SERVEI DE SISTEMES D'INFORMACIÓ GEOGRÀFICA I TELEDETECCIÓ
Universitat de Girona

VII JORNADAS DE SIG LIBRE

Lineamientos para el desarrollo de aplicaciones SIG Web.

B.S. Acosta Correa [(1)], *A.V. Yanza Hurtado* [(2)]

[(1)] Coordinadora Centro de Información Geográfica (CentroIG). Universidad EAFIT, Medellín, Colombia, sacosta@eafit.edu.co

[(2)] Asistente de Investigación Centro de Información Geográfica (CentroIG). Universidad EAFIT, Medellín, Colombia, ayanza@eafit.edu.co.

RESUMEN

El proyecto de generar una serie de lineamientos que orienten el desarrollo de aplicaciones SIG web, surgió en el CentroIG (Laboratorio de SIG), con el apoyo de la dirección de investigación de EAFIT. La propuesta surgió a raíz de dos casos: la construcción del Campus Universitario Georreferenciado (2009) y del prototipo del Sistema de Movilidad de Medellín (2010), en los cuales se enfrentaron dificultades a lo largo de las etapas de diseño, construcción y producción. A partir de esto, se vio la oportunidad de recoger el conocimiento y las lecciones aprendidas, para divulgarlas a los profesionales del campo de las tecnologías de la información y las comunicaciones, especialmente para aquellos del medio local y nacional. Con el objetivo de que los lineamientos tuvieran un buen fundamento se planteó una metodología que abarcara en primer lugar el estudio teórico de buenas prácticas en ingeniería de software y SIG. En segunda instancia, se realizó un trabajo práctico, mediante la construcción de pilotos, haciendo énfasis en el cliente (frontend), uno con software propietario y otro con software libre. Entre las herramientas seleccionadas están Geoserver, GeoExt, ArcGIS Server y el API JavaScript de ESRI.
Los lineamientos se plantearon en tres momentos: antes, durante y después del desarrollo. Entre los propuestos se encuentran: la conformación adecuada del equipo humano, parámetros para la selección de tecnologías, la arquitectura como punto de partida y la modularización de una aplicación como eje para su evolución.

Palabras clave: SIG Web, construcción software, buenas prácticas, software libre, software propietario.

INTRODUCCIÓN

Existe hoy en día una creciente necesidad de aplicaciones de Sistemas de Información Geográfica (SIG) en Web, ya que son muchas las áreas del saber que requieren su uso para cumplir con mayor acierto sus procesos, tal como la gestión pública, medioambiental, ingenieril, entre otras. También son solicitadas por comunidades de usuarios que están dispersos y por organizaciones o entidades que precisen compartir e integrar datos georreferenciados para realizar diferentes tipos de análisis espacio-territoriales y ayudar en la toma de decisiones. De igual forma, los ciudadanos comunes se interesan cada vez más en herramientas que les permitan visualizar mapas y obtener información de su interés (rutas de transporte, estado del tráfico, sitios turísticos, localización de direcciones). Sin embargo, muchos de los productos software desarrollados no satisfacen plenamente los objetivos para los que fueron diseñados, presentando múltiples defectos cuando son puestos en uso. Fallas y errores que son originados en diferentes puntos del proceso de desarrollo: desde requisitos incompletos o mal entendidos, arquitecturas que pasan por alto calidades sistémicas vitales (*performance*, disponibilidad, mantenibilidad, usabilidad, etc), tecnologías inadecuadas o empleadas de manera equivocada, hasta despliegues en hardware insuficiente para soportar las aplicaciones.

A partir del recorrido y el quehacer diario en el Centro IG de la Universidad EAFIT, se evidencia que a nivel local, regional y nacional, hay poco conocimiento y falta de experiencia en la creación de aplicaciones SIG en ambiente Web: Preguntas del tipo ¿cómo publico un mapa en Internet?, ¿cómo incluyo un mapa en una página Web?, ¿qué herramientas puedo utilizar?, ¿dónde se guarda un mapa? son comunes de escuchar, por lo cual se hace necesario divulgar y contribuir de manera práctica en la formación académica en este campo.

Los lineamientos son fruto de tres factores: las experiencias y enseñanzas adquiridas de diferentes proyectos, el estudio de las bases teóricas y buenas prácticas en el campo de la ingeniería de software y los SIG y finalmente, de los aprendizajes derivados de la construcción de aplicaciones piloto, las cuales permitieron obtener perspectivas a nivel general (de todo el proceso) y detallado (en las etapas de análisis, diseño e implementación). Se elaboraron dos pilotos, uno con software libre y otro con herramientas de tipo propietario.

PROCESO DE CONSTRUCCION DE LAS APLICACIONES PILOTO

El punto de partida, anterior a la construcción de las aplicaciones, fue la revisión y estudio de temáticas relacionadas a la ingeniería de software (arquitectura, *frameworks*, lenguajes) y SIG (conceptos de geodesia, geoservicios), de manera que el soporte teórico estuviera actualizado y sirviera como referente a lo largo del proceso de desarrollo de los pilotos.

De un proyecto realizado en el CentroIG, denominado Campus Georreferenciado, que consistía en construir el mapa interactivo de la Universidad EAFIT, se reutilizaron artefactos como el levantamiento de requisitos [1] y los datos (almacenados en una Geodatabase), que constituyen un insumo indispensable y para el caso de estudio no tenían costo extra y cumplen con estándares de calidad.

Posteriormente se seleccionaron las tecnologías sobre las cuales se construyeron las aplicaciones. Luego, se hizo el diseño o arquitectura de los pilotos y se procedió con la codificación.

Requisitos de los pilotos

La tabla a continuación resume a manera de *backlog* los requisitos de las aplicaciones piloto.

Tabla 1. Requisitos de las aplicaciones piloto

#	Requisito	Tipo	Implicaciones	Prioridad
1	Localizar oficina o aula	Funcional		Alta
2	Localizar sitio de interés	Funcional		Alta
3	Desplegar información complementaria cuando se realicen 1y 2.	Funcional	Debe existir archivos con este tipo de información (i.e fotografías)	Media
4	Debe tener herramientas de zoom in/out, pan, reload y *select by point* (identificar)	Funcional	La herramienta *select by point* brindará sólo información de algunas capas.	Alta
5	La imagen de fondo que tenga el mapa puede ser de un servicio externo.	Funcional	La fuente puede ser Google, OpenStreetMap, Bing o ArcGIS	Baja
6	La base de datos puede ser Oracle 10g.	No funcional	Debe tener habilitada su extensión *locator*, para que pueda manejar la información geográfica	Media

Selección de herramientas

Para este proceso, de selección de herramientas, en primera instancia se consideraron dos conjuntos: libres y propietarios.

En la categoría de software propietario, se decidió por el producto ArcGIS Server de ESRI, ya que es una de las casas fabricantes de mayor reconocimiento a nivel mundial, es utilizado ampliamente tanto por entidades gubernamentales como por empresas privadas y EAFIT cuenta con la licencia para su uso, en la versión 9.3.

De otra parte, para el software libre, el primer criterio que se tuvo para realizar el filtro, fue que la herramienta hiciera parte de los proyectos cobijados por el *Open Source GeoSpatial Foundation* (OSGeo), ya que al estar bajo su sombrilla, tienen mayor estabilidad y perdurabilidad, que aquellos que se sostienen por sí mismos. La siguiente tabla muestra los otros parámetros de selección:

Tabla 2. Criterios evaluados para selección de herramientas

	Herramientas OSGeo								
Criterio	**1.OpenLayers**	**2.GeoMajas**	**3.MapBender**	**4.MapFish**	**5.GeoMoose**	**6.GeoServer**	**7.MapServer**	**8.deegree**	**9.QGISServer**
Lenguaje programación en top 20 de Tiobe [2]									
Fecha última versión inferior a 1 año									
Experiencia en el manejo de la herramienta									
Pertinencia: para clientes, que sean APIs que permitan programar y no tipo Geoportal que son orientados hacia usuario final. Para los servidores que tengan las funcionalidades suficientes y necesarias para el desarrollo de los pilotos.								(Orientado hacia IDEs *)	(Sólo WMS, va mejor con QGIS)
Número creciente de proyectos presentados en FOSS4GIS y Jornadas SIG Libre – SIGTE entre los años 2009 - 2010		?		?	?			?	?
Existencia de cursos de capacitación en Instituciones/Universidades , en idioma castellano entre los años 2010 – 2011								?	
Total de	6	3	3	3	2	6	5	2	2

= SI
= NO
? = No se registran datos o es indeterminado
*** IDE:** Infraestructura de datos espaciales.

Después de analizar las anteriores posibilidades y cumpliendo con los criterios establecidos, se optó en la parte cliente por OpenLayers, herramienta que cuenta con una madurez de más de 10 años. Sin embargo se necesitaba de un *framework* que soportara la creación de interfaces de usuarios (IU) y que tuviera componentes (*widgets*) para desplegar información geoespacial, por esto se eligió a GeoExt. En la parte del servidor, se escogió Geoserver, ya que se tenía mayor experiencia en su manejo y se tomó en consideración la comparación de [3].

Dentro de los requisitos se considera la preferencia de utilizar Oracle, ya que al ser el motor de bases de datos corporativa, la aplicación, con posibilidades de implantarse en la institución, debe estar acorde con la arquitectura empresarial de EAFIT. Entonces, a pesar de ser una herramienta licenciada, para el piloto con software libre se utilizó la versión gratuita XE (eXpressEdition).

A continuación se presentan las configuraciones de las herramientas seleccionadas tanto libres como propietarias:

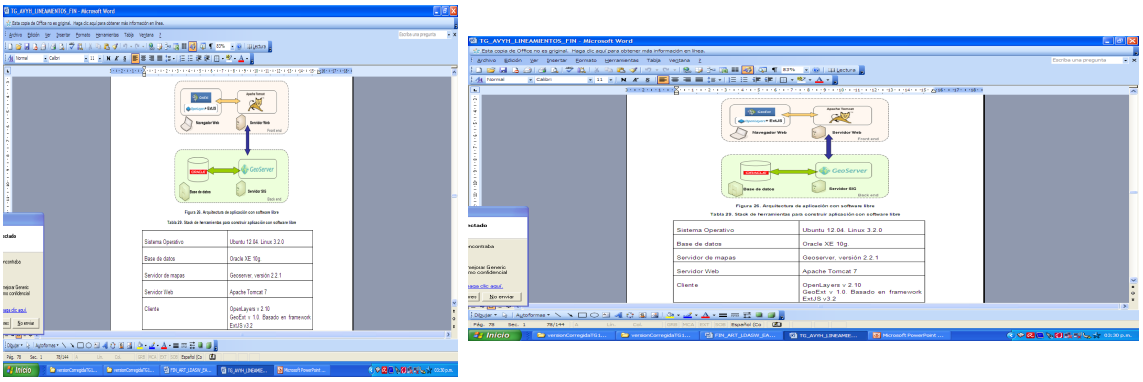

Figura 1. Arquitectura de la aplicación con software libre

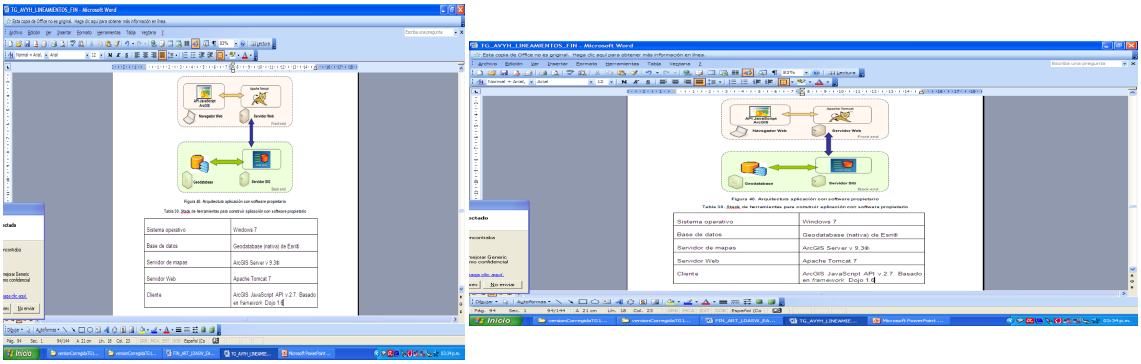

Figura 2. Arquitectura de la aplicación con software propietario

Diseño y construcción de aplicaciones piloto

Para la aplicación con software libre, primero se migraron los datos contenidos en la Geodatabase a OracleXE, para esto se extrajeron en formato *shapefile* las capas necesarias y luego se importaron. La publicación de la información se realizó

adecuando Geoserver mediante un *plugin* que permitiera el trabajo con el motor seleccionado.

Al ser el desarrollo centrado en el cliente, en el que se involucra GeoExt, y que a su vez depende de ExtJS, se buscó una estructura recomendada o sugerida para este f*ramework,* se acogió la organización planteada por [4], en la que sus principales componentes son:

- index.html: página que contiene el código HTML encargado de traer las librerías de ExtJS, los CSS y los archivos JavaScript propios de la aplicación (JS).
- assets: contiene los recursos de imágenes y los archivos de estilo (css).
- lib: contiene las librerías de ExtJS y los JS de la aplicación.
- Main.js, se encuentra en pack1: es el punto de entrada a la aplicación, y de allí se llamará a los otros JS ubicados en los otros paquetes.

La figura a continuación muestra la organización:

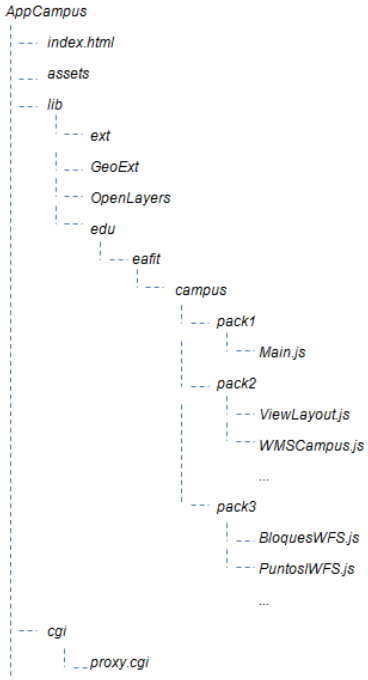

Figura 3. Estructura de la aplicación con software libre

En la estructura se observa la carpeta lib, que contiene el API OpenLayers, GeoExt, Ext y los JS de la aplicación, los cuales están conformados por pack1, pack 2 y pack3.

Pack2: corresponde a las clases que se encargan de la vista (IU) y que controlan las interacciones con el usuario, éstas hacen uso de los *widgets* disponibles en Ext y GeoExt. Se encuentra ViewLayout.js, WMSCampus entre otras.

Pack3: se encuentran las clases que representan el modelo y permiten almacenar los datos que son traídos desde el servidor. Por ejemplo BloquesWFS.js y PuntosIWFS.js

La carpeta CGI contiene un script proxy que permite solicitar URLs a través de AJAX que no están en el mismo dominio. OpenLayers tiene disponible uno hecho en python, que se modifica según los sitios permitidos, vale anotar que este mecanismo es adecuado para aplicaciones sencillas donde no se necesite mucha seguridad.

Para la codificación, se tomó de [4], la forma de crear clases en ExtJS, similar a los POJO (*Plain Old Java Object*). La imagen siguiente muestra un ejemplo:

```
Ext.ns("edu.eafit.campus.pack2"); (function() {
    var WMSCampus = Ext.extend(Object, {
        constructor : function(config) {
            Ext.apply(this, config);
        },
        mapBM : null,
        createMapa : function() {
            //console.log("en el mapa");

            var epsg900913 = new OpenLayers.Projection("EPSG:900913");
            var epsg4326 = new OpenLayers.Projection("EPSG:4326");
            var extent = new OpenLayers.Bounds(-75.58, 6.197, -75.576, 6.203);

            var options = {
                projection : epsg900913,
                displayProjection : epsg4326,
                units : "m",
                numZoomLevels : 40,
                maxResolution : 156543.0339,
                maxExtent : extent
            };

            extent.transform(new OpenLayers.Projection("EPSG:4326"), options.projection);

            var base = new OpenLayers.Layer.Google("Google Streets", {
                'sphericalMercator' : true,
                numZoomLevels : 40,
                maxExtent : extent
            });

            var bloques_eafit = new OpenLayers.Layer.WMS("Bloques EAFIT", "http://localhost:8080/geoserver/ora/wms", {
                layers : "ora:CMFC_BLOQUES_TB",
                transparent : true
            }, {
                isBaseLayer : false,
                visibility : true,
                buffer : 0,
                opacity : 0.9

            });
             mapBM = new OpenLayers.Map();
             mapBM = new OpenLayers.Map(options);
             var layers = [base_sat, base, bloques_eafit, esp_abier];

             mapBM.addLayers(layers);

             console.log("layers! " + mapBM.getNumLayers());

        },
        getMapa : function() {
            return mapBM;
        },
        setMapa : function(mp) {
            this.mapBM = mp;
        }
    })

    edu.eafit.campus.pack2.WMSCampus = WMSCampus;
})();
```

Figura 4. Ejemplo de código fuente con GeoExt

La imagen da cuenta de dos de las funcionalidades desarrolladas:

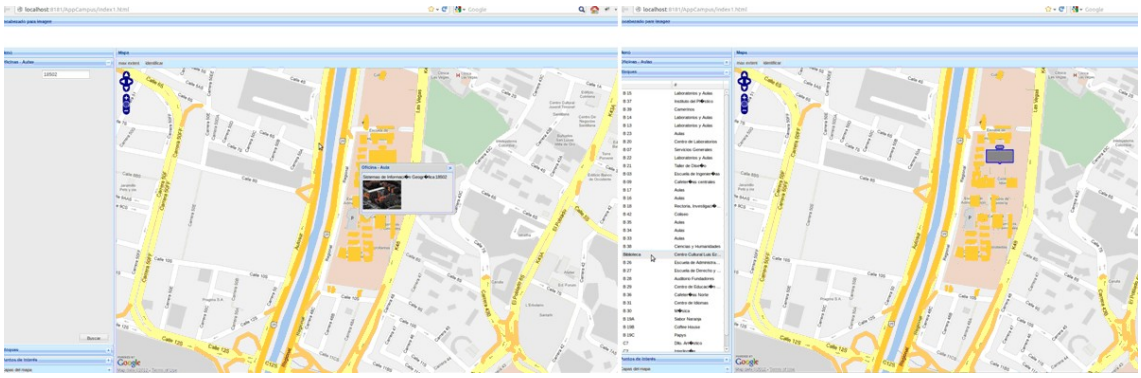

Figura 5. Funcionalidad de buscar oficina/aula y localizar bloque.

Para la aplicación con software propietario, por facilidad de manejo se eligió trabajar con la Geodatabase y publicarla con ArcGIS Server.

Para el desarrollo del cliente, se empleó el API JavaScript de ArcGIS, el cual depende de Dojo, la organización básica para una aplicación con este *framework* es la siguiente [5]:

Figura 6. Estructura básica de una aplicación con Dojo.

Los componentes de la aplicación piloto son:

- index.html: página que contiene los llamados al API desde el sitio que ESRI tiene dispuesto para esto (http://serverapi.arcgisonline.com). Y se establece el *layout* mediante los widgets de Dijit (i.e. BorderContainer, ContentPane).
- css: contiene los archivos de estilo.
- images: contiene las imágenes utilizadas en la aplicación.
- js: contiene los archivos JavaScript propios de la aplicación. Dentro los JS se encuentran base.js, que es una clase controladora, que llama a las otras clases encargadas de funcionalidades como identificar, mapa (que trae la capa de base y la de bloques), entre otras.

Un ejemplo de clase hecha con Dojo se muestra a continuación:

```
dojo.provide('app.busqueda2');

dojo.declare("app.busqueda2", null, {

    doBuscar : function(mapa, txt) {

        mapa.graphics.clear();
        queryTask = new esri.tasks.QueryTask("http://b18p5sig8:8399/arcgis/rest/services/CampusSIGEAFIT/mapaEAFITcampus_fullLayers/MapServer/184");
        query = new esri.tasks.Query();
        query.text = txt;
        query.returnGeometry = true;
        query.outFields = ["OBJECTID", "BLQ_NOMBRE", "BLQ_NOMBRE2"];
        //query.where = "BLQ_NOMBRE ='" + txt + "'";
        query.where = "BLQ_NOMBRE LIKE '%" + txt + "%'";
        queryTask.execute(query, function(results) {

            /*  for (var i=0, il=results.features.length; i<il; i++) {
            var featureAttributes = results.features[i].attributes;
            for (att in featureAttributes) {
            console.log(" " + att + " " + featureAttributes[att] + "");
            }
            }  */
            //build an array of attributes
            var items = dojo.map(results.features, function(feature) {
                console.log(feature.attributes);
                return feature.attributes;
            });
            var data = {
                identifier : "OBJECTID",
                items : items
            };
            store = new dojo.data.ItemFileReadStore({
                data : data
            });
            grid.setStore(store);
            grid.setSortIndex(1, "true");
            //sort on the state name
            dojo.forEach(results.features, function(feature) {
                mapa.graphics.add(feature);
            });
        });
    }
});
```

Figura 7. Ejemplo de código fuente con API ArcGIS JavaScript

La imagen muestra dos de las funcionalidades desarrolladas:

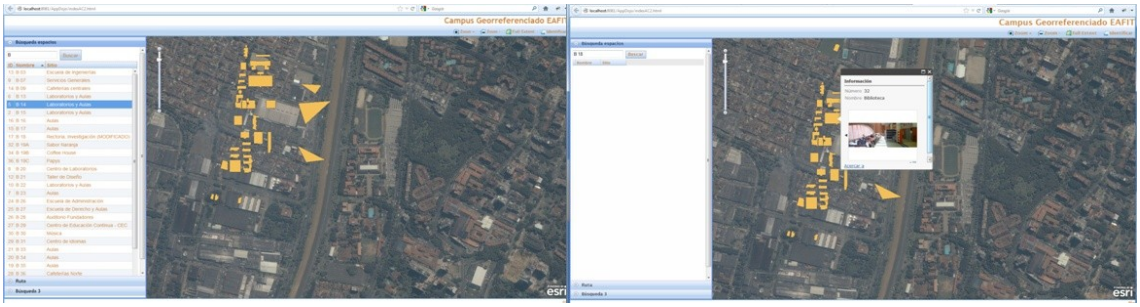

Figura 8. Funcionalidad de buscar bloques e identificar.

LINEAMIENTOS PARA EL DESARROLLO DE APLICACIONES SIG WEB

Los lineamientos se establecen en tres etapas: antes, durante y después del proyecto:

Antes

- Dimensionar la magnitud del desarrollo, en términos de funcionalidad: hay diferencia entre una aplicación sencilla en la que sólo se desea embeber un mapa en una página Web -con tareas básicas como tener controles de zoom, paneo o listar capas- a crear una en la que sean múltiples funcionalidades de mediana - alta complejidad (i.e. trazado de rutas, generación de nueva información geográfica, consultas avanzadas). En el primer caso, por ejemplo, el uso de la librería OpenLayers es suficiente; en el segundo, el trabajo debe ser más cuidadoso, se debe tener una arquitectura, emplear patrones y *frameworks*, que faciliten el desarrollo, trabajo en equipo y reuso de código. Sin embargo, por simple que parezca la aplicación, se puede aplicar la estructura más básica que consiste en separar HTML, estilos y código, esto da pie a que continúe su crecimiento.
- Tomar conciencia que el desarrollo de aplicaciones del lado del cliente tiene el mismo rigor que las tradicionales del lado del servidor (en Java o .Net), en especial cuando se utiliza JavaScript. Se debe evitar el crecimiento desordenado o código spaghetti.
- Preparar la conformación del equipo de trabajo, el cual debe contar con varios perfiles de dos áreas de conocimiento, de parte de la rama de los SIG y de la ingeniería. Los roles que se pueden establecer son los siguientes: director del proyecto, profesionales SIG, arquitecto, diseñador de interfaz y de experiencia de usuario, desarrolladores, integrador(es), probador(es).
- Evaluar si el equipo de trabajo tiene los conocimientos necesarios, en especial si es la primera vez que se enfrenta un desarrollo de este tipo. Es importante para definir si se establece un plan de capacitaciones y en qué áreas enfocarse, por ejemplo en patrones, *frameworks*, manejo de datos geográficos/geodesia o metodologías. Es deseable que los desarrolladores tengan experiencia con HTML, CSS, JavaScript básico y les sea familiar la programación orientada a objetos. Los cursos cortos, ya sean presenciales o virtuales son una opción para instruir al personal e incentivarlo para que utilicen buenas prácticas y eviten errores comunes, los recursos que demanda esta tarea se ven compensados con un proceso de desarrollo más limpio y quienes están acostumbrados a programar asimilan con facilidad este tipo de temáticas.

- Revisar las metodologías que se ajusten a las características del proyecto (tiempo, presupuesto, personal). Una buena alternativa son las ágiles, ya que se adaptan a este tipo de aplicaciones que requieren que se muestren resultados rápidamente.
- Hacer un estudio o barrido sobre las tecnologías vigentes, esto es de especial importancia ya que ayuda a mitigar riesgos en el desarrollo y a mejorar las perspectivas de extensibilidad y mantenibilidad. Las consultas pueden hacerse en blogs, publicaciones especializadas y en las propias páginas de las herramientas, también es de utilidad asistir a jornadas, encuentros o presentaciones académicas y comerciales de las herramientas. Factores que pueden tenerse en cuenta son:
  - Fecha del último *release*: indica si está actualizado o está en vía de ser abandonado.
  - Tiempo entre cada *release*: muestra el grado de trabajo y evolución que hay en el proyecto
  - Fecha prevista de lanzamiento de la próxima versión: si es pronto, quizá es prudente esperar a las últimas mejoras que utilizar la corriente.
  - Tipo de licenciamiento: si es de código abierto o propietario y condiciones especiales.
  - Dependencia de otros *frameworks*, incluyendo su influencia directa sobre el tipo de licencia, el panorama de cambio, ya que algunos pueden presentar saltos abruptos en el manejo. Es importante analizar si los f*rameworks* proponen una aplicación de referencia (o mecanismo similar a los *blueprints* de Java).
  - Lenguaje que utiliza: apunta al grado de madurez de la tecnología y recepción por parte de la comunidad desarrolladora.
  - Propietario u organización que lidera el desarrollo: da idea del soporte que brinda.
  - Actividad que tiene en los foros o listas de correo: esto señala el grado de acogida entre la comunidad desarrolladora y si es fácil conseguir ayuda sobre la herramienta.
  - Participación en encuentros de orden mundial, regional o local en los que se promocione su uso: pone de manifiesto la aceptación y uso que está teniendo en la comunidad, además del nivel de difusión de la herramienta.
  - Cantidad y características de proyectos que se hayan desarrollado con las herramientas (i.e. gubernamentales, de propósito específico o único, complejidad de las funcionalidades implementadas).
  - Unido al anterior lineamiento, si se tienen los recursos disponibles, es oportuno realizar pruebas de concepto o pilotos, o utilizar demos de productos, que permitan dar un panorama global del manejo de las posibles tecnologías a emplear y los conceptos que manejan.
  - Una excelente aproximación a las tecnologías y a los conceptos a tener en cuenta, es probar aplicaciones desarrolladas bajo las herramientas candidatas a utilizar, es decir, actuar como usuario, de esta manera se aprecia la parte gráfica o visual, la velocidad de respuesta, usabilidad, tipo de funcionalidades implementadas, entre otras particularidades.
  - Es recomendable que asista a eventos de orden académico y comercial, relacionados con esta temática, de esta manera puede interactuar con desarrolladores, proveedores de tecnología y usuarios. Estos espacios son valiosos ya que puede presentar inquietudes y conocer de primera mano las experiencias de otros proyectos.
  - Si se tiene un cliente potencia, se debe indagar cuáles son las políticas sobre el uso de software de código abierto y propietario, ya que esto restringe las opciones en cuanto a tecnologías.

Durante

- Mantener siempre presente el tipo de usuario a quien va dirigida la aplicación y diferenciarlo del rol del cliente, quien solicita el desarrollo del producto y tiene mayor conocimiento de los requisitos, mientras que el usuario es la persona que interactúa directamente con la aplicación. El contenido y los mapas que se brindan deben ser útiles e interesantes, de manera que cautiven la atención del usuario. El éxito de una aplicación además de estar mediada por el cumplimiento de los requisitos, está relacionada con el provecho que las personas saquen de ella, y en algunos casos por el crecimiento en número de usuarios. Priorizar las funcionalidades a implementar: aquellas que tengan valor significativo para el cliente deben ocupar los primeros lugares.
- Analizar qué calidades sistémicas prevalecerán sobre otras, ya que esto impacta directamente a la arquitectura, por ejemplo, si se requiere de altos niveles de seguridad, el *performance* se ve afectado; si la disponibilidad es de 7 X 24, se deben contar con mecanismos de redundancia; si el público a quien se dirige la aplicación es no experto, la usabilidad y la experiencia de usuario deben primar.
- Conformar un grupo para la selección de tecnologías, entre los integrantes deben figurar el director y el arquitecto. Las condiciones del proyecto, la experiencia y las pruebas de concepto son determinantes en este proceso.
- El punto de partida para la aplicación puede ser un esquema de GUI (Interfaz gráfica de usuario).
- Configurar el ambiente de desarrollo, de forma que sea un estándar para todo el equipo de trabajo.
- Dado que la información que se brinde a través del mapa debe estar vigente y ser de interés, se debe generar un plan para la actualización de datos, que contemple la frecuencia, la entidad responsable (empresa desarrolladora, cliente o compartida) y costos aproximados.

Los siguientes lineamientos se dirigen hacia los desarrolladores:

- Suscribirse a foros y listas de correo es una buena opción tanto para buscar ayuda como para colaborar con otros programadores (i.e. lists.osgeo.org).
- Iniciar buscando los ejemplos con funcionalidad similar a la deseada, si no se encuentra, pasar al trac y/o a la lista de correo y una vez que se tenga una idea de la línea a tomar apoyarse en la documentación del API.
- Es fundamental que para las tareas de depuración cuente con una herramienta, dado que son aplicaciones del lado cliente, el navegador sobre el que está probando debe incluirla. Es útil tanto para ubicar errores como para hacer pruebas de rendimiento, al medir cuánto tiempo se demora una función (o trozo de código).
- Hacer de las buenas prácticas una rutina diaria, de manera que se interioricen y se conviertan en hábitos, por ejemplo: evitar variables globales y anidamientos excesivos; optimizar bucles; asignar nombres - ojalá cortos- a las variables y funciones que se auto expliquen; los comentarios deben ser los necesarios; utilizar un analizador de código para comprobar la calidad sintáctica, esto permite identificar problemas potenciales, ahorrando tiempo en el proceso de depuración y mejorando la integración con los productos de otros desarrolladores.
- La minimización y ofuscación de código son alternativas que se deben tener presentes para optimizar tiempos de carga.

Después

Estos lineamientos conducen a la evaluación del proyecto, donde a partir de la experiencia y aprendizaje, se establezcan diagnósticos y se propongan opciones de

cambio y mejora para próximos proyectos. Se encuentran enfocados desde tres puntos:

El software creado en sí mismo:

- Grado de reutilización: cuántos módulos pueden emplearse en construir otra aplicación (con características similares).
- Cuál es el componente que es más susceptible a presentar fallos, para proponer alternativas que mitiguen los riesgos.
- Cuantificar tiempos de respuesta, en especial cuando hay picos de peticiones, se debe considerar tanto del *backend* como el *frontend*.

Respecto al cliente / usuario:

- Medir el grado de satisfacción del cliente: donde éste califique cuantitativa y cualitativamente el trabajo realizado, en aspectos que van desde el cumplimiento de tiempos, receptividad ante adiciones y cambios propuestos en las funcionalidades, metodología empleada, entre otros.

- Medir el grado de utilización por parte de los usuarios: saber cuántas personas ingresan a la aplicación; poner a disposición en la misma página una encuesta breve sobre si le gusta y un espacio para que manifiesten ideas de ampliar la funcionalidad o para presentar inconformidades sobre el aplicativo. Esta perspectiva permite obtener un panorama de la evolución del software.

Desde el equipo de desarrollo:

- Obtener y analizar percepciones sobre la metodología de desarrollo empleada, de modo que se propongan ajustes y se apliquen para los proyectos siguientes.
- Grado de dominio de tecnologías, curva de aprendizaje, preguntar si en un próximo proyecto se emplearían las conocidas o se optarían por otras.
- Examinar si el ambiente de desarrollo, el sistema de control de versiones y demás herramientas fueron efectivos y eficaces. Listar posibles tecnologías sustitutivas y complementarias.

CONCLUSIONES

- Los lineamientos parten de una base teórica, recogen y abstraen las lecciones derivadas de un proceso práctico, independizándolos de herramientas o plataformas particulares (agnósticos), de modo que pueden utilizarse en otro tiempo y circunstancias. Sin embargo, las tecnologías por su rápida y constante evolución pueden desembocar cambios que modifiquen o supriman algunos lineamientos de los propuestos, lo cual les otorga una naturaleza dinámica.
- Las aplicaciones piloto constituyeron el medio y no el fin en el proceso de estructurar los lineamientos, su construcción fue una buena forma de vivenciar el proceso de desarrollo de software, con los inconvenientes en cada una de las etapas, desde los requisitos hasta la implementación, permitiendo tener varios panoramas, desde la visión a gran escala de la arquitectura hasta los detalles de la codificación.
- Toda herramienta tecnológica (aplicación, plataforma, *framework*) tiene pros y contras, lo importante es hacer el balance y determinar cuál de todas las que se encuentran disponibles es la que mejor se adapta a las necesidades y brinda más y mejores beneficios a un proyecto en particular. En muchos casos es posible utilizar híbridos entre tecnologías libres y propietarias, debido al uso de servicios y a la estandarización, tal como se demostró en la construcción de las aplicaciones piloto.

REFERENCIAS

[1] Valencia A., A.M. (2010). "E*strategia de integración de una base de datos georreferenciada con los sistemas corporativos*". Proyecto de grado Ingeniería de Sistemas. Universidad EAFIT.

[2] TIOBE Software. The Coding standards company http://www.tiobe.com/index.php/content/company/Home.html

[3] Graser, A. (2010). “Geoserver vs. Mapserver”. Disponible en: http://underdark.wordpress.com/2010/06/08/geoserver-vs-mapserver/

[4] Chandan, R. (2011). "*Sencha / ExtJS : Object Oriented JavaScript*". Disponible en: http://www.slideshare.net/rohan.chandane/sencha-extjs-object-oriented-javascript

[5] SitePen http://www.sitepen.com/about/index.html