programadores es a través del código, con lo cual es indispensable que se sigan ciertos estándares de programación para mantener el código legible.

El ciclo de vida ideal consta de cuatro **fases**.

Planificación: En esta fase el cliente establece la prioridad de cada historia de usuario y correspondientemente, los programadores realizan una estimación del esfuerzo necesario de cada una de ellas. Se toman acuerdos sobre el contenido de la primera entrega y se determina un cronograma en conjunto con el cliente. Esta fase incluye varias iteraciones sobre el sistema antes de ser entregado. El Plan de Entrega está compuesto por iteraciones de no más de tres semanas.

Diseño: La metodología *XP* sugiere que hay que conseguir diseños simples y sencillos. Hay que procurar hacerlo todo lo menos complicado posible para conseguir un diseño fácilmente entendible y se pueda implementar que a la larga costará menos tiempo y esfuerzo desarrollar. Se generan artefactos como el glosario de términos y las tarjetas CRC (Contenido, Responsabilidad, Colaboración).

Construcción: La fase de construcción requiere de las tarjetas CRC (Contenido, Responsabilidad, Colaboración), estas permiten desprenderse del método de trabajo basado en procedimientos y trabajar con una metodología basada en objetos. En esta fase se realiza el desarrollo de las

iteraciones, donde las historias de usuario se descomponen en tareas de programación o en tareas de ingeniería.

Prueba: Esta fase garantiza el correcto funcionamiento del código que se va implementando. Las pruebas se dividen en dos: en pruebas unitarias y pruebas de aceptación. Las unitarias verifican que el código correspondiente a un módulo específico se comporte de manera esperada.

Beneficios y ventajas de la Programación Extrema.

- Programación organizada.
- Menor taza de errores.
- Satisfacción del programador.
- El cliente tiene el control sobre las prioridades.
- Se hacen pruebas continuas durante el proyecto.
- La XP es mejor utilizada en la implementación de nuevas tecnologías donde los requerimientos cambian rápidamente.

Análisis de la situación y selección de la metodología.

En el proyecto (tesis) se cuenta con, un estudiante (tesista) y un profesor (tutor). El trabajo consiste en implementar un sistema para la visualización y construcción de caché de mapas basado en tecnologías ágiles. Se selecciona la metodología *Extreme Programming* para un mejor desarrollo de la aplicación, debido a que se adapta en gran medida tanto al tipo de proyecto, las condiciones de trabajo y tiempo de entrega del mismo. Además, los requisitos cambian frecuentemente y el sistema debe adaptarse y ampliar sus funcionalidades de forma rápida. Igualmente permite un control constante de las tareas para obtener resultados en cortos plazos de tiempo. De manera que se logre la simplicidad en las soluciones implementadas y el coraje para enfrentar los cambios. En resumen, con XP no se implementa nada que no se necesite, es decir, se minimiza la documentación y se agiliza la implementación.

2.4. Arquitectura de software.

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. (Guglielmetti, 2004)

Algunos objetivos dentro de un esquema de Arquitectura de *Software* pueden ser: el *software* debe ser sostenible, esto es, fácilmente analizable, modificable, corregible; también puede ser un objetivo el nivel de interacción con otros sistemas informáticos, o su escalabilidad. (Guglielmetti, 2004)

2.4.1. Arquitectura cliente/servidor.

En el mundo de *TCP/IP* las comunicaciones entre computadoras se rigen básicamente por lo que se llama modelo Cliente-Servidor, éste es un modelo que intenta proveer usabilidad, flexibilidad, interoperabilidad y escalabilidad en las comunicaciones. Desde el punto de vista funcional, se puede definir la arquitectura Cliente/Servidor como una arquitectura distribuida que permite a los usuarios finales obtener acceso a la información en forma transparente aún en entornos multiplataforma. Se trata pues, de la arquitectura más extendida en la realización de Sistemas Distribuidos. (Mariel, y otros, 2004)

El **cliente** es el proceso que permite al usuario formular los requerimientos y pasarlos al servidor.

Las funciones que lleva a cabo el proceso cliente se resumen en los siguientes puntos:

- Administrar la interfaz de usuario.
- Interactuar con el usuario.
- Procesar la lógica de la aplicación y hacer validaciones locales.
- Generar requerimientos de bases de datos.
- Recibir resultados del servidor.
- Formatear resultados.

El **servidor** es el proceso encargado de atender a múltiples clientes que hacen peticiones de algún recurso administrado por él. Normalmente maneja todas las funciones relacionadas con la mayoría de las reglas del negocio y los recursos de datos.

Las funciones que lleva a cabo el proceso servidor se resumen en los siguientes puntos:

- Aceptar los requerimientos de bases de datos que hacen los clientes.
- Procesar requerimientos de bases de datos.
- Formatear datos para trasmitirlos a los clientes.
- Procesar la lógica de la aplicación y realizar validaciones a nivel de bases de datos.

Capítulo II
Características del sistema

Un sistema Cliente/Servidor es un Sistema de Información distribuido basado en las siguientes características:

- Servicio: unidad básica de diseño. El servidor los proporciona y el cliente los utiliza.
- Recursos compartidos: Muchos clientes utilizan los mismos servidores y comparten tanto recursos lógicos como físicos.
- Protocolos asimétricos: Los clientes inician "conversaciones". Los servidores esperan su establecimiento pasivamente.
- Transparencia de localización física de los servidores y clientes: El cliente no tiene por qué saber dónde se encuentra situado el recurso que desea utilizar.
- Independencia de la plataforma *Hardware* y *Software* que se emplee.
- Sistemas débilmente acoplados. Interacción basada en envío de mensajes.
- Encapsulamiento de servicios. Los detalles de la implementación de un servicio son transparentes al cliente.
- Escalabilidad horizontal (añadir clientes) y vertical (ampliar potencia de los servidores).
- Integridad: Datos y programas centralizados en servidores facilitan su integridad y mantenimiento.

Esta arquitectura permite distribuir físicamente los procesos y los datos en forma más eficiente lo que en computación distribuida afecta directamente el tráfico de la red, reduciéndolo grandemente.

Entre las principales ventajas del esquema Cliente/Servidor están:

- Uno de los aspectos que más ha promovido el uso de sistemas Cliente/Servidor, es la existencia de plataformas de hardware cada vez más baratas.
- El esquema Cliente/Servidor facilita la integración entre sistemas diferentes y comparte información permitiendo, por ejemplo que las máquinas ya existentes puedan ser utilizadas pero utilizando interfaces más amigables al usuario.
- Al favorecer el uso de interfaces gráficas interactivas, los sistemas construidos bajo este esquema tienen mayor interacción y más intuitiva con el usuario.
- Una ventaja adicional del uso del esquema Cliente/Servidor es que es más rápido el mantenimiento y el desarrollo de aplicaciones, pues se pueden emplear las herramientas existentes.
- La estructura inherentemente modular facilita además la integración de nuevas tecnologías y el crecimiento de la infraestructura computacional, favoreciendo así la escalabilidad de las soluciones.

Por último, el esquema Cliente/Servidor contribuye además, a proporcionar, a los diferentes departamentos de una organización, soluciones locales, pero permitiendo la integración de la información relevante a nivel global.

2.5. HTML 5.

HTML es una implementación del estándar *SGML* (*Standard* Gen*e*ralized *Markup Language*), estándar internacional para la definición de texto electrónico independiente de dispositivos, sistemas y aplicaciones. (Martín, 2001)

Características:

- Permite crear lenguajes de codificación descriptivos.
- Define una estructura de documentos jerárquica, con elementos y componentes interconectados.
- Proporciona una especificación formal completa del documento.
- No tiene un conjunto implícito de convenciones de señalización.
- Soporta, por tanto, un conjunto flexible de juegos de etiquetas.
- Los documentos generados por él son legibles.

Por último, *HTML* es un Metalenguaje para definir lenguajes de diseño descriptivos; proporciona un medio de codificar documentos hipertexto cuyo destino sea el intercambio directo entre sistemas o aplicaciones.

2.6. Lenguajes de programación.

Un lenguaje de programación es aquel elemento dentro de la informática que nos permite crear programas mediante un conjunto de instrucciones, operadores y reglas de sintaxis; que pone a disposición del programador para que este pueda comunicarse con los dispositivos hardware y software existentes. (Definición.org, 2012)

2.6.1. PHP 5.3.

PHP (*Hypertext Preprocessor*), es un lenguaje de programación interpretado de alto nivel, embebido en páginas *HTML* y ejecutado en el servidor, diseñado originalmente para la creación de páginas Web dinámicas. (Henst, 2001)

Características de PHP.

- Es un lenguaje multiplataforma y libre, por lo que se presenta como una alternativa de fácil acceso para todos.
- El código se actualiza continuamente con mejoras y extensiones de lenguaje para ampliar las capacidades de PHP.
- Capacidad de conexión con una gran cantidad de bases de datos como *MySQL*, *PostgreSQL*, entre otras.
- Permite aplicar técnicas de programación orientada a objetos.
- Orientado completamente al desarrollo de aplicaciones Web dinámicas con acceso a información almacenada en una base de datos.
- Permite la integración con varias bibliotecas externas, generar documentos en *PDF* y analizar código *XML*[xxvi]. (Jason Gilmore, 2006.)

Fundamento de *PHP*.

PHP está orientado al desarrollo *Web*, es de gran velocidad por lo que no requiere de muchos recursos del sistema y además se integra perfectamente con muchos servidores y principalmente Apache, el cual fue seleccionado como servidor a utilizar. Es libre y está disponible bajo la licencia *GPL*, es multiplataforma por lo que no tendrá ningún inconveniente al usarlo en cualquier computadora de la Universidad. Se caracteriza por la simplicidad de su código y por la amplia documentación que brinda.

2.6.1.1. PHP mapscript.

Es un módulo *PHP* que habilita las funcionalidades de *MapScript* de *MapServer* en una librería de PHP que se carga dinámicamente. En términos simples, este módulo permite utilizar este poderoso lenguaje para crear y modificar de forma dinámica las imágenes de los mapas de MapServer. (Morissette, 2011)

2.6.2. JavaScript.

JavaScript es un lenguaje de scripts desarrollado por *Netscape* para incrementar las funcionalidades del lenguaje *HTML*, el cual permite a los desarrolladores crear acciones en sus páginas *Web*. (Pérez, 2008)

Características de *JavaScript*.

- Al ser un lenguaje interpretado, no requiere compilación. El navegador del usuario se encarga de interpretar las sentencias *JavaScript* contenidas en una página *HTML* y ejecutarlas adecuadamente.

- Es dinámico por lo que se puede cambiar totalmente el aspecto de la página a gusto del usuario.
- Es un lenguaje orientado a eventos.
- Permite la programación orientado a objetos. El modelo de objetos de *JavaScript* está reducido y simplificado, pero incluye los elementos necesarios para que los scripts puedan acceder a la información de una página y puedan actuar sobre la interfaz del navegador.
- Es soportado por la gran mayoría de los navegadores como *Internet Explorer*, *Chrome*, *Netscape*, *Opera*, *Mozilla Firefox*, entre otros.
- Soporta cuatro tipos de datos, pero no es necesario declarar el tipo de las variables, argumentos de funciones ni valores de retorno de las funciones.

En resumen *JavaScript*, es una de las múltiples maneras que han surgido para extender las capacidades del lenguaje *HTML*. Al ser la más sencilla, es por el momento la más extendida.

2.7. Entornos de desarrollo.

Un entorno de desarrollo integrado (*IDE* por sus siglas en inglés, *Integrated Development Environment*), es un programa informático compuesto por un conjunto de herramientas de programación. Puede dedicarse en exclusiva a un sólo lenguaje de programación o bien, poder utilizarse para varios (Martínez, 2009).

Las herramientas que normalmente componen un entorno de desarrollo integrado son las siguientes: un editor de texto, un compilador, un intérprete, unas herramientas para la automatización, un depurador, un sistema de ayuda para la construcción de interfaces gráficas de usuario y, opcionalmente, un sistema de control de versiones (Martínez, 2009).

Los *IDEs* proveen un marco de trabajo amigable para la mayoría de los lenguajes de programación tales como *C++*, *Python*, *Java*, C#, *Delphi*, *Visual Basic*, etc. En algunos lenguajes, un *IDE* puede funcionar como un sistema en tiempo de ejecución, en donde se permite utilizar el lenguaje de programación en forma interactiva, sin necesidad de trabajo orientado a archivos de texto (Martínez, 2009).

2.7.1. NetBeans 7.1.

NetBeans IDE es un entorno de desarrollo visual para aplicaciones programadas a partir del uso del lenguaje de programación *Java*, de modo que puede ejecutarse en cualquier ambiente que ejecute *Java* (NetBeans, 2011).

Es un producto de código abierto, con todos los beneficios del *software* disponible en forma gratuita, el cual ha sido examinado por una comunidad de desarrolladores. Es una herramienta para que los programadores puedan escribir, compilar, depurar y ejecutar programas (NetBeans, 2011).

Aparte de la filosofía de distribución y desarrollo que respalda a Net*B*eans, el *IDE* ofrece a los desarrolladores numerosas ventajas, en la creación de nuevas aplicaciones multiplataforma. En una era en la cual la arquitectura orientada al servicio (*SOA*) requiere servicios con cierta relación que manejen procesos específicos del negocio, *NetBeans* satisface los requisitos con un conjunto de herramientas independientes de la plataforma, modulares y orientadas al objeto.

La automatización de los requerimientos de diseño demuestra ser particularmente importante en el diseño de aplicaciones para de Arquitectura Orientada a Servicios (*SOA*) donde los desarrolladores trabajan generalmente con múltiples tecnologías y protocolos. Este *IDE* de *Java* es de mucha confianza para los desarrolladores y sumamente útil por su manual de instrucciones.

Además, *NetBeans* en la versión 7.1 incluye un conjunto de herramientas personalizadas para *PHP*, como resaltado sintáctico y semántico, plantillas, completado automático de código, consejos y soluciones rápidas, documentación emergente, formateo de código y plegado, etc. (NetBeans, 2012)

2.8. Sistema gestor de base de datos.

Un Sistema Gestor de Bases de Datos (SGBD) es una colección de programas cuyo objetivo es servir de interfaz entre la base de datos, el usuario y las aplicaciones. Se compone de un lenguaje de definición de datos, de un lenguaje de manipulación de datos y de un lenguaje de consulta. Un SGBD permite definir los datos a distintos niveles de abstracción y manipularlos, garantizando la seguridad e integridad de los mismos. Un SGBD permite crear y mantener una base de datos, asegurando su integridad, confidencialidad y seguridad.

2.8.1. PostgresSQL 8.4.

PostgreSQL es un sistema gestor de bases de datos relacionales orientadas a objetos, desarrollado en la Universidad de California en el Departamento de Ciencias de la Computación de Berkeley. Distribuido bajo licencia BSD y con su código fuente disponible libremente. Es compatible con una gran parte del estándar SQL y ofrece muchas características modernas (PostgreSQL, 2012):

- Consultas complejas.
- Llaves foráneas.

- Disparadores.
- Vistas.
- Integridad de las transacciones.
- Control de concurrencia multiversión.

Además, posee opciones de seguridad flexibles en un extenso conjunto de protocolos de seguridad y opciones de configuración como rasgos internos, que ayudan a tener el control sobre quiénes y que accede a los datos dentro de la base de datos.

2.9. Apache 2.2.

Apache es un servidor *Web* potente y flexible que funciona en distintas plataformas y entornos, estas hacen que a menudo sean necesarias diferentes características o funcionalidades, o que una misma característica o funcionalidad se realice de diferente manera para obtener un mayor resultado. El diseño modular de Apache permite a los administradores de sitios *Web* elegir que características se van a incluir en el servidor al seleccionar los módulos que se van a cargar, ya sea al compilar o al ejecutar el servidor (Lomeli, 2007). Entre sus características destacan:

- Multiplataforma
- Es un servidor de web conforme al protocolo *HTTP*
- Modular: Puede ser adaptado a diferentes entornos y necesidades, con los diferentes módulos de apoyo que proporciona, y con la *API* de programación de módulos, para el desarrollo de módulos específicos.
- Basado en hebras a partir de la versión 2.0.
- Incentiva la realimentación de los usuarios, obteniendo nuevas ideas, informes de fallos y parches para la solución de los mismos.
- Se desarrolla de forma abierta.

Además, gracias a ser modular se han desarrollado diversas extensiones entre las que destaca PHP, un lenguaje de programación del lado del servidor.

2.10. OpenLayers 2.11.

OpenLayers es una librería *JavaScript* puro para la visualización de los datos del mapa en la mayoría de navegadores modernos, con ningún servidor de las dependencias lado. *OpenLayers* implementa un (aún en desarrollo) *API* de *JavaScript* para la construcción deW*web* ricas aplicaciones basadas

geográfica, similar a la de *Google Maps* y *MSN Virtual Earth API*, con una diferencia importante *OpenLayers* es el *Software* Libre, desarrollado por y para la comunidad *Open Source software*. (OpenLayers, 2012)

Ventajas.

- *OpenLayers* no requiere instalación)
- Menor procesamiento en el servidor.
- Puede ampliar fácilmente el código para su aplicación en particular.
- Puede utilizar múltiples Servidores de datos.

OpenLayers permite poner un mapa dinámico en cualquier página *Web*. Puede mostrar bloques de mapas y marcadores desde cualquier fuente, fue desarrollado inicialmente por *MetaCarta* y se lo dio al público para promover el uso de la información geográfica de todo tipo. *OpenLayers* es totalmente gratuito, de código abierto de *JavaScript* (*Open Source*), liberado bajo una licencia tipo *BSD*.

2.11. Framework a utilizar: ExtJS 4.0.2a.

De acuerdo a la definición de la página *Web ExtJS* es una librería *Javascript* desarrollado por Sencha que permite construir aplicaciones complejas en Internet. Esta librería incluye:

- Componentes de interfaz de usuario personalizable y de alto rendimiento.
- Modelo de componentes extensibles.
- Un *API* fácil de usar.
- Licencias *Open Source* y comerciales.

Una de las grandes ventajas de utilizar *ExtJS* es que nos permite crear aplicaciones complejas utilizando componentes predefinidos así como un manejador de diseños, gracias a esto provee una experiencia consistente sobre cualquier navegador.

Usar un motor de renderizado como *ExtJS* nos permite tener además estos beneficios:

- Existe un balance entre Cliente – Servidor. La carga de procesamiento se distribuye, permitiendo que el servidor, al tener menor carga, pueda manejar más clientes al mismo tiempo.
- Comunicación asíncrona. En este tipo de aplicación el motor de renderizado puede comunicarse con el servidor sin necesidad de estar sujeta a un clic o una acción del usuario, dándole la libertad de cargar información sin que el cliente se dé cuenta.

- Eficiencia de la red. El tráfico de red puede disminuir al permitir que la aplicación elija que información desea transmitir al servidor y viceversa, sin embargo la aplicación que haga uso de la pre-carga de datos puede que revierta este beneficio por el incremento del tráfico.

Además, la ventana flotante que provee ExtJS es excelente por la forma en la que funciona. Al moverla o redimensionarla solo se dibujan los bordes haciendo que el movimiento sea fluido lo cual le da una ventaja tremenda frente a otros. (Sencha, 2012)

2.12. Conclusiones parciales.

El desarrollo de las tecnologías de información en la *Web* ha permitido el surgimiento de numerosas técnicas y herramientas que posibilitan su avance, por lo que constituye de gran importancia la correcta selección de estas para desarrollar cualquier sistema. A partir del estudio realizado en el presente capítulo se determinaron las tecnologías más adecuadas para la implementación de un sistema para la visualización y construcción de caché de mapas. Dichas tecnologías son de las más actuales y usadas en este campo, que con su correcto uso se podrá construir un software que cumplan con la calidad requerida. Una vez seleccionada las herramientas y tecnologías a utilizar, se puede proceder a realizar el análisis y diseño de la solución propuesta.

3. Capítulo III. Análisis y diseño de la solución propuesta.

3.1. Introducción.

Para el desarrollo exitoso de la aplicación es necesario tener conocimiento de los procesos que ocurren en la actualidad acerca de la visualización de información geográfica. En este capítulo se desarrollan las dos primeras fases del ciclo de vida de la metodología *XP*: planificación y diseño. En el que se crean las historias de usuarios para lograr un mejor entendimiento acerca del *software* que se desarrollará y se describen las tarjetas CRC[1] para un mejor entendimiento del sistema.

3.2. Propuesta del sistema.

Se propone la realización de un sistema que permita brindar un servicio Web de mapas base para la visualización de mapas en los proyectos que se desarrollan, mediante el uso de la tecnología de visualización de mapas basado en el modelo de tiles (*TileCache*).

3.2.1. Personas relacionadas con el sistema.

Personas relacionadas con el sistema	Justificación
Usuario	Es la persona que interactúa con el sistema.

Tabla 2. Personas relacionadas con el sistema.

3.3. Fase de Planificación.

La planificación es la fase inicial en cualquier proyecto *XP*. A partir de este punto se empieza a interactuar con el cliente y con el equipo de desarrollo para establecer los requisitos del sistema. Se tienen en cuenta seis elementos: historias de usuario, plan de entregas, velocidad de proyecto, iteraciones, rotaciones y reuniones. (Joskowicz, y otros, 2008)

3.3.1. Historias de usuarios.

Son la técnica utilizada para especificar los requisitos del *software*. Se trata de tarjetas en las cuales el cliente describe brevemente las características que el sistema debe poseer, sean requisitos funcionales o no funcionales. El tratamiento de las historias de usuario es muy dinámico y flexible. Cada historia de usuario es lo suficientemente comprensible y delimitada para que los programadores puedan implementarla en semanas. (Canós, y otros, 2003)

[1] Cargo o Clase, Responsabilidad y Colaboración.

Las historias de usuario tienen el mismo propósito que los casos de uso. (Escribano, 2002)

Las historias de usuario son similares al empleo de escenarios, con la excepción de que no se limitan a la descripción de la interfaz de usuario. También conducirán el proceso de creación de las pruebas de aceptación (empleadas para verificar que las historias de usuario han sido implementadas correctamente).

Existen diferencias entre estas y la tradicional especificación de requisitos. La principal diferencia es el nivel de detalle. Las historias de usuario solamente proporcionaran los detalles sobre la estimación del riesgo y cuánto tiempo conllevará la implementación de dicha historia de usuario.

Beck en su libro (Beck, 1999) presenta un ejemplo de ficha en la cual pueden reconocerse algunos de sus contenidos. A efectos de planificación, las historias pueden ser de una a tres semanas de tiempo de programación (para no superar el tamaño de una iteración). Las historias de usuario son descompuestas en tareas de programación y asignadas a los programadores para ser implementadas durante una iteración.

3.3.2. Relación de historias de usuarios.

Como parte del proceso de trabajo dentro de la fase de exploración se identificaron las siguientes Historias de Usuarios:

<table>
<tr><th colspan="3">Historia de Usuario</th></tr>
<tr><td>Numero: 1.</td><td colspan="2">Nombre de Historia: Adicionar mapa.</td></tr>
<tr><td colspan="3">Usuario: Cliente.</td></tr>
<tr><td colspan="2">Prioridad en el Negocio: Alta.</td><td>Riesgo en Desarrollo: Alto.</td></tr>
<tr><td colspan="2">Puntos de Estimación: 3.</td><td>Iteración Asignada: 1.</td></tr>
<tr><td colspan="3">Descripción: Permite al cliente adicionar un nuevo mapa que posteriormente el sistema brindará como un servicio mediante la tecnología TileCache. Para ello el cliente deberá introducir el nombre del nuevo mapa, subir al servidor el archivo “.map” y la simbología en un fichero compactado correspondiente al mismo. Por último, configurar el TileCache de acuerdo a sus necesidades y la selección de las capas que brindara el posterior servicio.</td></tr>
</table>

Observaciones: El mapa puede debe ser un fichero de tipo "*.map*", el cual no debe poseer direcciones a capas fuera del ámbito del mismo. El sistema genera de forma automática un sistema de carpetas para el almacenamiento de los mapas y sus características.

Tabla 3. Historia de Usuario #1. Adicionar mapa.

Historia de Usuario	
Numero: 2.	**Nombre de Historia**: Editar mapa.
Usuario: Cliente.	
Prioridad en el Negocio: Alta.	**Riesgo en Desarrollo**: Alta.
Puntos de Estimación: 3.	**Iteración Asignada**: 2.
Descripción: Una vez cargado el mapa el cliente tiene la posibilidad de re-editar las configuraciones de *TileCache* que se ajusten a su necesidad, el nombre del mapa y las capas a brindar por el servicio.	
Observaciones: Una vez modificado el mapa se borrara la caché generada por el mismo. En caso de no realizar configuraciones particulares el sistema guarda las necesarias por defecto.	

Tabla 4. Historia de Usuario #2. Editar mapa.

Historia de Usuario	
Numero: 3.	**Nombre de Historia**: Eliminar mapa.
Usuario: Cliente.	
Prioridad en el Negocio: Alta.	**Riesgo en Desarrollo**: Bajo.
Puntos de Estimación: 1.	**Iteración Asignada**: 3.
Descripción: El cliente selecciona el mapa que desea eliminar.	

Observaciones: Un vez eliminado el mapa, de igual manera se borrará toda información referente al mismo.

Tabla 5. Historia de Usuario #3. Eliminar mapa.

Historia de Usuario	
Numero: 4.	**Nombre de Historia**: Brindar *URL* del servicio a consumir.
Usuario: Cliente.	
Prioridad en el Negocio: Alta.	**Riesgo en Desarrollo**: Bajo.
Puntos de Estimación: 2.	**Iteración Asignada**: 3.
Descripción: Una vez introducidos los datos correspondientes al mapa que desea consumir por el servicio. El sistema brinda la *URL* correspondiente al mapa introducido por el cliente.	
Observaciones: El cliente puede ver otros datos relacionado con el servicio.	

Tabla 6. Historia de Usuario #4. Mostrar *URL* del servicio a consumir.

Historia de Usuario	
Numero: 5.	**Nombre de Historia**: Visor de mapas.
Usuario: Cliente.	
Prioridad en el Negocio: Alta.	**Riesgo en Desarrollo**: Alta.
Puntos de Estimación: 3.	**Iteración Asignada**: 4.
Descripción: El sistema muestra en un visor generado por *OpenLayers* el mapa que seleccione el cliente, consumiendo el servicio de tiles.	

Observaciones: Una vez que el cliente adiciona el mapa, el sistema brinda la posibilidad de mostrar un prototipo de interfaz correspondiente a las configuraciones realizadas por el cliente

Tabla 7. Historia de Usuario #5. Visor de mapas.

3.3.3. Estimación de esfuerzos.

Para lograr un desarrollo eficiente y satisfactorio, se realiza una estimación de esfuerzos para cada una de las Historias de Usuarios identificadas en el proceso de planificación, llegando a los resultados que siguientes.

Historias de Usuarios	Puntos de Estimación
Adicionar mapa.	3
Editar mapa.	3
Eliminar mapa.	1
Brindar *URL* del servicio a consumir.	1
Visor de mapas.	2

Tabla 8. Estimación de esfuerzos.

3.3.4. Plan de iteraciones.

En la metodología XP, la creación del sistema se divide en tres etapas. Normalmente, los proyectos suelen tener más de tres etapas, cada una de las cuales toma el nombre de iteración. Es por ello que esta metodología se dice que es iterativa.

La duración ideal de una iteración está entre una y tres semanas. Para cada una de las iteraciones se establecen un conjunto de historias que se van a llevar a cabo y un módulo, dando como resultado de cada iteración la entrega de dicho módulo, habiendo superado éste las pruebas de aceptación establecidas por el cliente de forma que verifique los requisitos.

Después de haber definido las HU y estimado el esfuerzo propuesto para la realización de cada una de ellas, se tomó la decisión de realizar el sistema en 4 iteraciones.

Iteración #1. Se implementará la Historia de Usuario “Adicionar mapa”. Al final se contará con un prototipo del sistema, el cual será presentado al cliente con el objetivo de lograr una retroalimentación para el grupo de trabajo.

Iteración #2. Se implementará la Historia de Usuario “Editar mapa” de alta prioridad. Una vez concluida el usuario podrá modificar las configuraciones del mapa adicionado.

Iteración #3. Se implementaran las funcionalidades de las historias de usuario “Eliminar mapa” y “Brindar *URL* del servicio a consumir”. Finalizada la misma el usuario podrá realizar las funcionalidades descritas en dichas historias de usuario.

Iteración #4. Se implementara la Historia de Usuario “Visor de mapas” de alta prioridad. Concluida la misma el sistema será capaz de mostrar en un visor sencillo el mapa que el usuario desea consumir por el servicio.

3.3.5. Plan de duración de las Iteraciones.

El plan de iteración consiste en seleccionar las historias de usuario que, según el plan de entregas, corresponderían a esta iteración. También se eligen qué pruebas de aceptación fallidas se corregirán.

Para lograr una mayor organización del trabajo se crea un plan de duración de las iteraciones; el mismo tiene como objetivo mostrar la duración de cada iteración así como el orden en que serán implementadas las historias de usuarios en cada una de ellas. Este plan tiene como finalidad mostrar la duración de cada iteración, así como el orden en que serán implementadas las HU en cada una de ellas.

Iteración	Historias de Usuarios	Duración Total de la Iteraciones
1	Adicionar mapa.	3
2	Editar mapa.	3
3	Eliminar mapa.	2
	Brindar *URL* del servicio a consumir.	

4	Visor de mapas.	2

Tabla 9. Plan de duración de las iteraciones.

3.3.6. Plan de entregas.

Se realiza un cronograma de entregas donde se establece qué historias de usuario serán agrupadas para conformar una entrega y el orden de las mismas. Normalmente, el cliente ordenará y agrupará de acuerdo a sus prioridades las historias de usuario. El cronograma de entregas se realiza en base a las estimaciones de tiempos de desarrollo realizadas por los desarrolladores.

A continuación se muestra el plan de entregas desarrollado para dar solución al problema planteado. Para desarrollar el mismo se tuvo en cuenta los puntos de estimación para obtener un resultado final.

Número de la Iteración	Duración de la Iteración	Fecha de Inicio	Fecha Final
1	3	05/03/2012	27/03/2012
2	3	28/03/2012	18/04/2012
3	2	19/04/2012	04/05/2012
4	2	05/05/2012	19/05/2012

Tabla 10. Plan de entrega.

3.4. Fase de diseño.

La Metodología *XP* plantea que la implementación de un *software* debe realizarse de forma iterativa, obteniendo al culminar cada iteración un producto funcional que debe ser probado y mostrado al cliente para incrementar la visión de los desarrolladores con la opinión de éste.

Esta metodología para el diseño de sus aplicaciones, no requiere la representación del sistema mediante diagramas de clase utilizando notación *UML*, sino que utiliza otras técnicas como las llamadas tarjetas CRC (Contenido, Responsabilidad y Colaboración). Sin embargo la utilización de estos diagramas puede aplicarse siempre y cuando influya en el mejoramiento de la comunicación entre el equipo de desarrollo y se enfoquen en la información importante. (Joskowicz, y otros, 2008)

3.4.1. Tarjetas CRC.

Para poder diseñar el sistema como un equipo se debe cumplir con tres principios: Cargo o Clase, Responsabilidad y Colaboración (CRC). Las tarjetas CRC permiten desprenderse del método de trabajo basado en procedimientos y trabajar con una metodología basada en objetos. Las tarjetas CRC permiten que el equipo completo contribuya en la tarea del diseño. (Corbea, y otros, 2007)

Permiten al programador centrarse y apreciar el desarrollo orientado a objetos olvidándose de los malos hábitos de la programación procedural clásica. Debido a la facilidad de su uso y entendimiento, se decidió utilizarlas para diseñar la aplicación que se desea desarrollar.

Tarjeta CRC	
Clase: layer.	
Responsabilidades: • __construct. • add_named_layer. • add_bbox. • add_resolution.	**Colaboraciones:**

Tabla 11. Tarjeta CRC – layer.

Tarjeta CRC	
Clase: cfg_cache.	
Responsabilidades: • __construct. • get_type. • get_base.	**Colaboraciones:**

Tabla 12. Tarjeta CRC – cfg_cache.

Tarjeta CRC	
Clase: ms_layer.	
Responsabilidades:	**Colaboraciones:**

• __construct. • get_jMap. • get_mapfile. • get_simbol_tmp. • set_simbol_name.	• layer.

Tabla 13. Tarjeta CRC – ms_layer.

Tarjeta CRC	
Clase: tilecache_cfg.	
Responsabilidades: • __construct. • add_layer.	**Colaboraciones:** • layer. • cfg_cache.

Tabla 14. Tarjeta CRC – tilecache_cfg.

Tarjeta CRC	
Clase: template.	
Responsabilidades: • __construct. • get_content. • write_out_file. • delete_ByName. • get_extent. • get_extension.	**Colaboraciones:** • layer.

Tabla 15. Tarjeta CRC – template.

Tarjeta CRC	
Clase: cfg_control.	
Responsabilidades: • __construct. • get_matrix.	**Colaboraciones:** • tilecache_cfg.

• get_cfg_file. • get_layerByName. • is_layer. • load_cfg. • add_layer. • layer_to_array. • delete_layer. • write_cfg_out_file. • load_matrix. • write_cfg_file	

Tabla 16. Tarjeta CRC – cfg_control.

Tarjeta CRC	
Clase: layer_control.	
Responsabilidades: • __construct. • get_manager. • insert_layer.	**Colaboraciones:** • layer. • layer_manager.

Tabla 17. Tarjeta CRC – layer-control.

Tarjeta CRC	
Clase: ms_control.	
Responsabilidades: • __construct. • get_manager. • is_dir. • map_type_validation. • get_layer_by_index. • compressed_type_validation. • copy_files. • extract_simbol.	**Colaboraciones:** • ms_layer. • cfg_control. • Template. • maps_manager.

• search_files. • check_simbol_file. • insert_map. • insert_layers. • modify_mapByName. • save_all. • save_new_name. • get_mapByName. • get_layerByname. • delete_map. • delete_dir. • loadAll_ByName. • include_projection. • include_resolutions. • include_bbox. • include_scale. • calculate_url. • url_openlayer. • change_url. • get_levels	

Tabla 18. Tarjeta CRC – ms_control.

Tarjeta CRC	
Clase: layer_manager.	
Responsabilidades: • insert_layer. • delete_layer	**Colaboraciones:**

Tabla 19. Tarjeta CRC – layer_manager.

Tarjeta CRC
Clase: maps_manager.

Responsabilidades:	**Colaboraciones:**
• __construct. • insert_map. • get_map_names. • get_idByName. • get_mapByName. • get_map_nameById. • get_pathByname. • delete_map. • type_ByName. • get_allByName.	

Tabla 20. Tarjeta CRC – maps_manager.

3.5. Requisitos no funcionales del sistema.

Los requisitos no funcionales son propiedades o cualidades que el producto debe tener. Estas propiedades son las características que hacen al producto atractivo, usable, rápido o confiable.

Usabilidad.

- El sistema podrá ser usado por personas con conocimientos básicos en el manejo de computadoras.

Confiabilidad.

- La herramienta de implementación a utilizar debe tener soporte para recuperación ante fallos y errores.

Apariencia o interfaz externa.

- Debe brindar una interfaz amigable, interactiva, intuitiva y de fácil comprensión para el usuario, facilitando en todo momento la interacción de este con el sistema.

Rendimiento.

- El tiempo de respuesta para visualizar el mapa en la pantalla será mínimo, para procesar, actualizar y recuperar información debe operar con igual rendimiento, en dependencia de la cantidad de información.

Restricciones de diseño

- El diseño debe ser sencillo, con pocas entradas, donde no es necesario mucho entrenamiento para utilizar el sistema. Se logrará un producto altamente configurable y extensible.

Hardware

- Para las computadoras clientes:
  - Se requiere tarjeta de red.
  - Al menos 128 MB de memoria *RAM*.
  - Procesador 512 MHz como mínimo.
- Para los servidores:
  - Se requiere tarjeta de red.
  - El Servidor de Mapas tenga como mínimo 2GB de *RAM* y 80GB de disco duro.
  - Procesador 3 GHz como mínimo.

Clientes

- Para las computadoras clientes:
  - Navegador *Web*: *Mozilla FireFox*, *Google Chrome*, *Opera*, *Safari*.
- Para los servidores:
  - *PHP Script* Language 5.3.
  - Servidor *Web*: *Apache Server* 2.2.4.
  - Módulo de *Python* para *Apache Server*.
  - Librería *Mapscript* para *Python* y *PHP*.

Requisitos de Licencia.

- La arquitectura de acuerdo a los tipos de licencias de las herramientas que se van a utilizar es legalmente de modelo libre, permitiendo la utilización, modificación y distribución de las mismas por terceros sin necesidad de obtener la autorización de sus respectivos titulares.

Requisitos Legales, de Derecho de Autor y otros.

- La mayoría de las herramientas de desarrollo son libres y del resto, las licencias están avaladas.

Finalizada la descripción de los requisitos no funcionales se procede a concluir el siguiente capítulo.

3.6. Conclusiones parciales.

Con el uso de la metodología de software *XP* se construye un software con un diseño simple pero adecuado siempre a la funcionalidad actual. Lo que permite al sistema prepararse para cualquier situación, por lo que, un buen diseño es esencial. De esta manera se construyeron los artefactos necesarios que proporcionarán la forma en la que se comportará el sistema. Además permitan la implementación del mismo y aumentar la velocidad de desarrollo aplicando la refactorización propuesta por *XP*. Finalizado el diseño se puede proceder a construir el sistema.

4. Capítulo III. Implementación y prueba.

4.1. Introducción.

En presente capitulo se detallan las iteraciones llevadas a cabo durante la etapa de construcción de la aplicación, exponiéndose las tareas de programación o ingeniería generadas por cada historia de usuario. Por último, se realizan las pruebas correspondientes donde la sólo se podrá liberar una nueva versión si ésta ha superado el cien por ciento de la totalidad de ella. De no ser así, se buscará el error y se solucionará.

4.2. Fase de desarrollo.

En la fase de desarrollo se realiza de forma paralela con el diseño. Destacan la disponibilidad del cliente, la unidad de pruebas, la programación por parejas y la integración.

4.2.1. Desarrollo de las iteraciones.

Durante el transcurso de las iteraciones se lleva a cabo una revisión del plan de iteraciones y se modifica en caso de ser necesario. Como parte de este plan, se descomponen las Historias de Usuarios en tareas de programación o ingeniería, las mismas pueden ser escritas en lenguaje técnico debido a que son para uso estricto de los programadores, donde el usuario no tiene que necesariamente comprenderlas. (Joskowicz, y otros, 2008)

A continuación se definen cada una de las iteraciones.

4.2.1.1. Iteración #1.

Historias de Usuarios	Tiempo de Implementación	
	Estimación	Real
Adicionar mapa.	3	3

Tabla 21. Iteración #1.

A continuación mediante tablas se evidencian las tareas de programación o ingeniería en las que la Historia de Usuario mencionada anteriormente fue desglosada para un mejor funcionamiento de la aplicación.

Tarea de la Ingeniería	
Número de la Tarea: 1.	**Número de la Historia de Usuario**: 1.
Nombre de la Tarea: Cargar el mapa a utilizar.	
Tipo de Tarea: Desarrollo.	**Puntos Estimados**: 2.
Fecha Inicio: 05/03/2012.	**Fecha Fin**: 19/03/2012.
Programador Responsable: Manuel Alejandro Pérez Rosabal.	
Descripción: Permite cargar el mapa, así como la simbología para luego poder trabajar en la lógica del negocio.	

Tabla 22. Tarea de la Ingeniería #1 para la Iteración #1.

Tarea de la Ingeniería	
Número de la Tarea: 2.	**Número de la Historia de Usuario**: 1.
Nombre de la Tarea: Configurar mapa.	
Tipo de Tarea: Desarrollo.	**Puntos Estimados**: 1.
Fecha Inicio: 20/03/2012.	**Fecha Fin**: 27/03/2012.
Programador Responsable: Manuel Alejandro Pérez Rosabal.	
Descripción: Una vez subido el mapa al servidor, permitirá al usuario realizar las configuraciones correspondientes a *TileCache*, así como seleccionar las capas que el servicio brindar.	

Tabla 23. Tarea de la Ingeniería #2 para la Iteración #1.

4.2.1.2. Iteración #2.

Esta iteración tiene como objetivo darle cumplimiento a la Historia de Usuario “Editar Mapa”.

Historias de Usuarios	Tiempo de Implementación	
	Estimación	Real
Editar mapa.	3	3

Tabla 24. Iteración #2.

La misma tiene la siguiente tarea de programación o ingeniería.

Tarea de la Ingeniería	
Número de la Tarea: 1	**Número de la Historia de Usuario**: 2
Nombre de la Tarea: Editar el mapa.	
Tipo de Tarea: Desarrollo.	**Puntos Estimados**: 3
Fecha Inicio: 28/03/2012.	**Fecha Fin**: 18/04/2012.
Programador Responsable: Manuel Alejandro Pérez Rosabal.	
Descripción: Permite editar el mapa una vez que ha sido adicionado por el usuario, es decir las configuraciones para el servicio brindado por *TileCache* así como la selección de capas.	

Tabla 25. Tarea de la Ingeniería #1 para la Iteración #2.

4.2.1.3. Iteración #3.

Esta iteración tiene como objetivo darle cumplimiento a las Historias de Usuarios “Eliminar Mapa” y “Brindar *URL* del servicio a consumir”.

Historias de Usuarios	Tiempo de Implementación	
	Estimación	Real
Eliminar mapa.	1	1

Brindar *URL* del servicio a consumir.	1	1

Tabla 26. Iteración #3.

A continuación mediante tablas se evidencian las tareas de programación o ingeniería en las que la Historia de Usuario mencionada anteriormente fue desglosada para un mejor funcionamiento de la aplicación.

Tarea de la Ingeniería	
Número de la Tarea: 1.	**Número de la Historia de Usuario**: 3.
Nombre de la Tarea: Eliminar mapas.	
Tipo de Tarea: Desarrollo.	**Puntos Estimados**: 1.
Fecha Inicio: 19/04/2012.	**Fecha Fin**: 26/04/2012.
Programador Responsable: Manuel Alejandro Pérez Rosabal.	
Descripción: Permite eliminar mapas que el usuario no desea ya consumir en el servicio.	

Tabla 27. Tarea de la Ingeniería #1 para la Iteración #3.

Tarea de la Ingeniería	
Número de la Tarea: 2.	**Número de la Historia de Usuario**: 4.
Nombre de la Tarea: Mostar la *URL* al usuario.	
Tipo de Tarea: Desarrollo.	**Puntos Estimados**: 1.
Fecha Inicio: 27/04/2012.	**Fecha Fin**: 04/05/2012.
Programador Responsable: Manuel Alejandro Pérez Rosabal.	

Descripción: Consiste en brindar un dirección de *URL*, en la misma el usuario le permitirá consumir el servicio de tiles del mapa que cargo.

Tabla 28. Tarea de la Ingeniería #2 para la Iteración #3.

4.2.1.4. Iteración #4.

Esta iteración tiene como objetivo darle cumplimiento a la Historia de Usuario “Visor de mapas”.

Historias de Usuarios	**Tiempo de Implementación**	
	Estimación	**Real**
Visor de mapas.	2	2

Tabla 29. Iteración #4.

La misma tiene la siguiente tarea de programación o ingeniería.

Tarea de la Ingeniería	
Número de la Tarea: 1.	**Número de la Historia de Usuario**: 5.
Nombre de la Tarea: Visualizar mapas.	
Tipo de Tarea: Desarrollo.	**Puntos Estimados**: 2.
Fecha Inicio: 05/05/2012.	**Fecha Fin**: 19/05/2012.
Programador Responsable: Manuel Alejandro Pérez Rosabal.	
Descripción: Permite mostrar el mapa una vez configurado por el usuario.	

Tabla 30. Tarea de la Ingeniería #2 para la Iteración #4.

4.3. Fase de pruebas.

Las pruebas son una actividad en la cual un sistema o componente es ejecutado bajo unas condiciones o requisitos especificados, los resultados son observados, así como registrados y una

evaluación es hecha de algún aspecto del sistema o componente. La prueba de *software* es un elemento crítico para la garantía de la calidad del *software* y representa una revisión final de las especificaciones del diseño y de la codificación.

Una de las prácticas de la metodología *XP* es el uso de las pruebas para garantizar el funcionamiento de los códigos que se vayan implementando. 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.

La metodología ágil *XP* divide las pruebas en dos grupos: pruebas unitarias y pruebas de aceptación. Las pruebas unitarias son desarrolladas por los programadores y se encargan de verificar el código automáticamente y las pruebas de aceptación están destinadas a verificar que al final de cada iteración las Historias de Usuario cumplen con la funcionalidad asignada y satisfagan las necesidades del cliente. (Gutiérrez, y otros, 2010)

4.3.1. Pruebas unitarias.

De acuerdo con lo que plantea la metodología *XP*, las pruebas unitarias o pruebas de unidad consisten en comprobaciones (manuales o automatizadas) desarrolladas por los programadores. Las cuales se realizan para verificar que el código correspondiente a un módulo concreto se comporta de manera esperada. Las pruebas unitarias proporcionan beneficios tales como. (Gutiérrez, y otros, 2010)

- Brindan al programador una inmediata retroalimentación de cómo está realizando su trabajo.
- El programador puede realizar cambios de forma segura respaldada por efectivos casos de prueba.
- Permite saber si una determinada funcionalidad se puede agregar al sistema existente sin alterar el funcionamiento actual del mismo.

Las pruebas unitarias no se le aplicarán al *software* debido a que según los expertos en la metodología de desarrollo *XP* recomiendan utilizar las pruebas de aceptación. Las pruebas de aceptación son consideradas las más adecuadas, pues significa el grado de satisfacción que tenga el cliente con el producto. En estas pruebas el cliente puede comprobar si todas las historias de usuario se implementaron de acuerdo a lo concebido.

4.3.2. 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. Las pruebas de aceptación se elaboran a lo largo de la iteración, en paralelo con el desarrollo del sistema y adaptándose a los cambios que el sistema sufra. Las pruebas de aceptación son consideradas como “pruebas de caja negra”.

Los clientes son responsables de verificar que los resultados de estas pruebas sean correctos. Así mismo, en caso de que fallen varias pruebas, deben indicar el orden de prioridad de resolución. Una historia de usuario no se puede considerar terminada hasta tanto pase correctamente todas las pruebas de aceptación. (Gutiérrez, y otros)

Caso de prueba de aceptación	
Código: PA_HU1.	**Número de la Historia de Usuario**: 1.
Nombre: Comprobar que el mapa sea cargado por el sistema.	
Descripción: Evaluar que el mapa se muestre de acuerdo a las necesidades del cliente.	
Condiciones de Ejecución: El cliente debe comprobar que el mapa se cargó correctamente.	
Entrada/Pasos de Ejecución: El cliente debe introducir el nombre del mapa, así como subir el archivo “.map” y la simbología referente al mismo. Luego el cliente selecciona las capas que brindará el servicio y configura el *TileCache* en correspondencia de sus necesidades.	
Resultado Esperado: Se muestra correctamente el mapa cargado por el cliente en el visor de mapas.	
Evaluación de la Prueba: Prueba satisfactoria. Se recomienda implementar el sistema para que permita adicionar otros tipos de mapas.	

Tabla 31. Caso de prueba de aceptación para la Historia de Usuario #1.

Caso de prueba de aceptación	
Código: PA_HU2.	**Número de la Historia de Usuario**: 2.

Nombre: Editar mapa.
Descripción: Evaluar que una vez cargado el mapa se pueda editar para mejor comodidad del usuario.
Condiciones de Ejecución: El cliente de comprobar que el mapa se muestre según sus especificaciones.
Entrada/Pasos de Ejecución: Primeramente debe adicionar el mapa, luego según sus necesidades cambiar las configuraciones del *TileCache* y las capas que desea servir mediante la tecnología.
Resultado Esperado: Se muestra correctamente el mapa según las especificaciones del cliente.
Evaluación de la Prueba: Prueba satisfactoria. Se recomienda que el usuario pueda editar el mapa con las simbologías presentes en el servidor, así como adicionar nuevos símbolos a las ya existentes.

Tabla 32. Caso de prueba de aceptación para la Historia de Usuario #2.

Caso de prueba de aceptación	
Código: PA_HU3.	**Número de la Historia de Usuario**: 3.
Nombre: Comprobar que el mapa fue eliminado.	
Descripción: Evaluar que el mapa seleccionado por el cliente fue eliminado satisfactoriamente.	
Condiciones de Ejecución: El cliente debe seleccionar el mapa.	
Entrada/Pasos de Ejecución: El cliente debe seleccionar el mapa que desea eliminar.	
Resultado Esperado: El mapa se elimine completamente de la aplicación de forma satisfactoria.	
Evaluación de la Prueba: Prueba satisfactoria.	

Tabla 33. Caso de prueba de aceptación para la Historia de Usuario #3.

Caso de prueba de aceptación

Código: PA_HU4.	**Número de la Historia de Usuario**: 4.
Nombre: Comprobar que se muestre la *URL* de donde se va a consumir el servicio de mapas.	
Descripción: Evaluar que la *URL* se muestre y esté disponible.	
Condiciones de Ejecución: El cliente de comprobar que la *URL* se muestre.	
Entrada/Pasos de Ejecución: El cliente debe seleccionar el mapa del cual desea ver la *URL* del servicio.	
Resultado Esperado: El cliente debe probar que la *URL* que el sistema muestra sea correcta.	
Evaluación de la Prueba: Prueba satisfactoria.	

Tabla 34. Caso de prueba de aceptación para la Historia de Usuario #4.

Caso de prueba de aceptación	
Código: PA_HU5.	**Número de la Historia de Usuario**: 5.
Nombre: Comprobar que se muestre en un prototipo de interfaz, el mapa que añadió y configuró el cliente.	
Descripción: Evaluar la correcta visualización del mapa y el tiempo de respuesta del mismo.	
Condiciones de Ejecución: El cliente de comprobar que el mapa una vez añadido y configurado se muestre en el visor de mapas.	
Entrada/Pasos de Ejecución: El cliente debe seleccionar el mapa a visualizar.	
Resultado Esperado: Se muestre el mapa de forma satisfactoria.	
Evaluación de la Prueba: Prueba satisfactoria. Se recomienda mejorar el visor de mapas, para que la generación de los *tiles* (teselas) se realice de manera automática y aumente la eficiencia de la aplicación.	

Tabla 35. Caso de prueba de aceptación para la Historia de Usuario #5.

4.3.3. Resultados de las pruebas de aceptación.

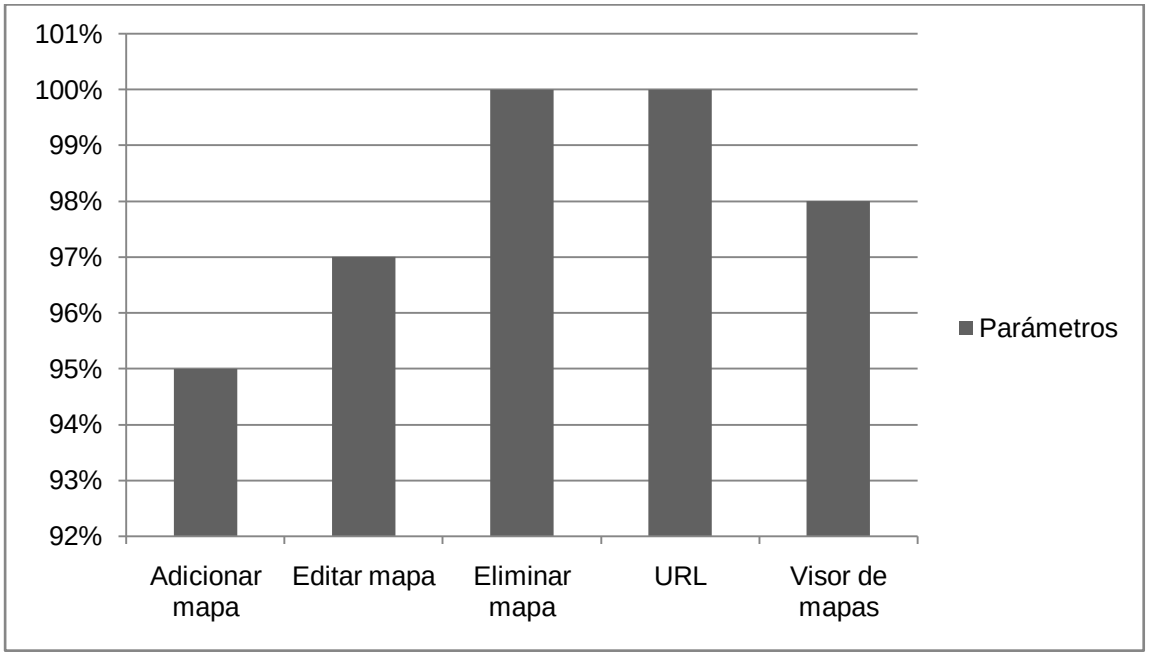

Figura 4. Resultado de la pruebas de aceptación.

4.4. Validación de la solución propuesta.

A continuación mediante tablas y graficas se evidencian algunos resultados de rendimientos obtenidos con la herramienta *firebug*[xxvii]. Comparando el tiempo de renderización de un mapa de prueba con la solución propuesta bajo un mismo escenario. Como la velocidad es inversamente proporcional al tiempo, tomamos esta variable para realizar la comparación, obteniendo los siguientes resultados.

- Comparación realizada disminuyendo la escala del mapa. (*Zoom In*)

Disminuir escala			
Escala Inicial	**Escala Final**	**Mapa de Prueba**	**TileCache**
8'923'284	4'461'642	4,80s	0,545s
4'461'642	2'230'821	3,99s	0,553s
2'230'821	1'115'411	6,33s	0,664s
1'115'411	557'705	3,58s	0,742s
Total		18.7s	2,504s

Tabla 36. Tabla comparativa #1.

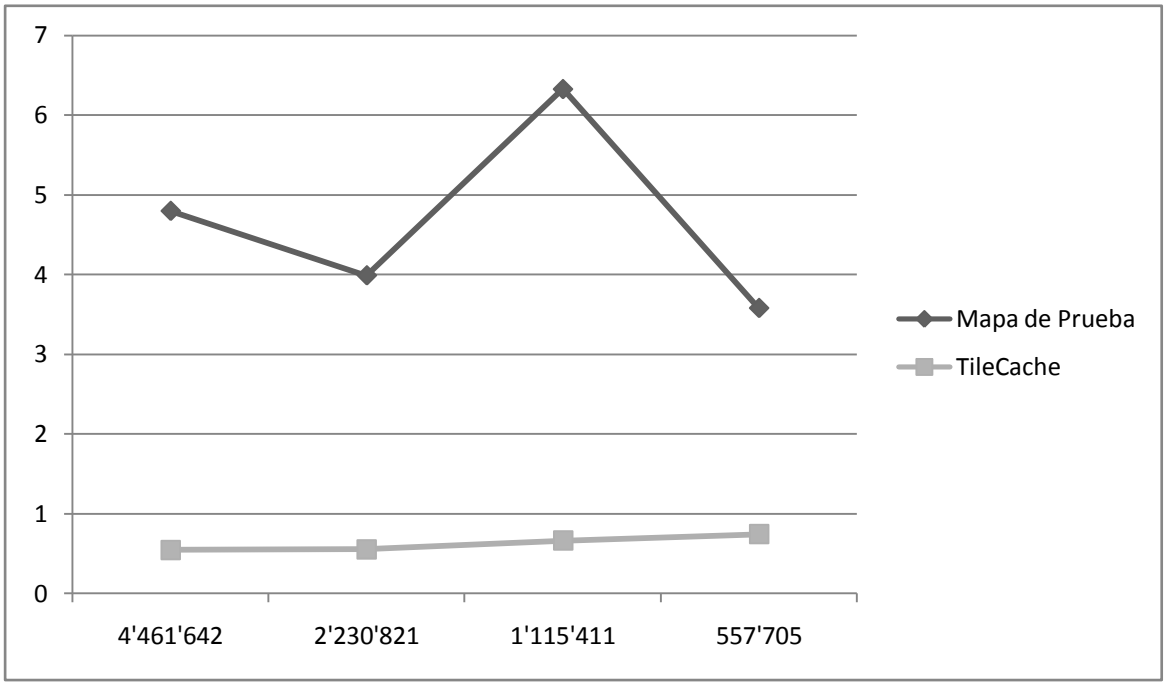

Figura 5. Gráfica de resultados #1.

- Comparación realizada aumentando la escala del mapa. (*Zoom Out*)

Aumentar escala			
Escala Inicial	**Escala Final**	**Mapa de Prueba**	**TileCache**
319'602	639'204	3,71s	0,753s
639'204	1'278'407	3,12s	0,548s
1'278'407	2'556'815	3,85s	0,550s
2'556'815	5'113'630	4,24s	0,467s
Total		14,92s	2,318s

Tabla 37. Tabla comparativa #2.

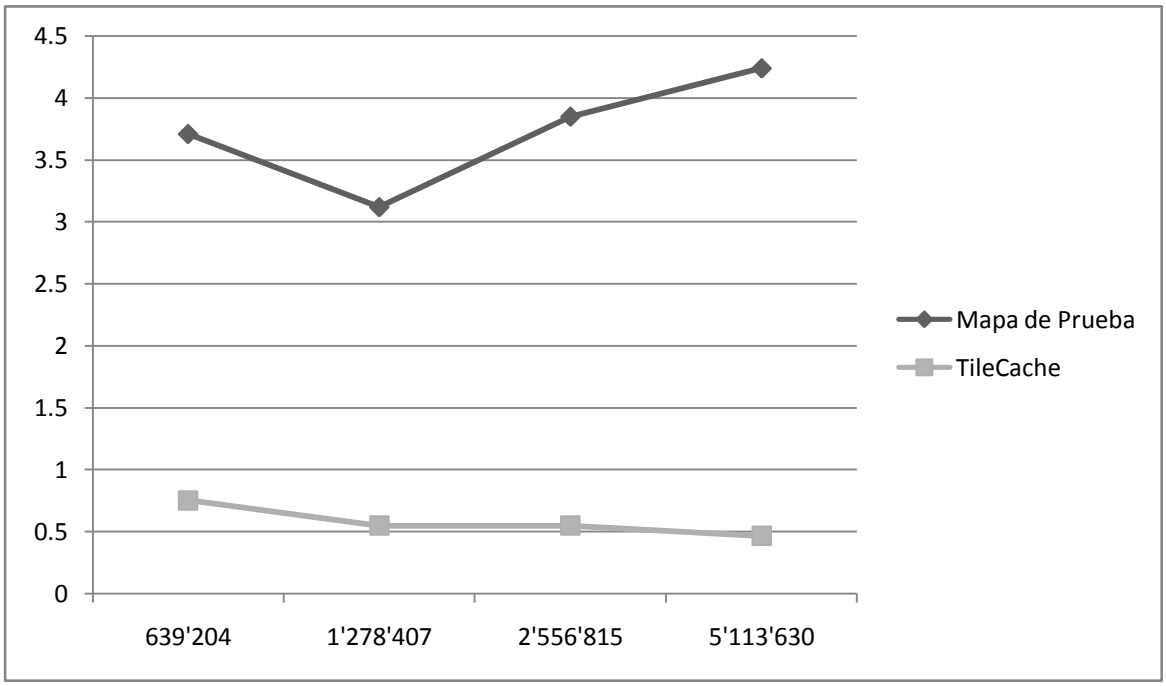

Figura 6. Gráfica de resultados #2.

- Comparación realizada moviendo el mapa. (*Pan*)

Moviendo el mapa		
Dirección	**Mapa de Prueba**	**TileCache**
Izquierda	3,71s	0,753s
Derecha	3,12s	0,548s
Arriba	3,85s	0,550s
Abajo	4,24s	0,467s
Total	14.92	2.318

Tabla 38. Tabla comparativa #3.

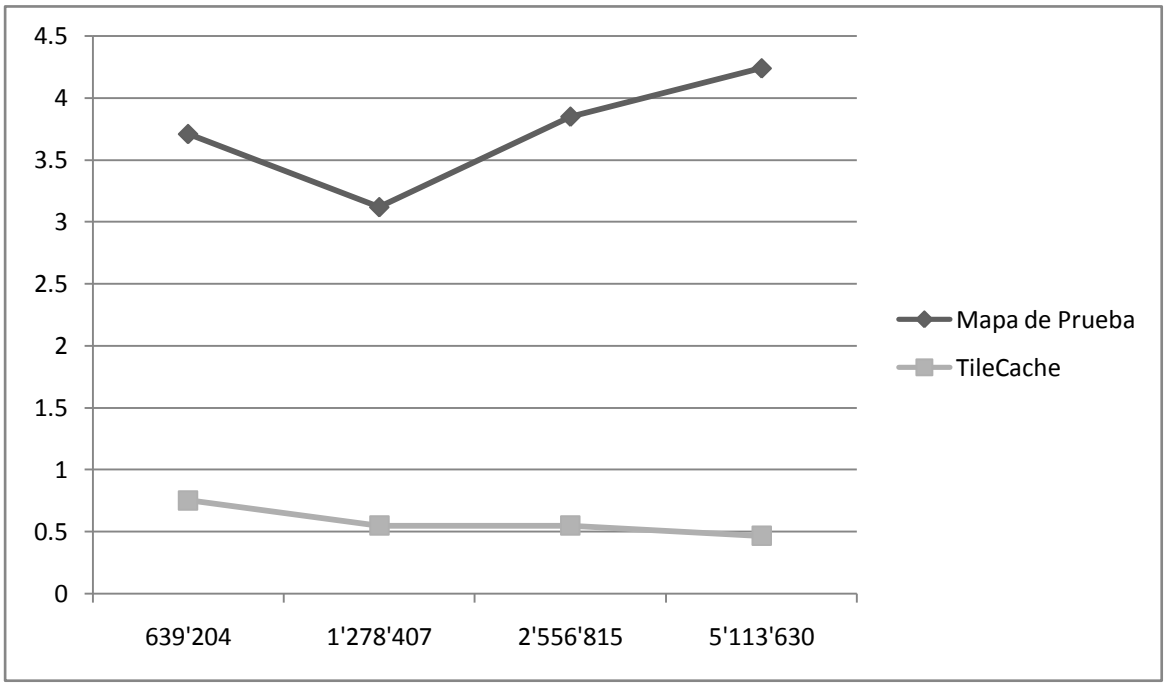

Figura 7. Gráfica de resultados #3.

Una vez finalizadas las pruebas y analizado los resultados se llega a la conclusión que para todos los casos de renderización (velocidad de visualización) de los mapas, *TileCache* lo realiza en menor tiempo, demostrando así, el objetivo general de la presente investigación.

4.5. Conclusiones parciales.

Después de concluido este capítulo, se obtiene completamente construido y probado el sistema. Se detallaron las tareas de programación o ingeniería generadas por cada historia de usuario que permitieron la implementación del sistema. Se realizó además un estudio sobre las pruebas unitarias y de aceptación. Las primeras permitieron validar la implementación de la aplicación demostrado la eficiencia en la visualización de los mapas usando TileCache. Por otro lado, las pruebas de aceptación como las más indicadas para comprobar el funcionamiento del *software*, debido a que demuestran la satisfacción del cliente con el producto. Demostraron la necesidad de aumentar la escalabilidad del sistema para que pudiera admitir otros tipos de mapas, la utilización de la simbología presente en el sistema y la generación de caché de forma automática.

Conclusiones Generales

Finalizado el proceso investigativo y el desarrollo del sistema para la visualización y construcción de caché de mapas basado en tecnologías ágiles se puede arribar a una serie de conclusiones que a continuación se exponen:

- Se dio cumplimiento al objetivo planteado, logrando desarrollar un sistema que permitiera gestionar el servicio de mapas base en caché para mejorar la renderización (velocidad de visualización) de los mismo, pues todas las historias de usuarios se implementaron satisfactoriamente teniendo en cuentas las exigencias no funcionales.
- Se logró documentar el proceso, así como las tecnologías y el resto de las herramientas que acompañan la implementación de la aplicación, siguiendo los criterios que rige la Universidad para posterior explotación, modificación y comercialización.
- Las validaciones realizadas al sistema, manifestaron que la eficiencia de las tecnologías basadas en el modelo de *tiles* para la renderización de los mapas, demostrado así, el objetivo general de la presente investigación.

Recomendaciones.

Trascurrido el proceso de desarrollo de la aplicación se formula las siguientes recomendaciones:

- Continuar el desarrollo de la aplicación para ampliar las capacidades del servicio y la adquisición de nuevos mapas que posibilite una mayor escalabilidad en la visualización de los mapas.
- Implementar el resto de los formatos soportados por la tecnología *TileCache*, de esta manera lograr una total explotación de la misma en la Universidad.

Bibliografía

Barroso, Alfonso Runío y Gutiérrez, Javier Puebla. 1997. *Los Sistemas de Información Geográficos:Origen y Persperctivas.* 1997.

Beck, K. 1999. *Extreme Programming Explained. Embrace Change.* [trad.] Addison Wesley. s.l. : Pearson Education, 1999.

Bosque Sendra, J. 1997. *Sistemas de Información Geográfica.* s.l. : Rialp, 1997. pág. 451.

Bravo, Javier Domínguez. 2000. *Breve Introducción a la Cartografía y a los Sistemas de Información Geográfica (SIG).* s.l. : CIEMAT, 2000.

Canós, José H., Letelier, Patricio y Carmen, María. 2003. *Todo Àgil.* Valencia : Universidad Politécnica de Valencia, 2003.

Carmona, Alvaro de J. y R., Jhon Jairo Monsalve. 1999. Sistemas de información geográfica - Monografías. *Monografías.com - Tesis, Documentos, Publicaciones y Recursos Educativos.* [En línea] 07 de 12 de 1999. [Citado el: 04 de 06 de 2012.] http://www.monografias.com/trabajos/gis/gis.shtml.

Cartografía digital y Espeleología. **Rico, José Lorenzo Herrero. 2000.** 27, s.l. : Lapiaz, 2000. 27.

Corbea, Maite Rodríguez y Pérez, Meylin Ordóñez. 2007. *La Metodologìa XP aplicables a los software esducativos en Cuba.* La Habana : s.n., 2007.

Corti, Paolo. 2008. A day with TileCache: generating KML Super-Overlays - Thinking in GIS. [En línea] 6 de 8 de 2008. [Citado el: 4 de 12 de 2010.] http://www.paolocorti.net/2008/08/06/a-day-with-tilecache-generating-kml-super-overlays/.

2008. Definición de mapa - Qué es, Significado y Concepto. [En línea] 2008. [Citado el: 3 de 12 de 2010.] http://definicion.de/mapa/.

Definición.org. 2012. Definición de lenguaje de programación. *definición.org.* [En línea] 2012. [Citado el: 24 de 05 de 2012.] http://www.definicion.org/lenguaje-de-programacion.

Entorno Virtual de Aprendizaje, UCI. 2011. [En línea] 24 de febrero de 2011. http://eva.uci.cu/mod/resource/view.php?id=20049.

Escribano, Gerardo Fernández. 2002. *Introducción a Extremie Programing.* [Docuemnto] 2002.

Espinosa, Antonio Membrides. 2000. *Fundamentos del Mapserver, Mapscript, PostGIS y su integración con el Cartoweb.* 2000.

2010. GeoWebCache - GeoBolivia. [En línea] 22 de 9 de 2010. [Citado el: 3 de 12 de 2010.] http://geobolivia.abc.gob.bo/?GeoWebCache.

Gianfelici, Esteban. 2008. El dato geográfico y la información geográfica.Definiciones y diferencias. [En línea] 7 de 2008. [Citado el: 3 de 12 de 2010.] http://www.mapasymapas.com.ar/el%20dato%20geografico.php.

—. 2008. Instalar MapServer y ms4w. Guia de instalación fácil y en castellano. [En línea] 7 de 2008. [Citado el: 3 de 12 de 2010.] http://www.mapasymapas.com.ar/mapserver.php.

Guglielmetti, Marcos. 2004. Definición de Arquitectura de Software - Significado y Definición de Arquitectura de Software. *MASTERMAGAZIN.* [En línea] 2004. [Citado el: 24 de 05 de 2012.] http://www.mastermagazine.info/termino/3916.php.

Gutiérrez, J. J., y otros. 2010. *PRUEBAS DEL SISTEMA EN PROGRAMACIÓN EXTREMA.* [Documento] Sevilla : Department de Lenguajes y Sistemas Informáticos University of Sevilla, 2010.

—. *PRUEBAS DEL SISTEMA EN PROGRAMACIÓN EXTREMA.* [Documento] Sevilla : Department de Lenguajes y Sistemas Informáticos University of Sevilla.

Henst, Van Der. 2001. 2001.

Hernández León, Rolando Alfredo y Coello Gónzalez, Sayda. 2002. *El paradigma cuantitativo de la investigación científica. s.l. : EDUNIV.* 2002.

Jason Gilmore, W. y H. Treat, Robert. 2006.. *Beginning PHP and PostgreSQL 8 from Novice to Professional. s.l. .* s.l. : Apress , 2006.

Javier García de Jalón. 2000. Aprenda Java como si estuviera en primero,Ciudad de la Habana. [En línea] 2000. http://www.tecnun.es/asignaturas/Informat1/AyudaInf/aprendainf/Java/Java2.pdf.

José H. Canós, Patricio Letelier y Mª Carmen Penadés. *Todo Àguil.* Valencia : Universidad Politécnica de Valencia.

Joskowicz y Reglas, Ing. José. 2008. *Prácticas en extreme Programming.* 2008.

2011. La arquitectura cliente servidor. [En línea] 5 de Febrero de 2011. http://temariotic.wikidot.com/la-arquitectura-cliente-servidor#toc0.

Lomeli, Walter Ivan Reyes. 2007. 4.3 CARACTERISTICAS Y VENTAJAS DEL APACHE for arquitectura_dela_web. *Scribd.* [En línea] Marzo de 2007. [Citado el: 01 de 06 de 2012.] http://www.scribd.com/doc/52208534/29/CARACTERISTICAS-Y-VENTAJAS-DEL-APACHE.

Mariel, Bertha y Avendaño, Márquez. 2004. Implementación de un reconocedor de voz gratuito a el sistema de ayuda a invidentes Dos-Vox en español. *Colección de Tesis Digitales Universidad de las Américas Puebla.* [En línea] 12 de 01 de 2004. [Citado el: 24 de 05 de 2012.] http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/marquez_a_bm/capitulo5.pdf.

Martín, Pedro Rufo. 2001. ASPTutor.com Descarga de Manuales y Ejemplos. *ASPTutor.* [En línea] 31 de 01 de 2001. [Citado el: 01 de 06 de 2012.] http://www.asptutor.com/zip/cbhtml.pdf.

Martín, R. García y Fernández, J.P. de Castro. *Benchmarking de implementaciones WMS-C de software libre.* Valladolid : s.n.

Martínez, Jaime Solís. 2009. Sitio Web de la E.U.I.T.I.O. *Sitio Web de la E.U.I.T.I.O.* [En línea] 14 de 01 de 2009. [Citado el: 04 de 06 de 2012.] http://www.google.com.cu/url?sa=t&rct=j&q=entorno+integrado+de+desarrollo+definicion&source=web&cd=2&ved=0CFgQFjAB&url=http%3A%2F%2Fforo.ignetwork.net%2Fshowthread.php%3F15188-IDE-Entorno-integrado-de-desarrollo-(Concepto-importante)&ei=BmTNT8yGBqed6AHgrY.

Meta Carta Labs. 2010. TileCache, from MetaCarta Labs. [En línea] MetaCarta Labs, 2010. [Citado el: 3 de 12 de 2010.] tilecache.org.

Morissette, Daniel. 2011. PHP MapScript - MapServer 6.0.3 documentation. *Map Server - open sorce web mapping.* [En línea] 2011. [Citado el: 24 de 05 de 2012.] http://mapserver.org/mapscript/php/index.html.

NetBeans. 2011. Bienvenido a NetBeans. [En línea] 20 de febrero de 2011. http://netbeans.org/index_es.html..

—. 2012. NetBeans. *NetBeans.* [En línea] 2012. [Citado el: 04 de 06 de 2012.] http://netbeans.org/features/.

Open Geospatial Consortium. Bienvenidos a Map Server. *Map Server.* [En línea] [Citado el: 23 de 11 de 2011.] http://mapserver.org/es/index.html.

—. 1994-2010. Web Map Service|OGC®. [En línea] 1994-2010. [Citado el: 3 de 12 de 2010.] http://www.opengeospatial.org/standards/wms.

Open Source Web Mapping. 2011. Bienvenidos a Map Server. *Map Server.* [En línea] 2011. [Citado el: 23 de 11 de 2011.] http://mapserver.org/es/index.html.

OpenLayers. 2012. OpenLayers: Home. *OpenLayers: Home.* [En línea] OpenLayers, 2012. [Citado el: 01 de 06 de 2012.] http://openlayers.org/.

OSGeo. 2010. WMS Tile Caching - OSGeo Wiki. [En línea] 25 de 2 de 2010. [Citado el: 3 de 12 de 2010.] http://wiki.osgeo.org/wiki/WMS_Tile_Caching.

Pérez, Iván Nieto. 2008. El Código - Tutoriales y ejempos de JavaScript, DHTML, HTML y CCS. *Capítulo 1: Introducción.* [En línea] 2008. [Citado el: 12 de 05 de 2012.] http://www.elcodigo.com/tutoriales/javascript/javascript1.html.

PostgreSQL. 2012. PostgreSQL: Documentation: 8.4: What is PostgreSQL? *PostgreSQL: The world's most advance open source database.* [En línea] PostgreSQL, 2012. [Citado el: 04 de 06 de 2012.] http://www.postgresql.org/docs/8.4/static/intro-whatis.html.

Ramírez, Ignacio Gámez. *Escalabilidad en servicios de mapas. Modelo de teselas en cache con OpenLayers.* s.l. : Geograma S.L.

—. 2008. *Escalabilidad en servicios de mapas. Modelo de teselas en cache con OpenLayers.* s.l. : Geograma S.L., 2008.

Roldan, Gabriel. OpenGeo: GeoWebCache. [En línea] [Citado el: 3 de 12 de 2010.] http://opengeo.org/community/geowebcache/.

Salinas, Jorge Gaspar Sanz y Lajara, Miguel Montesinos. 2009. Panorama actual del ecosistema de SIG libre. [En línea] 23 de 3 de 2009. [Citado el: 3 de 12 de 2010.] https://confluence.prodevelop.es/display/pan/Panorama+del+ecosistema+de+Software+Libre+para+SIG+-+completo?showComments=true&showCommentArea=true.

Sencha. 2012. HTML5 Framework for Desktop and Mobile Devices. Build HTML5 Apps for Any Browse. *HTML5 Framework for Desktop and Mobile Devices. Build HTML5 Apps for Any Browse.* [En línea] Sencha, 2012. [Citado el: 25 de 05 de 2012.] http://www.sencha.com/.

Silva, R. José Luis Batista. 2005. Aplicación de Sistemas de Información Geográfica en Cuba. [En línea] 11 de 2005. [Citado el: 3 de 12 de 2010.] http://www.mappinginteractivo.com/plantilla-ante.asp?id_articulo=1051.

SISTEMA DE INFORMACIÓN GEOGRÁFICA. *SISTEMA DE INFORMACIÓN GEOGRÁFICA.* [En línea] [Citado el: 23 de 11 de 2011.] http://sigeco.ecologia.campeche.gob.mx/info_sig.php.

Thompson, Ivan. 2008. Definición de Información. [En línea] 10 de 2008. [Citado el: 3 de 12 de 2010.] http://www.promonegocios.net/mercadotecnia/definicion-informacion.html.

WMS Tiling Client Recomendation (OSGeo). [En línea] [Citado el: 2 de 12 de 2010.] http://wiki.osgeo.org/wiki/WMS_Tiling_Client_Recommendation.

WordPress. 2008. Definición de sistema de información - Qué es - Significado y Concepto. [En línea] 2008. [Citado el: 3 de 12 de 2010.] http://definicion.de/sistema-de-informacion/.

Zaldívar, Yoenis Pantoja. 2008. *Documento Visión.* La Habana : Dirección de Calidad de la Infraestructura Productiva, 2008.

Glosario de términos

[i] ***Software*** es un término genérico que se aplica a los componentes no físicos de un sistema informático, como los programas, sistemas operativos, etc., que permiten a este ejecutar sus tareas.

[ii] ***TileCache*** se traduce como el almacenamiento en memoria de los tiles (mosaicos, teselas o pequeñas imágenes).

[iii] **SVG** (*Scalable Vector Graphics)* es un lenguaje para describir gráficos vectoriales bidimensionales, tanto estáticos como animados (estos últimos con ayuda de *SMIL*), en *XML*.

[iv] **WebCGM** (*Web Computer Graphics Metafile*) es un formato de imagen de tipo vectorial, es decir, que almacena un conjunto de líneas y figuras. Esto es óptimo para diagramas complejos en que se requiere hacer zoom o visualizar por capas.

[v] ***Common Gateway Interface*** (Interfaz de Salida Común) es de las primeras formas de programación *Web* dinámica que permite extender las capacidades de *HTTP*.

[vi] ***Aplication Programming Interface*** (Interfaz de Programación de Aplicaciones) es una "llave de acceso" a funciones que nos permiten hacer uso de un servicio web provisto por un tercero, dentro de una aplicación web propia, de manera segura.

[vii] ***TileCache*** se traduce como el almacenamiento en memoria de los tiles (mosaicos o pequeñas imágenes).

[viii] ***Web Mapping Service*** (Servicio de Mapas *Web*) – en Caché.

[ix] **Licencia *BSD*** (*Berkeley Software Distribution*) es una licencia de software libre permisiva.

[x] ***Transportation Management Systems*** (Sistemas de Gestión de Transporte) permite la comunicación de ida y vuelta, es decir, la retroalimentación.

[xi] ***Python*** es un lenguaje de *scripting* (secuencias de comandos) independiente de plataforma y orientado a objetos, preparado para realizar cualquier tipo de programa, desde aplicaciones *Windows* a servidores de red o incluso, páginas *Web*.

[xii] **Renderización** es el proceso de generar una imagen desde un modelo.

[xiii] ***Tiles*** se traduce como pequeñas imágines denominadas también teselas o mosaicos.

[xiv] **Pixel** es la menor unidad homogénea en color que forma una imagen digital.

[xv] ***HTTP*** *HyperText Transfer Protocol* (Protocolo de transferencia de hipertexto) es el método más común de intercambio de información en la *World Wide Web*, el método mediante el cual se transfieren las páginas web a un ordenador.

[xvi] ***Hardware*** es el soporte físico del conjunto de elementos materiales que componen un ordenador. *Hardware* también son los componentes físicos de una computadora tales como el disco duro, *CD-Rom*, disquetera (*floppy*), etc. En dicho conjunto se incluyen los dispositivos electrónicos y electromecánicos, circuitos, cables, tarjetas, armarios o cajas, periféricos de todo tipo y otros elementos físicos.

[xvii] ***GET*** recupera información de un servidor, en este caso de mapas.

[xviii] ***GetMap*** recupera información de un mapa que consta de una o más capas.

[xix] ***API REST*** es una librería de funciones, a la que se accede por el protocolo HTTP.

[xx] ***GeoRSS*** (*Really Simple Syndication*) pertenece a la familia de estándares RSS y se utiliza para representar información georreferenciada anotando geográficamente las entradas del RSS para que dicha información pueda ser localizada en el mapa.

[xxi] ***Mapnik*** es una herramienta para el renderizado de mapas.

[xxii] **DiskCache** es la caché en disco.

[xxiii] ***MemoryCache*** es la caché en memoria.

[xxiv] ***Memcache*** es una memoria de clave y valor para almacenar en pequeños trozos de datos arbitrarios (cadenas, objetos).

[xxv] **Clúster** es el conjunto o conglomerado de computadoras construidos mediante la utilización de hardwares comunes.

[xxvi] ***XML*** *Extensible Markup Language*, una especificación/lenguaje, diseñado especialmente para los documentos de la *Web*. Permite que los diseñadores creen sus propias etiquetas, permitiendo la definición, transmisión, validación e interpretación de datos entre aplicaciones y entre organizaciones.

[xxvii] ***Firebug*** es un *plugin* de *Firefox* que nos brinda un paquete de utilidades para el desarrollo de páginas y aplicaciones Web. Nos permite debugear, monitorizar y modificar el *CSS*, *HTML* y *JavaScript* en ejecución, es decir, a medida que lo visualizamos.