Universidad de las Ciencias Informáticas

FACULTAD 6

Título: "Biblioteca JavaScript para el desarrollo de interfaces gráficas de usuario de RIA".

Trabajo de Diploma para optar por el título de

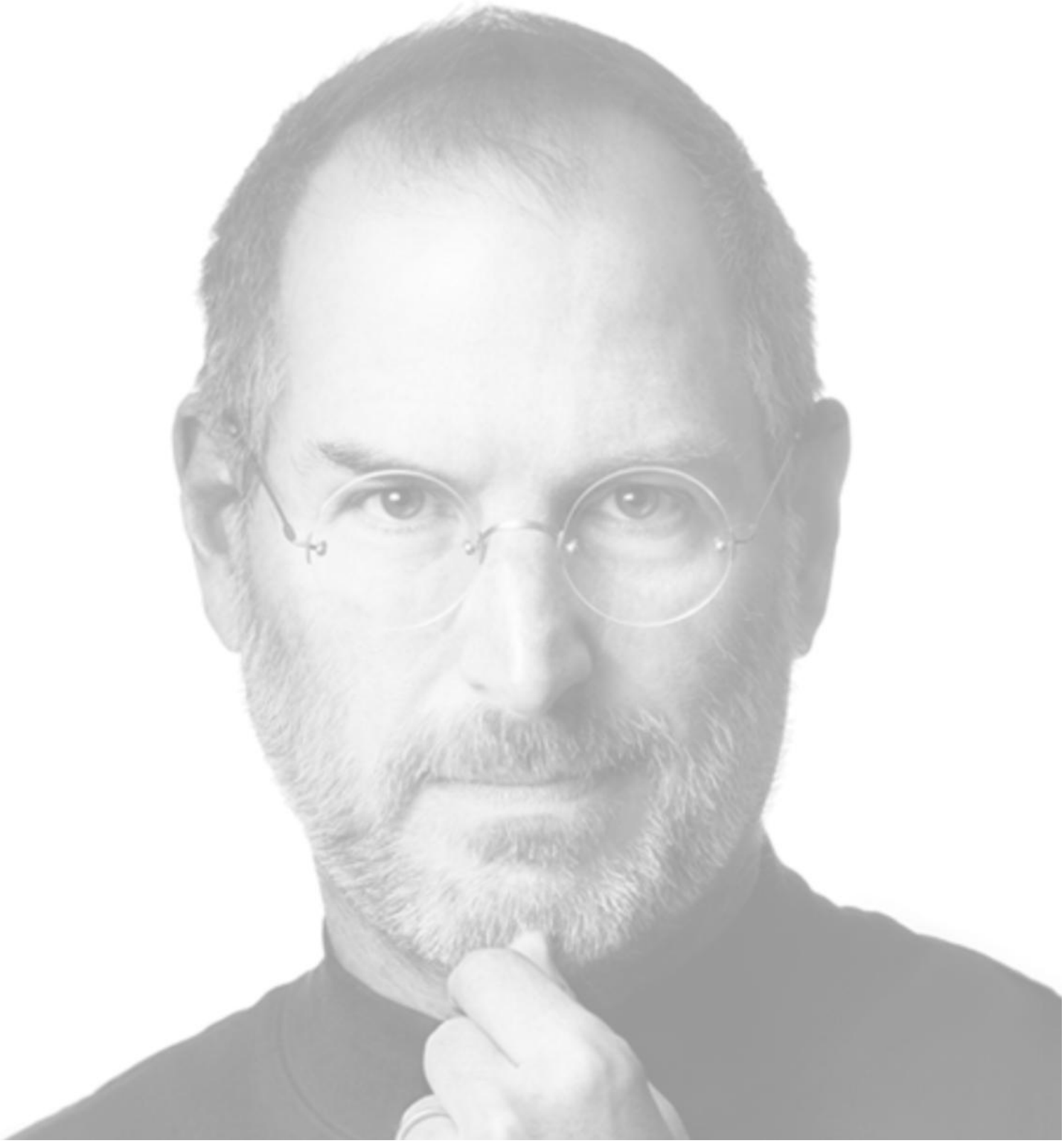
Ingeniero en Ciencias Informáticas

Autores: José Roberto Guzmán Ojeda.
Reisel Betancourt Santana.

Tutores: Ing. Armando Robert Lobo.
Ing. Yunier Vega Rodríguez.

Cotutor: Ing. Yoander Iñiguez Bermúdez.

La Habana, Junio de 2013
"Año 55 de la Revolución"



“LA GENTE QUE ESTÁ LO SUFICIENTEMENTE LOCA COMO PARA PENSAR QUE PUEDEN CAMBIAR EL MUNDO, SON LAS QUE LOGRAN HACERLO”

STEVE JOBS

1955-2013

DECLARACIÓN DE AUTORÍA

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

Para que así conste firman la presente, a los ____ días del mes de _____ del año _____.

José Roberto Guzmán Ojeda

_____

Firma del Autor

Reisel Betancourt Santana

_____

Firma del Autor

Ing. Armando Robert Lobo

_____

Firma del Tutor

Ing. Yunier Vega Rodríguez

_____

Firma del Tutor

Ing. Yoander Iñiguez Bermúdez.

_____

Firma del Cotutor

DATOS DE CONTACTO

Ing. Armando Robert Lobo
Ing. en Ciencias Informáticas, UCI 2009. Especialista General.
Email: arobert@uci.cu

Ing. Yunier Vega Rodríguez
Ing. en Ciencias Informáticas, UCI 2009. Especialista Superior.
Email: yvrodriguez@uci.cu

Ing. Yoander Iñiguez Bermúdez
Ing. en Ciencias Informáticas, UCI 2010. Especialista General.
Email: yiniguez@uci.cu

AGRADECIMIENTOS

A la Revolución por darme la oportunidad de estudiar y formarme como ingeniero en la UCI.

A mis padres por el apoyo incondicional y la confianza brindada durante mis años de estudio.

A mi hermano por el cariño y la preocupación.

A mis abuelos por preocuparse por mí en estos años de estudio.

A mis tíos por ser como padres y estar pendientes de mí en todo momento.

A mis primos por ser como hermanos y preocuparse por mí durante la carrera.

A mis nuevos hermanos Jose, Ernesto y Antonio por la amistad sincera y su presencia en cada momento de estos cinco años.

A Jose nuevamente por ser el mejor dúo de tesis.

A Chela, Maricel, Claudia, Sucel y Salvador por ser excelentes amigos.

A los profes Yanet, Glennis, Garnache, Jorge Luis y Yunier por el apoyo durante estos años.

A todos mis compañeros de aula y de apartamento, en especial al piquete de la farándula y al grupo de Rock and Roll.

A demás familiares, profesores y compañeros que de una forma u otra han estado presentes en estos largos años de estudio. Gracias a todos.

Reisel

A mis padres Roberto y Leodigna, gracias por enseñarme a ser el hombre que hoy soy.

A mis abuelos Flor y Pablo por guiarme y apoyarme en mis sueños.

A mi niñas Laura y Amanda, por las piñatas, el amor y la alegría que me transmiten.

A mi abuela Dalia por la preocupación y el apoyo.

A Chelin, por estar junto a mí en los buenos y malos momentos y todo el amor que me ha dado.

A mis suegros y cómplices Anita y Sergito.

A tía Caridad por estar siempre pendiente.

A mis tíos Armando y Leoarmis: ustedes sembraron la semilla.

A los grandes: Yaikiel Hernández, Maikel Zuñiga, Alberto Garnache y Deivis Álvarez.

A los mejores amigos que la Universidad me dio: Reisel, Antonio y Ernesto.

A Salvador, Marisel, Claudia y Susel, otras de las buenas adquisiciones; muchas gracias por su ayuda.

Jose

DEDICATORIA

De Reisel

A Isel y Reinol por ser los mejores padres del mundo, sabiendo guiarme y apoyarme durante toda la vida y en especial estos cinco años de universidad.

Gracias por todo.

De Jose

A mi Titi, por todas y cada una de las lecciones, incluyendo aquella primera que nunca olvidaré.

RESUMEN

La continua expansión de Internet ha posibilitado a muchas empresas utilizar la Web como plataforma para el desarrollo y despliegue de sus productos y servicios. Debido al empleo de AJAX, así como a la combinación de CSS, HTML y JavaScript a través de las bibliotecas de componentes visuales o *widgets toolkits*, las aplicaciones web han logrado tener gran aceptación por los usuarios finales. En la Universidad de las Ciencias Informáticas, y más específicamente en el departamento Integración de Soluciones perteneciente al Centro de Tecnologías de Gestión de Datos, la comercialización de este tipo de aplicaciones se encuentra limitada.

Las aplicaciones enriquecidas de Internet desarrolladas por el Departamento utilizan la biblioteca Ext JS, la cual a pesar de brindar facilidades para la creación de interfaces gráficas, emplea un esquema de licenciamiento dual como modelo de negocio que dificulta la comercialización del producto final, atentando contra la soberanía tecnológica del país. Ante esta problemática se decide construir una biblioteca JavaScript para el desarrollo de interfaces gráficas de usuario utilizando como plataforma jQuery UI y añadiéndole una capa de abstracción que redefine cada uno de sus *widgets* y los agrupa en una nueva jerarquía. Esta capa brinda soporte a la Programación Orientada a Objetos con la posibilidad de instanciar y manipular cada uno de los *widgets*, así como mecanismos que permiten al desarrollador definir otros nuevos.

Palabras claves: Aplicaciones enriquecidas de Internet, aplicaciones web, bibliotecas, *widget toolkits*.

ÍNDICE DE CONTENIDOS

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

INTRODUCCIÓN

El éxito o fracaso de un producto informático está condicionado por los niveles de aceptación que tenga sobre sus usuarios. Hoy día, la facilidad de uso es una de las pautas fundamentales que todo diseñador o desarrollador prioriza, pues muchas veces los productos con más y mejores funcionalidades no son los que mayor aceptación tienen, sino los que por su usabilidad permiten al usuario alcanzar sus objetivos, desde compartir en redes sociales hasta encontrar y procesar información.

Las aplicaciones web no se encuentran exentas de este principio, por lo que la creación de interfaces gráficas con la usabilidad y funcionalidad requerida por el usuario final, es fundamental en el desarrollo de las mismas. ¿Qué tienen *Facebook*, *Flickr*, *Gmail*, *Tweeter* y *SkyDrive*, algunas de las más populares aplicaciones de la llamada *Web 2.0*, en común? Todas ellas ofrecen enriquecidas interfaces gráficas haciendo uso de tres piezas claramente distinguibles, el contenido: HTML (*Hypertext Markup Languaje*), la presentación: CSS (*Cascade Style Sheet*) y el comportamiento: JavaScript.

Aún cuando las tecnologías anteriores no sean recientes, han tomado mayor auge en los últimos años, debido en gran medida a la necesidad de proveer a la Web de una *experiencia de usuario*[1] semejante a la que proporcionan las aplicaciones de escritorio, pues aunque esto fuese posible utilizando otras tecnologías como *Flash* o *Silverlight*, tienen la desventaja de necesitar extensiones en el navegador que pudiesen no estar disponibles. Tal limitante ha impulsado el desarrollo de las bibliotecas JavaScript, herramientas que en la actualidad poseen diversos usos, desde los más simples como la manipulación del DOM (*Document Object Model*) y la realización de animaciones a través de las bibliotecas de uso específico; hasta agilizar y flexibilizar el desarrollo de las aplicaciones enriquecidas de Internet (RIA, por sus siglas en inglés) con las bibliotecas de componentes visuales.

Las bibliotecas JavaScript juegan entonces un rol determinante en la construcción de las interfaces gráficas de usuario o GUI (*Graphical User Interface*) de las RIA, no solo porque aceleran el proceso de desarrollo de las mismas, sino porque además permiten reutilizar código ya existente, promover buenas prácticas de implementación, resolver los problemas de compatibilidad entre navegadores, así como facilitar la interacción cliente-servidor.

[1] Conjunto de factores y elementos relativos a la interacción del usuario con un entorno o dispositivo concreto, cuyo resultado es la generación de una percepción positiva del mismo.

En este contexto y con la finalidad de desarrollar bienes y servicios informáticos relacionados con las tecnologías de gestión de datos, el Centro de Tecnologías de Gestión de Datos (DATEC) perteneciente a la Facultad 6 de la Universidad de las Ciencias Informáticas (UCI) y más específicamente el departamento de Integración de Soluciones (IS), utiliza para la creación de la GUI de sus productos la biblioteca Ext JS.

Esta biblioteca emplea un esquema de licenciamiento dual como modelo de negocio, del cual se derivan una versión comercial y otra comunitaria con licencia GPL (*General Public Licence*) versión 3. A causa de la imposibilidad de adquirir la versión de pago, el departamento de IS se ha visto en la necesidad de utilizar la versión comunitaria, con menos prestaciones y restricciones legales en cuanto a la redistribución del software modificado; tales restricciones establecen que el producto final debe entregarse bajo este mismo tipo de licencia, impidiendo comercializar bajo el esquema más conveniente la GUI de los productos, al permitirle al cliente modificar y re-comercializar la misma, todo lo cual atenta contra la soberanía tecnológica del país.

Partiendo de la situación antes descrita se plantea como **problema de la investigación**: ¿cómo contribuir a la soberanía tecnológica de las interfaces gráficas de usuario de las aplicaciones enriquecidas de Internet que se desarrollan en el departamento Integración de Soluciones del Centro de Tecnologías de Gestión de Datos?

Se define entonces como **objeto de estudio**: bibliotecas para el desarrollo de interfaces gráficas de usuario, y como **campo de acción**: biblioteca para el desarrollo de interfaces gráficas de usuario de aplicaciones enriquecidas de Internet.

Para dar solución al problema de la investigación planteado se propone como **objetivo general**: desarrollar una biblioteca JavaScript para el diseño de interfaces gráficas de usuario de aplicaciones enriquecidas de Internet.

El objetivo general anteriormente expuesto se desglosa en los siguientes **objetivos específicos**:

- Elaborar el marco teórico referencial sobre el tema.
- Realizar el análisis de la biblioteca.
- Realizar el diseño de la biblioteca.
- Implementar la biblioteca diseñada.

- Validar la biblioteca propuesta.

Para dar cumplimiento a los objetivos planteados se realizarán las siguientes **tareas de la investigación**:

- Revisión bibliográfica de las bibliotecas para el diseño de interfaces gráficas de usuario desarrolladas con JavaScript y orientadas a la Web.
- Realización de análisis de las bibliotecas para el desarrollo de interfaces gráficas de usuario de aplicaciones enriquecidas de Internet.
- Descripción de las herramientas y tecnologías seleccionadas para el desarrollo de la biblioteca.
- Identificación de los requisitos funcionales y no funcionales.
- Definición de la arquitectura de la biblioteca.
- Confección de las Tarjetas CRC (*Class – Responsability - Colaboration*).
- Definición de los estándares de codificación.
- Implementación de los componentes que conforman la biblioteca.
- Realización de pruebas unitarias y de aceptación a la biblioteca.

El presente trabajo consta de Introducción, capítulos enumerados del uno al tres, Conclusiones, Recomendaciones, Referencias bibliográficas, Bibliografía, Anexos y Glosario. A continuación se describe el contenido de cada uno de los capítulos.

Capítulo I. Fundamentación teórica: En este capítulo se analizan los elementos fundamentales para el entendimiento del dominio del problema. Incluye la identificación y descripción de las principales bibliotecas para el desarrollo de GUIs y sus capacidades. Se realiza además un análisis de la metodología de desarrollo, tecnologías y herramientas a utilizar en la construcción de la solución.

Capítulo II. Planeación y Diseño: En este capítulo se exponen las características y funcionalidades de los componentes que constituyen la biblioteca, se capturan los requisitos funcionales y no funcionales de la misma, además de detallarse la arquitectura y patrones presentes en el desarrollo. Finalmente, se realiza el diseño de la solución a través de las Tarjetas CRC.

Capítulo III. Implementación y Pruebas: En este capítulo son definidos los estándares de codificación y Tareas de Ingeniería (TI) asociadas a cada Historia de Usuario identificada. Además, se realiza la

implementación de la biblioteca así como las distintas pruebas necesitadas para su validación, acorde a la metodología de desarrollo seleccionada.

CAPÍTULO 1. FUNDAMENTACIÓN TEÓRICA

A continuación se expone un análisis sobre los principales conceptos relacionados con las bibliotecas JavaScript y su ámbito. Se presenta a la vez un acercamiento a la situación actual de estas, haciendo énfasis en las características, ventajas e insuficiencias que las distinguen. Además, se realiza una breve descripción de la metodología y herramientas de desarrollo a utilizar durante la construcción de la solución.

1.1 Interfaz Gráfica de Usuario

La GUI constituye una pieza fundamental de cualquier producto de software. Su surgimiento se remonta al año 1973, cuando fueron realizados los primeros esbozos de lo que a la postre serían las primeras GUIs.

Carlos Marrero, en su investigación "Interfaz gráfica de usuario. Aproximación semiótica y cognitiva" define a la GUI como *"el artefacto interactivo, que por su diseño y a través de ciertos interfaces humanos, posibilita la interacción de una persona con un sistema informático, haciendo uso de las gramáticas visuales y verbales (signos gráficos como iconos, botones, menús y verbales como tipografía)"* [1].

Para la mayoría de los usuarios la interfaz es la aplicación, debido en gran medida a que es la parte que ven y a través de la cual interactúan, siendo así, la GUI es el medio para evaluar un sistema y la causa de que estos sean descartados, incluso sin considerar las funcionalidades que brindan. En tal sentido, y con el propósito de facilitar y agilizar la creación de este tipo de interfaces las bibliotecas de componentes visuales juegan un importante papel.

1.2 Bibliotecas de componentes visuales

En informática, las bibliotecas (del inglés *libraries*) son definidas como "*una abstracción de código que provee funcionalidades genéricas que pueden ser utilizadas para desarrollar aplicaciones de manera rápida, fácil, modular y sencilla, ahorrando tiempo y esfuerzo*" [2].

Extrapolando este concepto, puede inferirse que una biblioteca de componentes visuales facilita la creación de interfaces gráficas al incorporar *widgets* que mejoran la experiencia de usuario. Los *widgets* pueden clasificarse en dos grupos, los de interacción tales como botones, etiquetas, casillas de

verificación, entre otros y los de agrupamiento, que actúan como contenedores de otros *widgets*, ejemplos de estos son las ventanas y paneles.

En sus inicios, el uso de este tipo de herramientas se encontraba limitado solo a las aplicaciones de escritorio, sin embargo el continuo desarrollo de Internet y su ascenso como plataforma preferida para el desarrollo de aplicaciones distribuidas, propiciaron la aparición de herramientas que permitían crear aplicaciones web con características de aplicaciones de escritorio.

1.3 Aplicaciones enriquecidas de Internet

Las RIA surgen con el propósito de dar respuesta al desafío que implica la fusión entre las aplicaciones web y las de escritorio, para así mejorar la experiencia y productividad del usuario. La expresión fue acuñada en 2002 por *Macromedia Inc*.

En términos formales las RIA son definidas como *"la combinación de lo mejor de la funcionalidad de UI (User Interface) de una aplicación desktop, con el amplio alcance y el bajo costo de deployment[2] de una aplicación web, y lo mejor de las comunicaciones interactivas."* [3].

Las RIA mejoran la usabilidad de una aplicación web al incorporar los componentes y funcionalidades que brindan las aplicaciones de escritorio, tales como copiar, cortar y pegar, redimensionar columnas y ordenarlas; unido a la flexibilidad de presentación y despliegue inherentes a la Web.

La fluida comunicación entre el cliente y el servidor se garantiza al usar AJAX (del inglés *Asynchronous JavaScript And XML*). Esta tecnología brinda a las RIA la interactividad y rendimiento típico de las aplicaciones de escritorio mediante el empleo de una técnica, en la cual, utilizando el objeto *XMLHttpRequest* las peticiones HTTP[3] se envían de forma asíncrona al servidor. Este tipo de peticiones posibilita que el usuario pueda seguir interactuando con otros componentes de la aplicación, mientras anteriores pedidos son atendidos, pues una función en el cliente conocida como *call-back function* es la encargada de procesar la respuesta del servidor y reflejar los cambios en la página.

[2] Despliegue; procedimiento necesario para la puesta en marcha de una aplicación.
[3] *HTTPRequest,* mecanismo utilizado por una página web para realizar pedidos al servidor y recibir respuesta por parte de este.

La siguiente figura muestra mediante una serie de pasos este proceso, gracias al cual la interacción del usuario con la aplicación no se ve interrumpida a causa de recargas o largas esperas por la respuesta del servidor (Figura # 1).

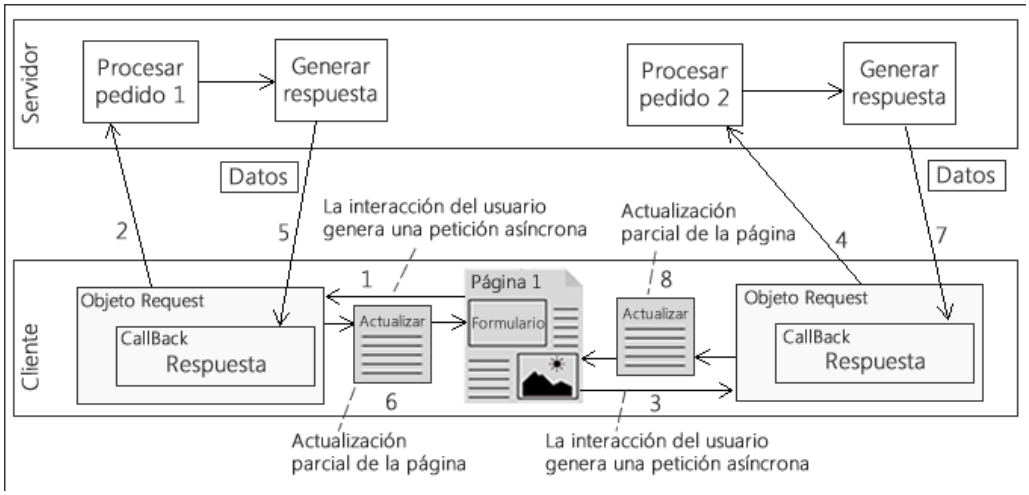


Figura # 1. Escenario de interacción de una RIA.

1.3.1 Compatibilidad Web

Como cualquier otro contenido alojado en Internet, las RIA dependen de los navegadores web para su visualización y ejecución, sin embargo la existencia de varios de estos programas y la violación de los estándares definidos trajeron consigo que la experiencia de usuario fuese un completo caos. Algunos de los problemas enfrentados por los usuarios eran:

- Vistas incompletas de las interfaces.
- Diseños desajustados que no cabían en pantalla.
- Gestión incorrecta de los eventos asociados a componentes HTML.
- Scripts interpretados de manera incoherente.

No era solamente el usuario final quien se veía afectado a causa de esta falta de compatibilidad, los desarrolladores pasaban horas tratando de garantizar la correcta visualización de sus aplicaciones en los distintos navegadores, muchas veces sin éxito. Tal dificultad, unida al hecho del aumento de la complejidad de las aplicaciones y la necesidad de agilizar su construcción, provocaría el surgimiento de un nuevo tipo de herramientas: las bibliotecas JavaScript.

1.4 Bibliotecas JavaScript

Las bibliotecas JavaScript facilitan el manejo de las funcionalidades provistas por los navegadores y ayudan a cerrar la brecha de incompatibilidad existente entre los mismos. En la actualidad el uso de estas bibliotecas ha ganado gran aceptación, no solo por desarrolladores independientes, sino también por las grandes compañías que las utilizan para añadir dinamismo a sus sitios y desarrollar sus aplicaciones.

Básicamente, una biblioteca JavaScript es un archivo o conjunto de archivos que al ser incluidos en una página web, proporcionan, por medio de funciones y utilidades implementadas en su interior, facilidades de desarrollo al programador. Algunas de estas facilidades son:

Compatibilidad: Agregan la posibilidad de escribir código JavaScript totalmente compatible con todos los navegadores.

Comunicación asíncrona: Brindan facilidades para utilizar el objeto *XMLHttpRequest* con el objetivo de realizar llamadas asíncronas al servidor.

Manipulación del DOM: Maximizan la capacidad de agregar, editar, cambiar y eliminar elementos del DOM de manera dinámica.

Efectos visuales: Utilizando la manipulación de los elementos o técnicas CSS3[4] permiten crear efectos visuales y animaciones (aparecer, desvanecer, mover, redimensionar, entre otros).

Validaciones: Permiten de una manera relativamente fácil validar campos dentro de uno o varios formularios, lo cual simplifica y reduce el código para procesar estos últimos.

Manejo de Eventos: Estandariza la gestión de eventos en los distintos navegadores.

Drag'n Drop: Funcionalidad que brinda la posibilidad de arrastrar (*drag*) elementos dentro de una misma página y soltarlos (*drop*) con el objetivo de que interactúen con otros.

Widgets. Algunas bibliotecas incorporan componentes visuales que permiten crear interfaces gráficas de manera sencilla.

[4] Última especificación del lenguaje usado para definir la presentación de un sitio web, añade soporte para animaciones y efectos.

1.4.1 Clasificación

Atendiendo a las funcionalidades que brindan, las bibliotecas JavaScript pueden clasificarse en dos grupos: las especializadas o de propósito específico, que posibilitan la realización de tareas como la manipulación del DOM, animaciones y efectos, AJAX, criptografía, gestión de eventos, entre otras; y las bibliotecas de componentes visuales (también conocidas como *Widget Toolkits*), las cuales incorporan algunas de las habilidades antes mencionadas con el objetivo de conformar un núcleo como soporte para la creación de *widgets*, empleados en la construcción de GUIs.

El uso de bibliotecas JavaScript para el desarrollo de RIA permite la obtención de resultados en cortos períodos de tiempo y con un mínimo de líneas de código, lo cual facilita el posterior mantenimiento de la aplicación, dejando espacio para atender otros aspectos del proyecto. Entre las ventajas aparejadas al uso de estas herramientas pueden listarse las siguientes:

- Agilizan el proceso de desarrollo.
- Permiten la reutilización de código.
- Promueven buenas prácticas de implementación.
- Facilitan la interacción cliente - servidor.
- Resuelven los problemas de compatibilidad entre navegadores.
- Adaptabilidad de la aplicación a los diversos dispositivos.
- Incorporan *widgets* que mejoran la experiencia de usuario.

Aunque los beneficios de usar este tipo de bibliotecas superan a sus desventajas, conocer estas últimas supone tomar posibles acciones para mitigarlas. A continuación, algunas de ellas:

- Su uso añade diversas funcionalidades que pudieran no ser necesarias e impactarían de manera negativa en el rendimiento de la aplicación.
- Los programadores noveles suelen volverse dependientes de estas bibliotecas y no llegan a dominar el lenguaje JavaScript.
- Muchas bibliotecas son incompatibles entre sí, implicando que no puedan combinarse entre ellas para aprovechar todas sus ventajas.

El creciente número de estas herramientas y su evolución, confirman la necesidad y factibilidad de su aplicación. Como evidencia de ello, se encuentran las decenas de empresas que las utilizan, no solo en sus portales sino en sus aplicaciones (ver Anexo # 1). En el caso de la UCI, específicamente en DATEC, el panorama no es distinto.

1.5 Desarrollo de RIA en DATEC

Con la finalidad de dotar a sus productos de enriquecidas interfaces gráficas, el departamento de IS perteneciente a DATEC emplea la biblioteca Ext JS, la cual posibilita crear aplicaciones web de corte empresarial con características de aplicaciones de escritorio. Más allá de las ventajas que estas aplicaciones brindan a los usuarios finales, las cuales pueden apreciarse en la Tabla # 1, el uso de Ext JS para la creación de RIA ofrece a los desarrolladores una vía para el desarrollo ágil de las mismas.

Tabla # 1. Ventajas de las aplicaciones web.

Ventaja	Descripción
Instalación	No necesitan ser instaladas, solo accedidas mediante un navegador web.
Flexibilidad	Son capaces de ejecutarse en cualquier sistema operativo y sobre cualquier navegador.
Disponibilidad	Aunque esta característica pueda depender de terceros (como el propio proveedor de la conexión) por lo general las aplicaciones web se encuentran accesibles 24x7[5]
Bajo coste	El hecho de que la lógica del negocio (manejo de la concurrencia, acceso a datos, etc.) se ejecute en el servidor, implica que el usuario no deba invertir en potentes ordenadores para ejecutar las aplicaciones.

Esta biblioteca permite esencialmente construir complejas aplicaciones haciendo uso de componentes visuales previamente definidos, los cuales poseen como características su compatibilidad con los diferentes navegadores, la posibilidad de ser posicionados sobre distintos esquemas de distribución y una estética muy bien lograda. Ext JS cuenta además con una API (del inglés *Application Programming Interface*) de muy fácil uso y amplia documentación, que acelera el proceso de desarrollo, al tiempo que

[5]Término que hace referencia a las 24 horas de un día los 7 días de la semana.

brinda abstracción al lenguaje HTML y JavaScript facilitando el posterior mantenimiento de las aplicaciones, entre otras ventajas.

Entre los componentes visuales más usados en las soluciones que desarrolla el Centro, se encuentran los que conforman los formularios: *Input*, *Combobox*, *Radiobutton*, *Checkbox* y *Button*, y otros para la contención y presentación de datos, tales como *Panel*, *Window*, *Tab* y *Grid*. El uso de estos widgets garantiza la uniformidad y usabilidad del diseño, lo cual tributa a una positiva experiencia de usuario.

1.6 Soluciones existentes

En la actualidad existen varias bibliotecas JavaScript que permiten a los desarrolladores construir interfaces gráficas de usuario, algunas recientes y con no mucha aceptación, al tiempo que otras sobresalen por su robustez y facilidad de uso. Entre estas bibliotecas destacan jQuery UI, Dijit y YUI, cuya flexibilidad de licenciamiento garantiza además la soberanía tecnológica. A continuación se realiza una evaluación de las mismas, atendiendo a factores como la estructura, modularidad, apariencia, personalización, compatibilidad, tamaño y documentación, elementos que al decir de distintas bibliografías consultadas repercuten de manera significativa en la calidad del producto final o en su proceso de desarrollo.

jQuery UI

"*jQuery UI es una biblioteca de componentes visuales desarrollada sobre jQuery*" [4], por lo que su filosofía de uso es igual a la de esta última. Esta herramienta cuenta con varios *plugins* de interacción, así como una variada colección de *widgets* para el desarrollo de GUIs. Los *widgets* son completamente personalizables a través de diferentes temas disponibles para ello. Su primera versión fue lanzada en septiembre de 2007.

Estructura

La biblioteca consta de cuatro paquetes básicos, los cuales añaden distintas funcionalidades. El paquete UI Core es la dependencia fundamental y componente medular de la biblioteca y está constituido por *Core*, *Widget*, *Mouse* y *Position*. Este paquete contiene las funciones básicas e inicializadores necesarios para facilitar la creación de *widgets* basados en una API común, la cual incorpora una capa de abstracción

relativa a las interacciones mediante el ratón y mecanismos para obtener la posición de cualquier elemento.

El paquete *Interactions* añade habilidades a los *widgets*, como la capacidad para arrastrarlos y soltarlos, cambiar su tamaño, seleccionar grupos y reacomodar elementos dentro de una lista, todo esto gracias a los componentes *Draggable*, *Droppable*, *Resizable*, *Selectable* y *Sortable* respectivamente.

Los *widgets* disponibles están ubicados en el paquete homónimo. Entre ellos destacan *Accordion*, *Button*, *Datepicker*, *Dialog*, *Menu*, *Progressbar* y *Tabs*.

Modularidad

jQuery UI cuenta con *Builder*, herramienta que permite a los usuarios descargar solo aquellos componentes que se necesiten; el resultado final es una biblioteca personalizada a la medida de los requisitos de la aplicación a desarrollar. A pesar de esta característica, jQuery UI no cuenta con un administrador de dependencias capaz de cargar automáticamente los archivos *.js*[6] asociados a cada uno de sus *widgets*.

Apariencia

La apariencia de los componentes visuales es un factor clave, más aún si se tiene en cuenta que muchas veces los diseños desacertados o que provoquen frustración en el usuario pueden suponer el abandono de la aplicación. En el caso de esta biblioteca, el diseño de los *widgets* ha sido bien concebido, con líneas nítidas y colores agradables a la vista, sin embargo su tema por defecto no está ideado para ser usado en aplicaciones de corte empresarial como las desarrolladas por el departamento de IS.

Personalización

Si bien el diseño por defecto de los *widgets* es agradable, la personalización es un punto a considerar. La personalización contribuye a definir la identidad de un producto y con ello ayuda al cliente a percibir su origen y por tanto, a sentirse seguro de este y su uso; es la imagen de la empresa transformada en un producto. jQuery UI posee decenas de temas disponibles, además, cuenta con la herramienta *ThemeRoller*, la cual permite a los usuarios crear sus propios temas de manera sencilla.

[6] Extensión de los archivos JavaScript

Compatibilidad

La última versión estable de esta biblioteca es la 1.9.2, siendo compatible con versiones iguales o superiores a los siguientes navegadores:

- Internet Explorer 6.
- Mozilla Firefox 3.
- Safari 3.1.
- Opera 9.6.
- Google Chrome.

Tamaño

El tamaño de la biblioteca determina en gran medida su velocidad de descarga, por lo que tamaños mínimos deberán ser prioridades a la hora de desarrollar este tipo de herramientas. En el caso de jQuery UI y teniendo en cuenta el tamaño de jQuery este asciende a tan solo 323 KB.

Licencia

Al igual que jQuery, jQuery UI se distribuye bajo las licencias GPL y MIT (*Massachusetts Institute of Technology*). Ambas licencias permiten a los creadores de la biblioteca conservar los derechos intelectuales sobre la misma, sin embargo la del MIT, a diferencia de la GPL, no limita su uso por los desarrolladores de la forma más conveniente.

Documentación

Debido a su amplia aceptación y masivo uso, esta biblioteca cuenta con una de las comunidades de usuarios más grandes. Siendo así, existen además del sitio oficial, multitud de espacios donde encontrar ayuda y ejemplos de su uso.

Dijit

Dijit es la librería de *widgets* de Dojo. Es una colección de algunos de los más usados componentes para la creación de interfaces gráficas de usuario. Entre sus características más destacables se encuentran, la posibilidad de usarla de forma declaratoria (usando atributos especiales dentro de las etiquetas HTML) o programáticamente (mediante JavaScript); así como "*la internacionalización, la cual garantiza la compatibilidad de los mismos con las distintas culturas e idiomas*" [5] y el hecho de que los

desarrolladores puedan crear de una forma relativamente fácil sus propios *widgets*. Dijit es mantenida por la *Dojo Foundation* y su amplia comunidad.

Estructura

Dijit es además el nombre del paquete y directorio donde se encuentran definidos los *widgets* con los que cuenta. Algunas de sus principales clases son las siguientes (Ver Figura # 2):

- *dijit._Widget* es la clase principal de la jerarquía y clase padre de cada uno de los *widgets*.
- *dijit._Templated* provee mecanismos que permiten utilizar plantillas HTML en la construcción de los *widgets*.
- *dijit._Containeres* una clase capaz de brindar funcionalidades a aquellos *widgets* que pueden contener a otros como hijos.
- *dijit.form._FormWidgets* clase padre de los *widgets* utilizados en los formularios.
- *dijit.layout._LayoutWidgets* posibilita la existencia de aquellos *widgets* que necesitan de un *layout*[7] para contener otros dentro.

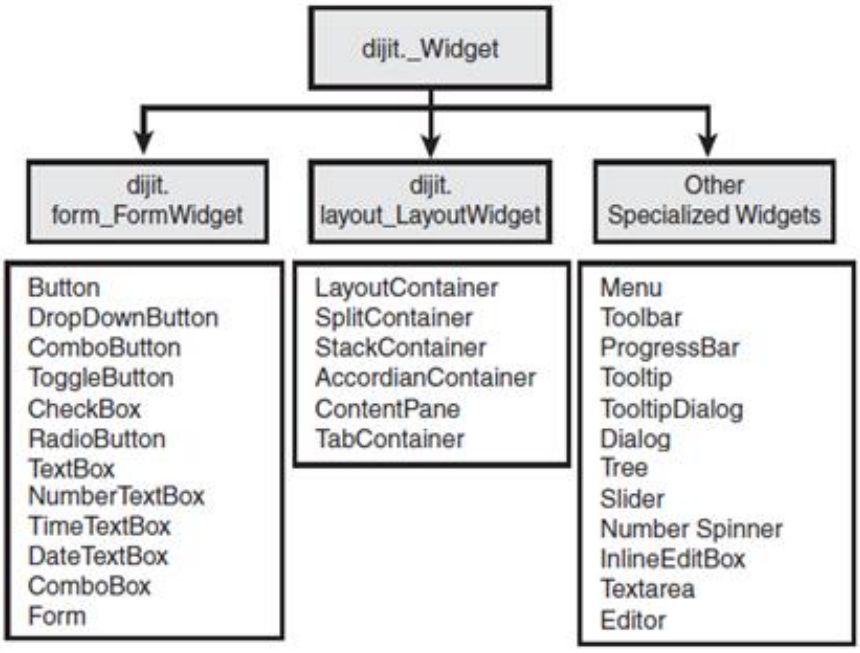


Figura # 2. Jerarquía de los componentes de Dijit [5].

[7] Esquema de distribución o disposición de los elementos dentro un diseño.

Modularidad

El uso de Dojo como núcleo supone la ventaja de aprovechar sus funcionalidades. Dojo cuenta con una utilidad llamada *require* la cual permite cargar solo aquellos módulos que sean necesarios; la aplicación de este mecanismo, conocido como carga bajo demanda, garantiza el empleo de solo aquellos módulos estrictamente necesarios. A pesar de esto, existe un alto acoplamiento entre sus componentes por lo que no es posible utilizar solo una parte de la biblioteca.

Apariencia

Una característica destacable de esta colección de componentes es la accesibilidad; los *widgets* pueden personalizarse con temas de alto contraste muy fáciles de visualizar y manejables mediante el teclado. Aunque la apariencia visual de Dijit en versiones anteriores a la 1.8 resultaba ser estéticamente pobre, la nueva versión supera con creces este aspecto.

Personalización

Dijit cuenta solamente con 4 temas oficiales. Sin embargo, la herramienta *Dijit ThemeTester* permite a los usuarios crear sus propios temas.

Compatibilidad

La versión 1.8, última liberación de esta biblioteca, es compatible con versiones superiores de los siguientes navegadores:

- Firefox 3.6
- Safari 5
- Chrome 13
- Internet Explorer 6

Tamaño

Debido a que Dijit utiliza a Dojo como base para la implementación de los componentes visuales, el tamaño total de esta biblioteca asciende a los 7 MB.

Licencia

Dijit forma parte de Dojo Toolkit, el cual se distribuye bajo las licencias AFL (*Academic Free License*) y BSD (*Berkeley Software Distribution*). Este licenciamiento dual, contrario al de alternativas como Ext JS, permite usar libremente el software como parte de productos de código abierto, así como privativos.

Documentación

Una de las principales debilidades de esta biblioteca es su escasa documentación. Si bien es cierto que la versión 1.8 ha incorporado una renovada documentación de su API, los tutoriales y ejemplos de uso disponibles aún son insuficientes, restando mucho para igualar en número y calidad a los recursos existentes para otras bibliotecas.

YUI

YUI, acrónimo en inglés de *Yahoo! User Interface*, es una biblioteca creada por *Yahoo! Inc*. haciendo uso de JavaScript. Está conformada por un conjunto de utilidades, *widgets* y recursos CSS que facilitan el diseño e implementación de interfaces gráficas de usuario, haciendo uso de técnicas DOM, DHTML y AJAX. A pesar de haberse liberado la versión 3, YUI 2, lanzada en 2006, sigue siendo la alternativa mayormente utilizada con una cuota de "*94.2% frente al 5.8% alcanzado por YUI 3*" [6].

Estructura

YUI está compuesto por tres capas: *Core*, *Utilities* y *Widgets*. La primera de ellas se encarga de gestionar las dependencias entre los componentes, sirviendo además como mecanismo de abstracción a funcionalidades para la gestión de eventos, manipulación del DOM, entre otras. *Utilities* posibilita la realización de animaciones, arrastrar y soltar elementos, así como realizar peticiones asíncronas al servidor. *Widgets* proporciona capacidades para la gestión del ciclo de vida de los componentes visuales, así como sus atributos. Algunos de los *widgets* disponibles se listan a continuación (Tabla # 2), nótese como algunos de ellos continúan en fase de prueba.

Tabla # 2. Componentes visuales de YUI.

Componentes estables	Componentes beta
AutoComplete Control	Charts Control
Button Control	DataTable Control
Calendar Control	Image Cropper
Container	Layout Manager
Color Picker Control	Rich Text Editor
Menu Control	Uploader

Modularidad

A pesar de su poca expansión, la última versión de YUI posee una arquitectura modular que permite hacer uso únicamente de los archivos que se necesitan. La mayor ventaja de esta versión, radica en su capacidad para cargar las dependencias asociadas a los módulos utilizados de forma totalmente autónoma, evitando la intervención del desarrollador y la carga innecesaria de archivos.

Apariencia

El diseño de los *widgets* en YUI 3 fue renovado, representando una importante mejora con respecto a YUI 2 (ver Anexo # 2). El nuevo esquema visual se adapta a las características de la web moderna, tales como el diseño adaptable y el uso de CSS3, al tiempo que simboliza además un salto cualitativo en la experiencia de usuario a la hora de interactuar con la aplicación.

Personalización

Uno de los puntos débiles de esta biblioteca es su reducida cantidad de temas y la dificultad de crearlos de una forma sencilla, puesto que no cuenta con una herramienta para ello.

Compatibilidad

Con el lanzamiento de la versión 3 el equipo de desarrollo de YUI suprimió la compatibilidad con navegadores antiguos como Internet Explorer 7 y Firefox 3. La última versión estable de esta biblioteca, la 3.8.0, solo es compatible con los navegadores web modernos o sus versiones superiores:

- Mozilla Firefox 4
- Internet Explorer 9
- Safari 5.2
- Opera 9.5
- Google Chrome 18

Tamaño

A pesar de contar con un cargador asíncrono y la personalización de la descarga a través de la herramienta YUI *Configurator*, la biblioteca, en su versión comprimida alcanza un tamaño de casi 6 MB, lo cual supone un alto tráfico entre el cliente y el servidor.

Licencia

Los componentes de YUI han sido desarrollados bajo licencia BSD y están disponibles para todo tipo de uso, desde proyectos personales hasta aplicaciones comerciales.

Documentación

YUI cuenta con una amplia comunidad y una vasta documentación. En su sitio oficial, mantenido por *Yahoo! Inc.* se encuentra el API de la biblioteca, así como guías de usuario y ejemplos que facilitan su aprendizaje.

1.6.1 Evaluación

Si bien es cierto que las bibliotecas anteriormente analizadas incluyen licencias flexibles que permitirían su uso en proyectos comerciales, estas presentan dificultades en áreas claves como la personalización, el rendimiento, la apariencia y la documentación. Así por ejemplo, jQuery UI, carece de abstracción para la Programación Orientada a Objetos (*Object Oriented Programming*, OOP), por lo que el desarrollador necesita crear toda la estructura de los *widgets* mediante HTML para luego transformarla utilizando las funcionalidades brindadas por la biblioteca, lo que implica el aprendizaje no solo de esta, sino también de HTML y dificulta la realización de posteriores labores de mantenimiento sobre el código. Aunque cabe destacar que la última versión de esta biblioteca añade soporte para la creación de instancias mediante constructores, no posee mecanismos que permitan añadir fácilmente componentes dentro de contenedores.

A diferencia de jQuery UI, Dijit favorece la creación de GUIs haciendo uso de OOP, sin embargo, su falta de documentación e incompatibilidad entre versiones (algunos usuarios reportaban código no funcional entre las versiones 1.6 y 1.7), así como el alto grado de acoplamiento entre sus componentes y elevado tamaño, suponen debilidades de considerable peso en el contexto de las redes cubanas.

En el caso de YUI, es precisamente el tamaño su principal inconveniente, más si se tiene en cuenta que la carga de archivos JavaScript bloquea la descarga paralela de otros. Como se ha visto, esta biblioteca cuenta con dos versiones en uso, la última de ellas, YUI 3, aún no cuenta con una gran cantidad de *widgets* y solo es compatible con navegadores modernos, mientras que la versión 2 posee una elevada curva de aprendizaje y necesita decenas de líneas de código para implementar sencillas funcionalidades.

Los elementos antes planteados demuestran la imposibilidad de tomar alguna de estas herramientas como solución absoluta al problema de la investigación, pues de una u otra forma atentarían contra la calidad de las aplicaciones o su proceso de desarrollo. No obstante, debido a su reducido tamaño, extensibilidad y arquitectura modular, resulta factible la utilización de jQuery UI como base tecnológica

para el desarrollo de una nueva biblioteca, la cual solucionaría las limitantes antes expuestas mediante el empleo de una capa de abstracción sobre sus *widgets* que facilite la OOP.

Otra de las características destacables de jQuery UI, son las decenas de *plugins* existentes y que pudieran utilizarse ante cualquier eventualidad, en este sentido y ante la falta del *widget Grid* en la plataforma seleccionada, se decide hacer uso de *jQGrid*, debido a su abanico de funcionalidades así como la compatibilidad con los disímiles temas existentes para la personalización.

La solución propuesta incluiría además un administrador de dependencias capaz de cargar de forma automática los archivos asociados a cada componente, lo cual facilitaría el trabajo de los desarrolladores y garantizaría la inclusión de solo aquellos archivos estrictamente necesarios. En cuanto a la personalización de los *widgets*, la biblioteca será compatible con los temas actuales y podrá ser personalizada utilizando las herramientas existentes para ello.

1.7 Ambiente de desarrollo

La adecuada selección de la metodología, herramientas y tecnologías es clave para obtener un producto de software con calidad. A continuación se define el entorno o ambiente de desarrollo a usar para la construcción de la solución propuesta.

1.7.1 Metodología de desarrollo del software

Extreme Programming

La Programación Extrema, del inglés eXtreme Programming (XP) ofrece una alternativa ágil a los procesos de desarrollo tradicionales caracterizados por ser rígidos y dirigidos por la documentación. Su funcionamiento se basa en retroalimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y *"coraje"* para enfrentar los cambios. "*Esta metodología resulta especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico*." [7] De acuerdo con algunos autores las fases definidas en XP son las siguientes: "***Planeación***, ***Diseño***, ***Codificación*** *y* ***Pruebas***". [8] [9]

Entre las características de esta metodología destacan:

- Desarrollo iterativo e incremental.
- Programación en parejas.
- Pruebas unitarias continuas.
- Integración del equipo de desarrollo con el cliente.
- Refactorización del código.

La solución propuesta presenta las siguientes particularidades:

- El hecho de que el cliente (los desarrolladores del departamento), se encuentre físicamente disponible, permite involucrarlo en el desarrollo de la solución y contribuye a la retroalimentación directa; un "*valor defendido por XP que promueve el éxito de un proyecto de desarrollo de software"* [10].
- A pesar de su complejidad, se trata de un proyecto de pequeñas dimensiones, que requiere de ejemplos de uso y código funcional, pero no de documentación exhaustiva.
- El uso de jQuery UI como base para el desarrollo supone la reutilización y refactorización de código, no solo del agregado, sino también del ya existente. XP avala esta práctica siempre y cuando no se introduzcan errores a la solución.
- La captura de requisitos mediante Historias de Usuario (HU) resulta efectiva en este caso, pues las descripciones de las mismas serán realizadas por programadores del departamento IS, lo cual, teniendo en cuenta que se habla el mismo idioma, propicia un mejor entendimiento entre las partes.
- *"El diálogo frontal, cara a cara, entre desarrolladores, gerentes y el cliente es el medio básico de comunicación."* [8]. La estrecha relación entre el equipo de desarrollo y el cliente, permite que se aplique este tipo de comunicación, posibilitando que no se deban generar grandes volúmenes de documentación.

- El equipo de desarrollo está compuesto por dos personas, lo cual propicia la revisión y discusión del código mientas se escribe. Esta práctica tributa a la calidad final del código y es defendida por XP.

En este escenario, el uso de XP supone:

- Rapidez en la obtención de versiones a causa de la rápida codificación.
- Obtención de productos fiables y robustos debido al diseño de las pruebas previo a la codificación.
- Reducción del número de posibles errores debido a que los requisitos del software son fáciles de modificar.
- Énfasis en la adaptabilidad y no en la previsibilidad, por lo que cambios en los requisitos no afectan los planes de entrega del producto.
- Mínimo de documentación.

Sin embargo, con el objetivo de describir los elementos ya presentes en jQuery UI, base de la nueva biblioteca, así como las relaciones existentes entre sus componentes y aquellos que serán implementados, se decide hacer uso de artefactos presentes en Open Up como el Modelo de Domino y Diagrama de Clases, los cuales permitirán una mejor comprensión de la solución. Es válido resaltar que esta combinación es permisible en el contexto de XP, siempre y cuando no altere las iteraciones y entregas del producto.

De los roles que propone la metodología XP, intervendrían en la construcción de la solución los siguientes:

Programador: Confecciona las Historias de Usuario y Tareas de Ingeniería, además escribe las pruebas unitarias y produce el código del sistema.

Encargado de pruebas: Ejecuta pruebas regularmente, difunde los resultados en el equipo y es responsable de las herramientas de soporte para pruebas.

Analista: A pesar de no estar definido dentro de la metodología seleccionada, se propone la participación de este rol debido al importante papel que puede jugar en la planificación y diseño de la solución, siendo

el encargado de realizar las estimaciones, el Modelado del Negocio así como el Diagrama de Clases y las Tarjetas CRC.

1.7.2 Lenguaje de modelado

Lenguaje Unificado de Modelado

"*UML es un lenguaje estándar que permite especificar, visualizar, construir y documentar todos los elementos que forman un sistema de software orientado a objetos*" [11]. Su propósito es facilitar el modelado de los artefactos del sistema a través de las etapas de su ciclo de vida y no el de describir sus métodos o procesos. Está compuesto por elementos gráficos y reglas que se combinan para crear diagramas.

Varios son los beneficios de usar este lenguaje de modelado, entre ellos:

- Es un estándar defacto en la industria.
- Posibilita establecer un conjunto de requisitos y estructuras necesarias para plasmar un sistema previo a la escritura del código.
- Aporta productividad en las labores de diseño arquitectónico.
- Facilita el entendimiento entre los miembros del equipo de trabajo.
- Contribuye a la organización de la documentación.

1.7.3 Herramienta CASE

Visual Paradigm for UML

Visual Paradigm es una herramienta profesional que soporta el ciclo de vida del desarrollo de software. Por sus características, este software de modelado UML es considerado como una herramienta muy completa y fácil de usar, lo cual contribuye a la rápida construcción de aplicaciones. Entre ellas destacan:

- Multiplataforma: Capaz de ejecutarse en sistemas operativos GNU/Linux, Windows y OS X.
- Facilita la colaboración: Idóneo para el modelado simultáneo a través del *Team Work Server* y *Subversion*.
- Genera documentación en formatos PDF y HTML.

- Permite la generación de código y la realización de ingeniería inversa para los lenguajes que soporta.

1.7.4 Lenguaje de programación

JavaScript

JavaScript es un lenguaje de programación interpretado desarrollado inicialmente por Netscape[8] y actualmente estandarizado en los navegadores web modernos gracias a la ECMA (*European Computer Manufacturers Association*) y su estándar *ECMAScript*. Inicialmente utilizado para añadir efectos y validar formularios, el lenguaje ha tomado mayor fuerza en los últimos tiempos debido al empleo de AJAX, así como a las bibliotecas que lo usan como soporte. El mismo funciona del lado del cliente y a pesar de ser considerado por muchos como no orientado a objetos, "*permite implementar varias de las características de este paradigma de programación*" [12]. Lo anterior, unido a su facilidad de uso y amplia aceptación, supone una ventaja para los desarrolladores.

1.7.5 Entorno Integrado de Desarrollo

NetBeans IDE 6.9

NetBeans es un Entorno Integrado de Desarrollo, del inglés *Integrated Development Enviroment* (IDE) disponible para los sistemas operativos GNU/Linux y Windows. Es un producto libre, gratuito y sin restricciones de uso, que posibilita el desarrollo de aplicaciones haciendo uso de varios lenguajes de programación incluido JavaScript. Entre sus características se encuentran la indentación y completamiento de código, resaltado de sintaxis y corrección de errores, todo lo cual resulta de marcada utilidad para la implementación utilizando JavaScript.

1.7.6 Suite de pruebas

QUnit

En programación, una prueba unitaria es una "*forma de probar el correcto funcionamiento de un módulo*" [13]. El objetivo de estas pruebas es asegurar que cada una de las partes que integren una solución o producto, funcionen correctamente. QUnit es una completa suite desarrollada por el *jQuery Team* para

[8] Empresa de software precursora de la entonces naciente *World Wide Web.*

testear módulos JavaScript. Inicialmente fue concebida para ser usada sólo sobre el código de jQuery, sin embargo su flexibilidad y eficacia para detectar errores, unido a su facilidad de uso, la convierten en una herramienta ideal para probar código JavaScript.

1.7.7 Tecnologías

CSS

Las Hojas de Estilo en Cascada o CSS (*Cascading Style Sheets*) es la "*tecnología creada por el W3C*[9] *con la finalidad de separar la estructura de la presentación*" [14]. Su uso posibilita aplicar formato a los documentos escritos en HTML haciendo uso de reglas y selectores. Varias son las especificaciones de CSS, la última de ellas, CSS3, incluye nuevas características como la posibilidad de aplicar bordes redondeados, gradientes y animaciones a los elementos HTML, sacando provecho de la aceleración por hardware en caso de estar disponible.

1.8 Conclusiones

El estudio realizado sobre las bibliotecas JavaScript permitió destacar las ventajas de jQuery UI sobre el resto, siendo seleccionada como plataforma para la creación de una nueva biblioteca para el desarrollo de GUIs. Debido a las características de la solución propuesta, se utilizará la metodología de desarrollo XP y QUnit como herramienta para automatizar las pruebas unitarias propuestas por dicha metodología, además del Modelo de Dominio (MD) y Diagrama de Clases presentes en OpenUp.

Finalmente, se adopta UML como lenguaje para el modelado y Visual Paradigm para llevar a vías de hecho esta tarea, así como el IDE NetBeans, JavaScript como lenguaje para la implementación y CSS para especificar el estilo de los *widgets*.

[9] *World Wide Web Consortium*. Fundado por Tim Berners Lee el 1ro de Octubre de 1994.

CAPÍTULO 2. PLANEACIÓN Y DISEÑO DE LA SOLUCIÓN

En este capítulo se realiza una descripción de la solución a desarrollar y su integración con la plataforma seleccionada. Se presenta el Modelo de Dominio e Historias de Usuario, en las cuales se exponen las características de cada una de las funcionalidades y componentes que constituyen la biblioteca. Se obtiene la Lista de Reserva del Producto (LRP), artefacto que recoge los requisitos funcionales y no funcionales del software. Es definida además la arquitectura de la biblioteca, los patrones de diseño empleados, las Tarjetas CRC identificadas y un diagrama de clases global donde se muestran las relaciones existentes entre todas las clases que conforman la biblioteca.

2.1 Situación actual

Los *widgets* de jQuery UI fueron desarrollados tomando a jQuery como núcleo y utilizando las diferentes funcionalidades brindadas por este. Por tal razón, su filosofía de uso es básicamente la misma: "*find something, manipulate it"* (encuentra algo, manipúlalo). Esta técnica implica la creación de una estructura HTML que luego debe ser seleccionada y transformada al *widget* en cuestión. (Ver Figura # 3)

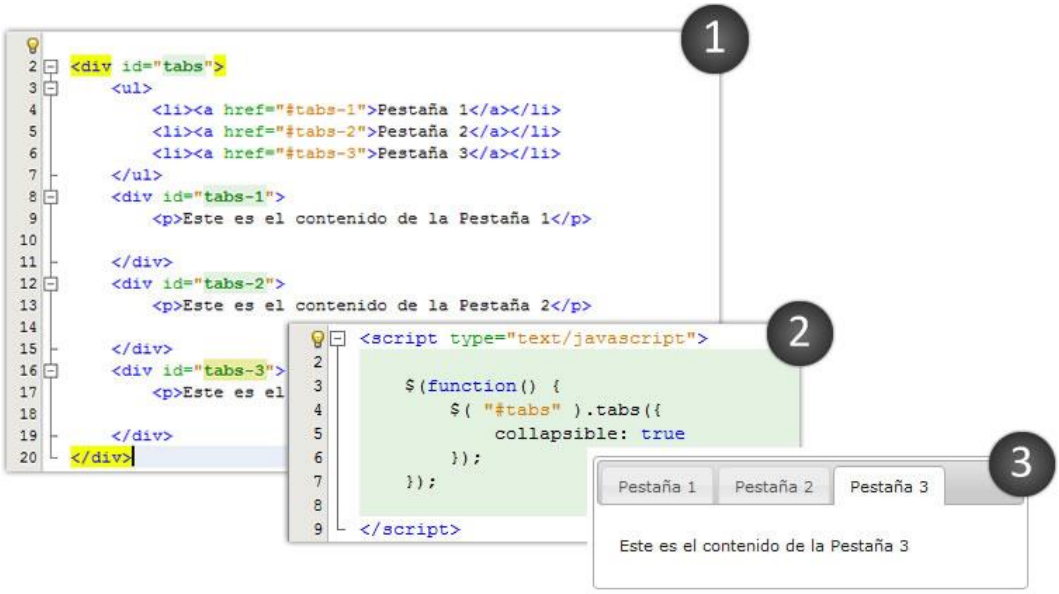


Figura # 3. Proceso para la creación del *widget* Tabs.

2.2 Modelo de Dominio

A pesar de no contar con un flujo de trabajo para definir el negocio, XP posibilita la especificación de este mediante la forma más cómoda y entendible para aquellas personas que trabajan en el desarrollo de la solución; siempre que no propicie demoras en la entrega del producto final. En este caso se decide utilizar un MD debido a que se cuenta con una base ya implementada, de la cual se necesita conocer su funcionamiento y responsabilidades.

"*Un Modelo de Dominio captura los tipos más importantes de objetos en el contexto del sistema. Los objetos del dominio representan las cosas que existen o los eventos que suceden en el entorno en el que trabaja el sistema*" [11]. Esta práctica es muy utilizada, pues favorece un mejor entendimiento del negocio y de aquello que debe realizarse (Ver Figura # 4).

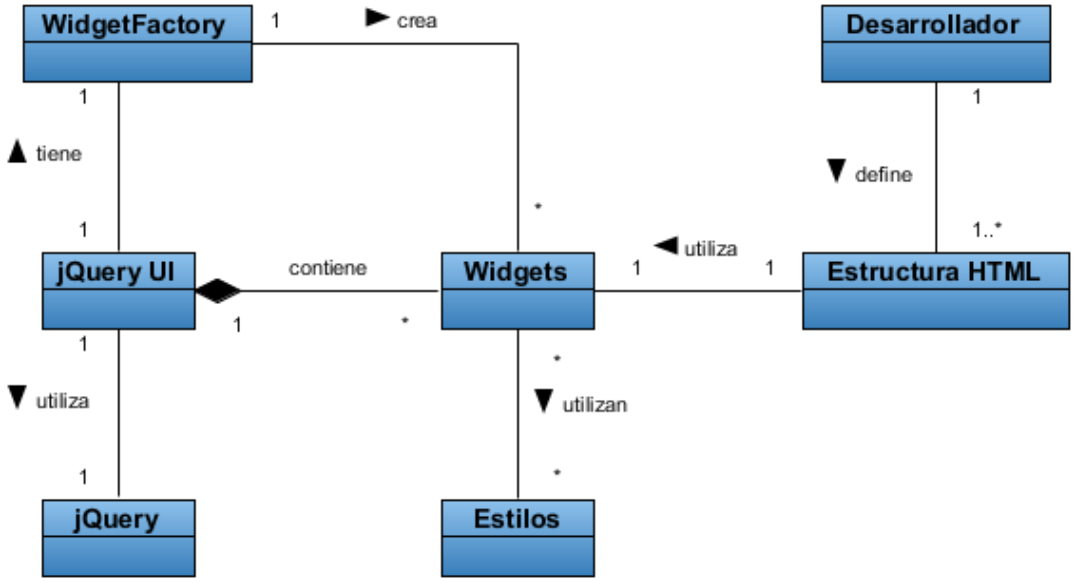


Figura # 4. Modelo de Dominio

El diagrama evidencia la estructura y funcionamiento de la biblioteca jQuery UI, la cual utilizando a jQuery como núcleo y a través del *Widget Factory* crea los *widgets* que la componen. Posteriormente a su definición, estos componentes visuales, podrán ser utilizados por el desarrollador siempre que se haya implementado la estructura de nodos requerida. La apariencia de los *widgets* es otorgada por las reglas CSS especificadas en las Hojas de Estilos.

Descripción de los objetos del dominio

Desarrollador: Encargado de la definición de las estructuras HTML que serán transformadas en *widgets*.

Estructura HTML: Colección o jerarquía de nodos necesitados por los *widgets* para su correcta inicialización y funcionamiento.

jQuery UI: Biblioteca de componentes para jQuery. Incorpora un conjunto de *widgets* y efectos visuales para la creación de aplicaciones web.

jQuery: Núcleo de jQuery UI. Proporciona funcionalidades para la selección y manipulación de los elementos del DOM, gestión de eventos, entre otras que pueden ser reutilizadas para facilitar la creación de *widgets*.

Widgets: Colección de componentes visuales presentes en jQuery UI.

Widget Factory: Mecanismo mediante el cual son creados los *widgets*.

Hojas de Estilo: Contiene la especificación de los distintos estilos asociados a los *widgets*.

2.3 Solución propuesta

A pesar de contar con características como la modularidad, poco peso y probada práctica en el mercado, la filosofía de uso de jQuery UI se encuentra marcada por una fuerte dependencia de HTML, dificultando la creación y posterior mantenimiento de aplicaciones empresariales.

Las modificaciones a realizar sobre jQuery UI con el objetivo de dar respuesta al problema planteado al inicio de esta investigación, incluyen la implementación de una capa de abstracción que soporte la OOP (Ver Figura # 5). Esta variante, en contraposición a la de modificar directamente los componentes, permitiría reutilizar la capa implementada en futuras versiones de jQuery UI, además de garantizar que el método clásico de funcionamiento de la biblioteca no sea afectado.

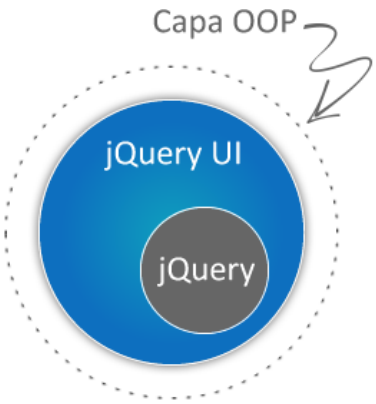


Figura # 5. Ubicación de la capa de soporte a la OOP

La capa de abstracción antes mencionada definiría nuevos *widgets* basados en aquellos ya existentes en jQuery UI y agrupándolos en una nueva jerarquía (Ver Figura # 6). Permitiendo o mejorando (la versión 1.9 ya permite instanciar mediante constructores) la creación de instancias de estas clases y su posterior manipulación, heredar componentes para crear otros, así como mecanismos que permitan al desarrollador definir sus propios *widgets*.

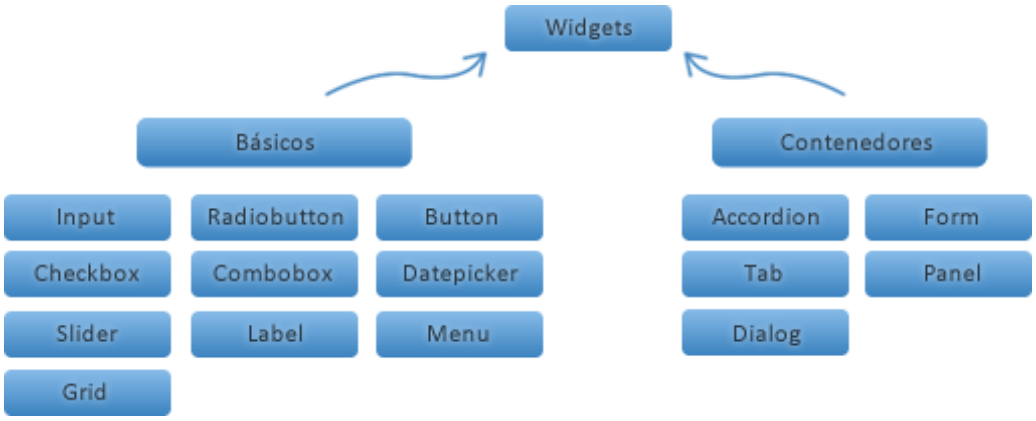


Figura # 6. Jerarquía de los *widgets*

La implementación de estos nuevos *widgets* requiere funcionalidades que faciliten la creación de las clases que los definirían, así como un cargador de dependencias, módulo muy común en las bibliotecas de este tipo, pero ausente en jQuery UI. Este mecanismo posibilita la carga automática de las dependencias asociadas a cada *widget*, evitando la inclusión innecesaria de código y garantizando su carga solo cuando sea necesaria.

2.4 Historias de Usuario

"Las HU son la técnica utilizada en XP para especificar los requisitos del software" [7]. Estas detallan cada uno de los requisitos del sistema, sean funcionales o no funcionales, sin hacer uso de extensos documentos, además, su correcta definición será clave para guiar el proceso de implementación del software. A pesar de ser elaboradas por el cliente, los desarrolladores pueden brindar ayuda en la identificación y elaboración de las mismas.

En el presente trabajo de diploma se obtuvieron un total de 18 HU que serán realizadas en tres iteraciones. A continuación se muestra la HU "Gestionar *widget* Combobox" (Ver Tabla # 3), las restantes podrán consultarse en el artefacto "Plantilla de Historias de Usuario".

Tabla # 3. HU "Gestionar *widget* Combobox".

Historia de Usuario	
Número: 10	**Nombre de la Historia de Usuario:** Gestionar *widget* Combobox.
Cantidad de modificaciones a la Historia de Usuario: 1	
Usuario: José R. Guzmán Ojeda	**Iteración asignada:** 2
Prioridad en negocio: Alta	**Puntos estimados:** 0.5 semana
Riesgo en desarrollo: Alto	**Puntos reales:** 0.5 semana
Descripción: Permitir la creación de instancias de cajas desplegables con múltiples elementos donde el usuario pueda seleccionar uno de ellos. Además permitirá la modificación de sus propiedades, la programación de eventos asociados al *widget* y la destrucción o eliminación del mismo.	
Observaciones:	
Prototipo de interfaces: Seleccionar	

2.5 Lista de Reserva del Producto

La LRP es una tabla que contiene los requisitos funcionales que debe cumplir la biblioteca a desarrollar (Ver Tabla # 4), ordenados según su prioridad de implementación en Muy Alta, Alta, Media y Baja; en esta

última categoría aparecen los requisitos de menor complejidad, así como los no funcionales. Se indica además el tiempo estimado por semanas para la implementación de los requisitos funcionales y el rol que realizó la estimación.

Tabla # 4. Lista de Reserva del Producto.

Ítem	Descripción	Estimación	Estimado por
Prioridad: Muy Alta			
1	Clase Util	1 semana	Analista
2	Clase Manager	0.5 semana	Analista
3	Clase Button	0.5 semana	Analista
4	Clase Label	0.5 semana	Analista
5	Clase Input	0.5 semana	Analista
6	Clase Radiobutton	0.5 semana	Analista
7	Clase Checkbox	0.5 semana	Analista
8	Clase Combobox	0.5 semana	Analista
9	Clase Panel	1 semana	Analista
10	Clase Dialog	1 semana	Analista
11	Clase Form	0.5 semana	Analista
12	Cargador de dependencias	0.5 semana	Analista
Prioridad: Alta			
13	Clase Menu	1 semana	Analista
14	Clase Tabs	1 semana	Analista
15	Clase Accordion	1 semana	Analista
16	Clase Grid	1 semana	Analista
17	Clase Datepicker	0.5 semana	Analista
18	Clase Slider	0.5 semana	Analista
Prioridad: Media			

Prioridad: Baja			
1	Requisitos de Software: Navegadores Web: • Internet Explorer 9 o superior. • Mozilla Firefox 4 o superior. • Opera 10 o superior. • Google Chrome 16 o superior.		
2	Restricciones en el diseño y la implementación: Las herramientas y plataforma empleadas para el desarrollo de la biblioteca deberán poseer licencias no restrictivas que contribuyan a la flexibilidad en la comercialización del producto final.		
3	Requisitos de Hardware: Mínimos: • Procesador: Pentium 4. • Memoria RAM: 512 MB.		
4	Requisitos de Soporte: Deberá generarse un API, utilizando la propia biblioteca desarrollada, donde se explique el funcionamiento de cada uno de los componentes y funcionalidades de la misma.		

2.6 Arquitectura de software

La arquitectura es fundamental en cualquier sistema, pues como un modelo de referencia, permite centrarse en la estructura y funciones del mismo, haciendo posible especificar características inherentes a su implementación.

Tal vez la definición mayoritariamente aceptada de arquitectura de software es la ofrecida por el estándar 1471 de IEEE (*Institute of Electrical and Electronics Engineers*): *"Parte fundamental de un sistema expresado por sus componentes, sus relaciones con otros componentes y otros entornos, y los principios que guían en su diseño y evolución*" [15]. Otro concepto ampliamente adoptado por diferentes autores, enuncia como arquitectura de software a "*la estructura o estructuras del sistema, lo que comprende a los componentes del software, sus propiedades externas visibles y las relaciones entre ellos*." [16]

La arquitectura de software describe detalles de diseño de una colección de componentes y sus interconexiones, conformando una vista abstracta del sistema en diseño, lo cual incide en aspectos del desarrollo de software como la comprensión, reutilización y construcción del mismo.

2.6.1 Arquitectura basada en componentes

En una arquitectura basada en componentes el sistema se divide en varios componentes individuales, los cuales a través de interfaces bien definidas colaboran entre sí con el objetivo de satisfacer los requisitos funcionales del software. El SEI (*Software Engineering Institute*) define a un componente como: "*el fragmento de un sistema que puede ser ensamblado con otros fragmentos para formar piezas más grandes o aplicaciones completas*" [17].

A grandes rasgos, un componente es una pieza de código pre-elaborado que encapsula funcionalidades expuestas a través de interfaces. Estas piezas, en dependencia del nivel de detalle desde el cual se mire, pueden ser desde una subrutina, biblioteca matemática o clase, hasta un paquete e incluso una aplicación que pueda ser usada por otra a través de una interfaz. Tal variedad es conocida como *"granularidad de un componente"* [17].

El uso de este tipo de arquitectura (Ver Figura # 7) posibilitará un mayor nivel de reutilización, lo cual es crucial para el desarrollo de la solución propuesta, aprovechándose componentes y módulos previamente

definidos. Permitirá además, probar los *widgets* implementados de forma individual y contribuirá al posterior mantenimiento del código.

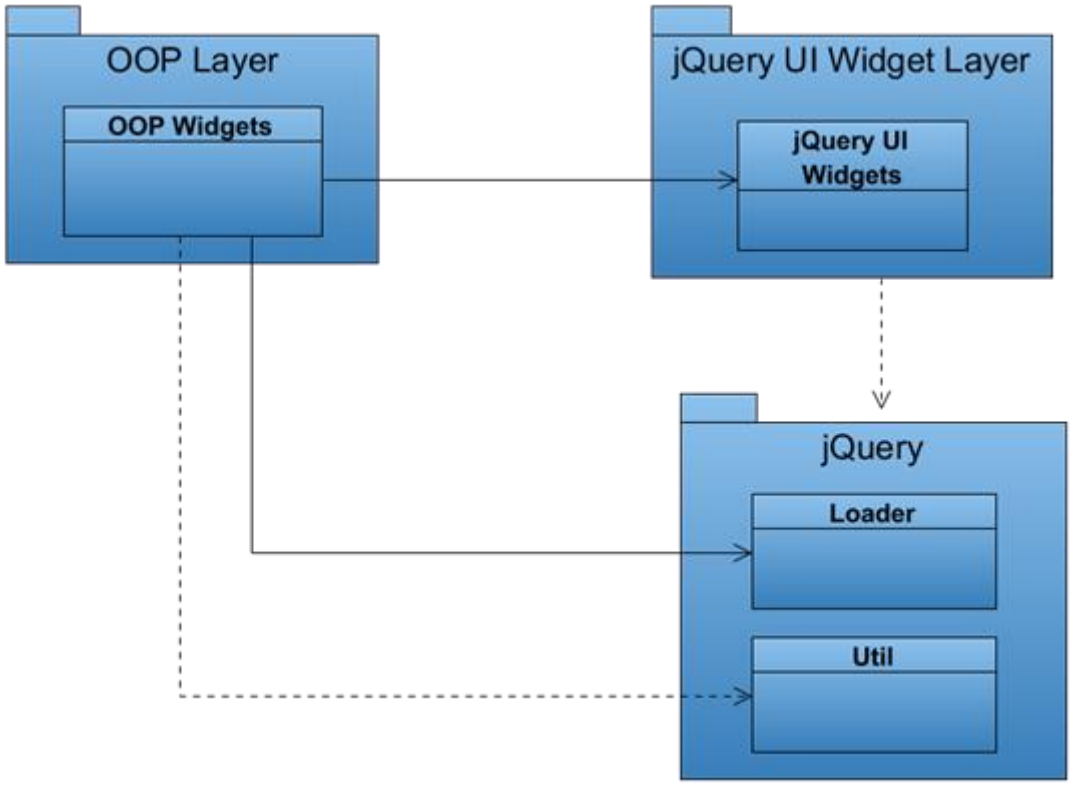


Figura # 7. Vista de la arquitectura de la biblioteca

2.7 Patrones de diseño

En informática, "*un patrón es una solución de diseño de software a un problema, aceptada como correcta, a la que se ha dado un nombre y que puede ser aplicada en otros contextos"* [18]. En el caso específico del diseño, los patrones facilitan la reutilización, extensibilidad y mantenimiento del software.

A continuación se muestran los patrones utilizados en la construcción de la solución divididos en dos grupos: Patrones Generales de Software para Asignar Responsabilidades (*General Responsibility Assignment Software Patterns*, GRASP) y GOF (*Gang of Four*). La ejemplificación del uso de estos últimos podrá encontrarse en los Anexos 3, 4 y 5 respectivamente.

Dentro de los GRASP utilizados se encuentran:

Experto

Enfatiza en la necesidad de asignar responsabilidades a las clases que poseen la información necesaria para cumplir con ella. Este patrón contribuye a un adecuado encapsulamiento, favoreciendo la robustez y el fácil mantenimiento de la biblioteca. La figura # 8 muestra la clase *Loader,* encargada de la carga automática de dependencias haciendo uso de las funcionalidades de jQuery.

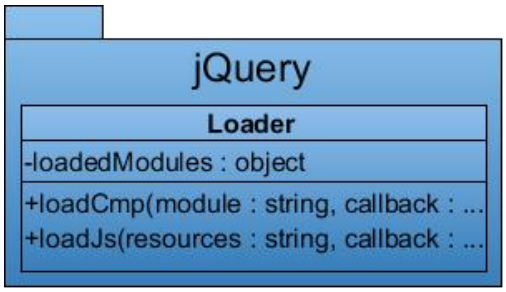


Figura # 8. Patrón "Experto"

Creador

Guía la asignación de responsabilidades relacionadas con la creación de objetos; "*se asigna la responsabilidad de que una clase A cree un objeto de una clase B sólo cuando:*

- *A contiene a B*
- *A es una agregación o composición de B*
- *A almacena a B*
- *A tiene los datos de inicialización de B*
- *A usa B*." [19]

Este patrón es ejemplificado en la Figura # 9, donde la clase *Datepicker* perteneciente al paquete *OOP Widgets* utiliza al componente homónimo existente en *jQuery UI Widgets.*

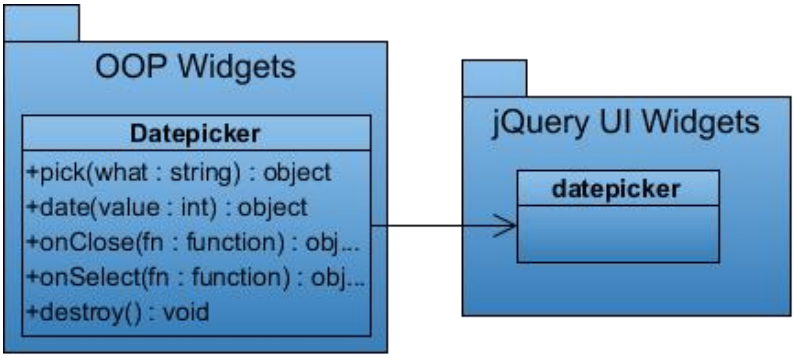


Figura # 9. Patrón "Creador"

Bajo acoplamiento

Se utiliza para contribuir a la modularidad de la biblioteca, garantizando un bajo acoplamiento entre clases y tratando de que cada una de ellas tenga la menor cantidad de relaciones posibles con otras. Lo antes dicho se evidencia en la Figura # 10.

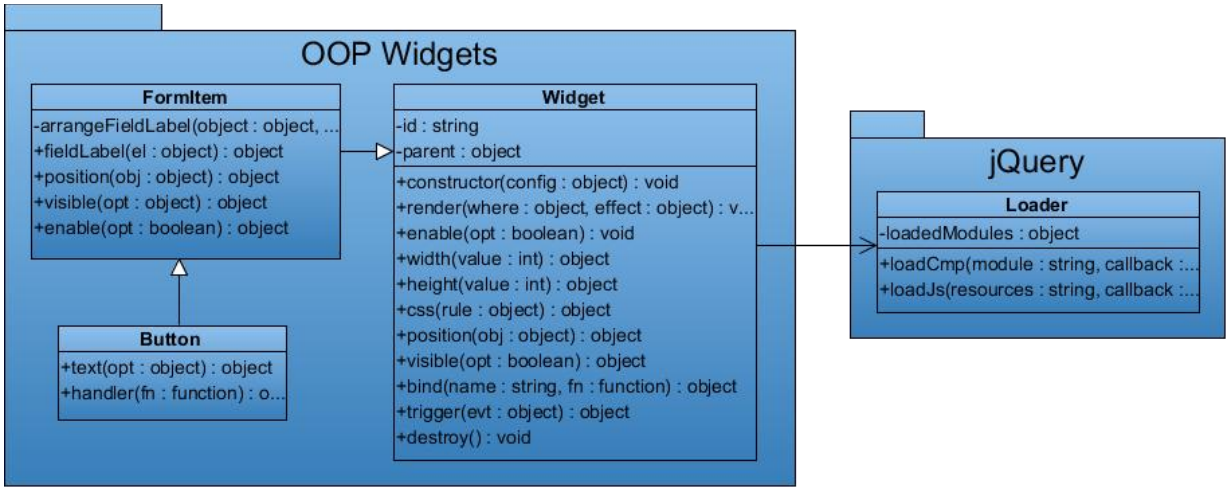


Figura # 10. Patrón "Bajo acoplamiento"

Alta cohesión

Establece que cada elemento del diseño debe realizar una labor única dentro del sistema y colaborar con otros objetos para compartir el esfuerzo, es decir, previene la mala práctica de asignar multitud de funcionalidades a una clase. En la Figura # 11 se muestra cómo las responsabilidades de carga automática de dependencias y los mecanismos para la definición de las clases han sido distribuidos en dos clases independientes: *Loader* y *Util* respectivamente.

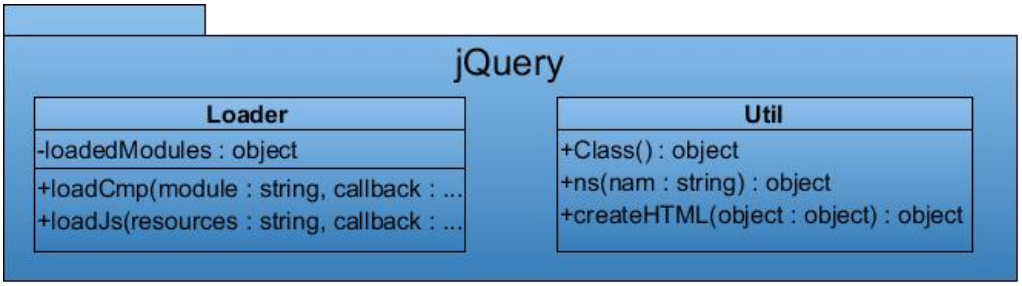


Figura # 11. Patrón "Alta cohesión"

La biblioteca propuesta hace uso de los siguientes patrones GOF:

Prototype

Es un patrón de muy fácil aplicación en JavaScript, su empleo posibilita clonar el prototipo de un objeto hacia otro con el fin de simular la herencia clásica. El uso de este patrón se evidencia en la construcción de la funcionalidad *Class* correspondiente a la clase *Util*, donde se definen los mecanismos necesarios para extender los atributos y funcionalidades de un widget hacia otro.

Adapter

Permite adaptar una interfaz de comunicación de un objeto, definiendo una nueva para ser utilizada, esta última facilita la reutilización de aquellos componentes existentes que de alguna forma no fuesen compatibles con lo requerido. El uso de este patrón se pone de manifiesto en la implementación de cada uno de los *widgets* presentes en la capa de abstracción, los cuales expondrían nuevas interfaces, comunicándose con aquellas definidas en los componentes de jQuery UI.

Facade

Proporciona un punto de acceso unificado para un conjunto de interfaces de un sistema, definiendo una interfaz de alto nivel encarga de facilitar la utilización de las mismas. La aplicación de este patrón permite agrupar las funcionalidades de creación y destrucción de los componentes.

2.8 Tarjetas CRC

En XP, el uso de Tarjetas CRC permite modelar el sistema a través de clases (unidades que encapsulan información referente a un objeto) y sus responsabilidades. Estas tarjetas son de muy fácil elaboración y en su construcción participa todo el equipo de desarrollo. Como se verá, están compuestas por el nombre

de la clase y dos columnas: la izquierda es usada para definir su responsabilidad, mientras que en la derecha se definen los nombres de las clases con las que se relaciona para llevar a vías de hechos su labor.

La Tabla # 5 muestra la tarjeta CRC asociada a la clase "*Combobox"*, al tiempo que en el expediente de proyecto podrán encontrase las restantes.

Tabla # 5. Tarjeta CRC correspondiente a la clase "Combobox".

Tarjeta CRC	
Clase: Combobox.	
Responsabilidades	**Colaboraciones**
Contiene funcionalidades que permiten la inicialización del widget, así como la modificación de sus propiedades y la programación de eventos. Las funcionalidades son: • ***readOnly****:* Cambia el valor de la propiedad *readOnly*, si no recibe parámetros retorna el valor de la propiedad *readOnly.* • ***selected:*** Retorna el valor del elemento seleccionado. Eventos del Combobox: • ***onChange***: Ejecuta una función especificada por parámetros al dispararse el evento onChange. • ***onClose***: Ejecuta una función especificada por parámetros al dispararse el evento onClose. • ***onFocus***: Ejecuta una función especificada por parámetros al dispararse el evento onFocus. • ***onOpen***: Ejecuta una función especificada por parámetros al dispararse el evento onOpen. • ***onSelect***: Ejecuta una función especificada por parámetros al dispararse el evento onSelect.	Súper Clase: FormItem. $.ui.combobox.

2.9 Diagrama de clases

A pesar de no estar definidos dentro de la metodología seleccionada, el uso de diagramas de clases posibilita describir mediante representaciones visuales las tarjetas CRC identificadas, así como sus relaciones. Inicialmente, esto permitió un mejor entendimiento en la concepción de la solución, contribuyendo posteriormente a la creación de un diagrama de clases global (Ver Anexo # 6), donde se muestran cada una de las clases presentes, sus atributos, métodos y relaciones. Este diagrama puede resultar de extrema valía para, en un futuro añadir nuevos *widgets* o funcionalidades a la biblioteca.

2.10 Conclusiones

La elaboración del Modelo de Dominio permitió un mejor entendimiento de la biblioteca jQuery UI, para así delimitar los elementos presentes y los que deberán ser implementados, con el objetivo de solucionar el problema de la investigación planteado. En este sentido, la elaboración de las 18 HU posibilitó la captura de 18 requisitos funcionales, los cuales podrán consultarse en la LRP.

El uso del patrón arquitectónico “basado en componentes” favorecerá la reutilización de los componentes presentes en jQuery UI, al tiempo que el empleo de 7 patrones de diseño posibilitó una mejor asignación de responsabilidades entre las clases y estructurar correctamente los componentes a implementar. Además, la confección de 23 Tarjetas CRC posibilitó la descripción de las clases de la biblioteca y la obtención de un diagrama de clases, utilizado para exponer de manera global las relaciones existentes entre estas, todo lo cual tributa a una mejor comprensión de la solución y permite pasar a la siguiente fase del desarrollo.

CAPÍTULO 3. IMPLEMENTACIÓN Y PRUEBAS

En este capítulo se definen los estándares de codificación que regirán el código fuente de la biblioteca. Se presentan las tareas ingenieriles asociadas a las HU, así como las iteraciones necesitadas para construir el producto final. Finalmente se realiza el diseño y aplicación de las pruebas necesitadas para validar la solución.

3.1 Estándares de codificación

Los estándares de codificación o convenciones de código son reglas que rigen la escritura del código fuente de un software. Su objetivo es pautar la forma en la que el código debe ser escrito, propiciando el mantenimiento y legibilidad del mismo por los miembros del equipo de desarrollo. En el caso de la metodología seleccionada, la definición de estos estándares es decisiva para alcanzar con éxito la propiedad colectiva del código, un precepto establecido por XP y que permite a los programadores modificar el código previamente definido por otro con el fin de añadir alguna funcionalidad o corregir cierto fallo. A continuación se muestran los estándares aplicados durante la implementación de la biblioteca.

Ficheros fuente

Los nombres de los archivos *.js* generados deberán escribirse en minúscula y no podrán contener caracteres especiales. Cada archivo, tal cual muestra la Figura # 12 contendrá la definición de una única clase.

pluss.manager.js

Source | History

```
1   $.util.ns("$.Manager");
2   $.Manager = (function($) {
3
4       var store = {},
5               types = {...},
21      sequence = {},
22              unique = function(type) {...},
36              registerComponent = function(id, type, component) {...},
42              unRegisterComponent = function(id, type) {...},
55              byId = function(id) {...},
76              byType = function(type) {...};
91
92      return {...};
102
    })(jQuery);
```

Figura # 12. Archivo *.js* contenedor de una única clase.

Definición de clases

Las clases deberán nombrarse en inglés, haciendo uso de la convención "*UpperCamelCase*". En el caso de sus métodos y atributos, serán agrupados según la visibilidad de los mismos; los métodos privados incluirán el "_" (*underscore*) delante.

Métodos

Los métodos serán escritos utilizando la convención "*lowerCamelCase*" y con el objetivo de reducir el tamaño de los archivos, aquellos utilizados para la obtención y modificación de valores (*getters* and *setters*) serán agrupados bajo un mismo identificador, coincidiendo este con el nombre de la propiedad a modificar u obtener. La Figura # 13 evidencia esta regla: el método *text* agrupa las funcionalidades necesarias para dar cumplimiento a un *getText* y un *setText*.

```
39          /*Método que asigna un nuevo texto al Label o retorna el existente */
40          text: function(opt) {
41              if (opt || opt == false) {
42                  this.placeholder.text(opt);
43                  return this;
44              }
45              else {
46                  return this.placeholder.text();
47              }
48          },
```

Figura # 13. *Getter* y *Setter* bajo un mismo nombre.

Es importante destacar que el método *Class* encargado de la definición de las clases que conforman la biblioteca no empleará la convención "*lowerCamelCase*" sino "*UpperCamelCase*", debido a que la palabra *class* se encuentra reservada por el lenguaje JavaScript.

Además se deberán cumplir las siguientes reglas de formato:

- No utilizar un espacio en blanco entre el nombre de un método y el paréntesis que abre su lista de parámetros o entre la palabra reservada *function* y este último.
- La llave de apertura aparecerá en la misma línea de la declaración del método.
- La llave de cierre ocupará una línea individual y deberá ajustarse a la sentencia declaratoria de la función.

En el caso de los métodos que respondan a Eventos, estos deberán ser antecedidos por la palabra “on”.

Atributos

Los atributos de las clases y variables auxiliares serán nombrados empleando la convención “*lowerCamelCase*”, evitando el empleo de abreviaturas, excepto para los casos en los que pueda inferirse fácilmente su significado, ejemplo de ello: multi, min, max, num, info, auto, entre otras.

Estructuras repetitivas

Con el objetivo de utilizar la menor cantidad de código posible, así como garantizar la rapidez en la ejecución del mismo, estructuras como el *for* deberán crear y asignar valores a sus variables fuera de su propia inicialización. Además, se evitará en todo momento el uso de **++,** utilizando en su lugar **+=1.** Para un mejor entendimiento de esta pauta, ver la Figura # 14.

```
83      var x = 0, length = this.length;
84      for (; x < length; x += 1) {
85      //Instrucciones
86      }
```

Figura # 14. Ciclo *for* utilizando la convención definida

Declaración de variables

Una variable debe estar acompañada siempre de la palabra reservada *var*. Sin embargo y como indica la Figura # 15, siempre que sea posible deberá aprovecharse una misma línea para declarar posteriores variables.

```
88      //Incorrecto!
89      var a;
90      var b;
91      var c;
92
93      //Correcto
94      var a, b, c;
```

Figura # 15. Declaración de variables.

Nombres de espacio

Los nombres de espacio o *namespaces* serán definidos en mayúscula.

Espaciado

Deberá emplearse una línea en blanco en los siguientes casos:

- Entre las variables locales de un método y su primera sentencia.
- Entre las distintas secciones lógicas de un método.

No deberán emplearse saltos de línea posteriores a la clausura de un método de una clase. En el caso de los espacios en blanco, estos deberán utilizarse:

- Después de cada coma en las listas de argumentos.
- Entre operadores lógico-aritméticos y sus operandos.
- En las expresiones de una sentencia *for*.

Comentarios

Podrán utilizarse los comentarios de línea (//) y de bloque (/**/), siempre que no sean empleados para resaltar obviedades. Cada método poseerá un encabezado con su objetivo, así como comentarios de apoyo a las variables auxiliares que emplee.

3.2 Desarrollo de las iteraciones

"El desarrollo de las HU es guiado por las Tareas de Ingeniería, las cuales son consideradas como una herramienta de planificación esencial para el programador" [20]. Estas tareas, especifican de forma técnica los detalles relacionados con la realización de cada HU en cada iteración, enunciando las actividades necesitadas para concretar su implementación. La Tabla # 6 muestra la descripción de cada una de las iteraciones, así como las HU asignadas a las mismas.

Tabla # 6. Descripción de las iteraciones.

Release	Descripción de la iteración	HU	Duración total
1	Agregar mecanismos al núcleo con el fin de propiciar la creación de clases y gestión de componentes.	1 y 2	1.5 semanas
2	Implementar las clases de los widgets de tipo básico.	3, 4, 5, 6, 7, 8, 9, 10 y 11	4.5 semanas
3	Implementar las clases de los widgets de tipo contenedor, además implementar el cargador de dependencias y la clase del widget *Grid*.	12, 13, 14, 15, 16, 17 y 18	6 semanas

Iteración 1

En esta iteración fueron implementadas las HU "Mecanismo para la definición de clases" y "Mecanismo para la gestión de instancias de *widgets".* Ambas, contenedoras de funcionalidades críticas para la posterior creación de los componentes que conforman la biblioteca y al igual que las restantes HU, requirieron de su desglose en TI como la mostrada en la Tabla # 7 y cuya implementación se evidencia en la Figura # 17.

Tabla # 7. Tarea de ingeniería # 1.

Tarea de Ingeniería	
Número Tarea: 1	**Número Historia de Usuario:** 1
Nombre Tarea: Implementar mecanismo para la creación de clases.	
Tipo de Tarea : Desarrollo	**Puntos Estimados:** 0.25 semana
Fecha Inicio: 21/01/2013	**Fecha Fin:** 22/01/2013
Programador Responsable: José R. Guzmán Ojeda	
Descripción: Mecanismo que permite mediante el clonado de prototipos definir clases y heredar entre ellas.	

Además de la anterior tarea, la primera iteración requirió de otras 8, las cuales podrán ser encontradas en el Expediente de Proyecto asociado a la presente investigación. Cabe resaltar que algunas de estas TI

resultaron en agregados a jQuery con el fin de otorgarle la capacidad para crear *namespaces*, generar nodos, entre otras.

Iteración 2

La segunda iteración propició el desarrollo de las HU de la 3 a la 11, todas correspondientes a *widgets* básicos como: las cajas de texto y de selección, botones, etiquetas, calendarios entre otros. Estas HU implicaron la creación de mecanismos no definidos por el cliente, pero necesarios para garantizar la reutilización y rapidez en la codificación, entre los cuales pueden citarse las clases *Widget* cuya implementación podrá encontrarse en la Figura # 18 y *FormItem*, esta última con la capacidad de asociar una etiqueta a un *widget* básico, tal cual muestra la Figura # 16.

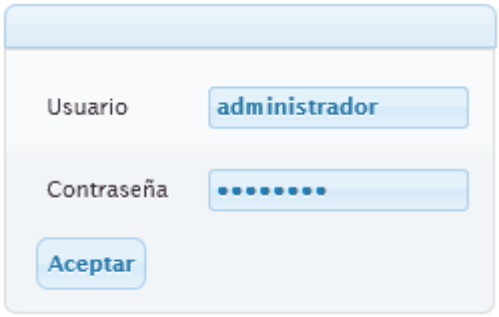


Figura # 16. Asociación entre etiqueta y caja de texto.

Iteración 3

En la última iteración fueron implementadas un total de 7 HU, de las cuales 5 correspondían a *widgets* de tipo contenedor, las dos restantes correspondientes al *widget Grid* y al "Cargador de dependencias" respectivamente. Los contenedores fueron dotados de capacidad para autoajustar y gestionar su contenido, siendo necesario para ello la creación de dos clases: *Container* y *MultiContainer*.

La HU "Cargador de dependencias" requirió el uso de funcionalidades previamente definidas en jQuery, tal cual muestra en la Figura # 19, no solo con el objetivo de propiciar la carga automática de las dependencias, sino además para permitir a los desarrolladores cargar bajo demanda, de manera síncrona o asíncrona cualquier otro archivo *.js*.

3.2.1 Implementaciones relevantes

```
14  Class: function() {
15      /*Función encargada de extender propiedades de un objeto sobre otro
16       * @target es el objeto de destino
17       * @source es el origen
18       * @override especifica si debe sobreescribir o no en el destino las
19       * propiedades ya existentes
20       * */
21      var extend = function(target, source, override) {...};
38
39      var len = arguments.length;
40      var data = arguments[len - 1];
41      var parentKlass = len > 1 ? arguments[0] : null;
42
43      var Klass, emptyParentKlass;
44      if (data.constructor === Object) {
45          Klass = function() {
46          };
47
48      } else {
49          Klass = data.constructor;
50          delete data.constructor;
51      }
52      //Si se ha especificado una clase padre desde la cual extender...
53      if (parentKlass) {
54          emptyParentKlass = function() {
55          };
56          emptyParentKlass.prototype = parentKlass.prototype;
57          Klass.prototype = new emptyParentKlass();
58          Klass.prototype.constructor = Klass;
59          Klass.base = parentKlass;
60          //Especificar "false" para evitar sobreescribir
61          //las propiedades si estas ya estaban presentes en el destino
62          extend(Klass, parentKlass, false);
63      }
64      extend(Klass.prototype, data);
65      return Klass;
66  },
```

Figura # 17. Mecanismo para la creación de clases. Iteración 1.

```
(function($) {
    //Definición del namespace
    $.Util.ns("$.UI.Widget");
    /*Cración de la Clase Widget
     * Clase abstracta, padre de todas los widgets
     * */
    $.UI.Widget = $.Util.Class({
        /*...*/
        constructor: function(config) {...},
        /*...*/
        render: function(where, effect) {...},
        /*...*/
        enable: function(opt) {...},
        /*...*/
        width: function(value) {...},
        /*...*/
        height: function(value) {...},
        /*...*/
        css: function(rule) {...},
        /*...*/
        position: function(obj) {...},
        /*...*/
        visible: function(opt) {...},
        /*...*/
        bind: function(name, fn) {...},
        /*...*/
        trigger: function(evt) {...},
        /*...*/
        destroy: function() {...}
    });
})(jQuery);
```

Figura # 18. Implementación de la clase Widget. Iteración 2.

```
(function($, w) {
//Variable encargada de almacenar el registro de todo aquello que ha sido cargado
    var loadedModules = {},
            //ubicacion por defecto de los archivos .js
            prefix = "",
            //Método auxiliar y cargador de Scripts
            getScript = function(resources, callback, type, internal) {
        //Si type es false o no esta definido, entonces hacer una llamada Síncrona
        if (!type) {...}
        var realPrefix = "";
        if (internal) {...}
        //Redefinir las Opciones de jQuery a la hora de hacer llamadas al servidor
        $.ajaxSetup({...});
        //Guardar referencia a $.getScript
        var _getScript = jQuery.getScript;
        // Si no es un Array, hacerlo
        if (!$.isArray(resources)) {...}
        var length = resources.length,
                handler = function() {...},
                deferreds = [],
                counter = 0,
                idx = 0;
        //Cargar cada uno de los Scripts(están en un Array)
        for (; idx < length; idx += 1) {...}
        //Si callback es una función, ejecutarla cuando sean cargados todos los Scripts
        if (callback && callback.prototype) {...}
    };
    //Lista de dependencias
    var dependencyList = {...};
    /*Cargar bases en caso de no estar cargadas*/
    (function() {...})();
    //Definición del namespace
    $.Util.ns("$.Loader");
    $.Loader = (function() {
        var loadCmp = function(module, callback, type) {
            if (!loadedModules[module]) {
                getScript(dependencyList[module], callback, type, true);
                loadedModules[module] = true;
            }};
        return {
            loadedModules: loadedModules,
            loadCmp: loadCmp,
            loadJs: getScript
        };
    })();
})(jQuery, window);
```

Figura # 19. Cargador de dependencias. Iteración 3.

3.3 Pruebas

En XP, la realización de pruebas es crucial para el desarrollo, estas permiten comprobar constantemente el código para así obtener un producto de mayor calidad. *"Esta metodología propone el uso de dos tipos de pruebas: unitarias y de aceptación"* [9]; las primeras, confeccionadas y utilizadas por los desarrolladores para verificar el código, mientras que las últimas permiten al cliente constatar la correcta realización de las HU definidas.

3.3.1 Pruebas unitarias

La metodología seleccionada propone la confección de estas pruebas, incluso antes de la propia implementación, así como la utilización de una herramienta que automatice la ejecución de las mismas. En tal sentido, y acorde con lo establecido en el ambiente de desarrollo, se hizo uso de la suite de pruebas QUnit, permitiendo probar cada una de las funcionalidades correspondientes a cada clase y componente.

La realización de estas pruebas requirió de la elaboración de un total de 119 Casos de Prueba (CP), los cuales a través de QUnit fueron ejecutados en cada uno de los navegadores definidos en los requisitos de software. En la Figura # 20 se muestra el CP utilizado para validar el componente Combobox y el resultado obtenido tras su ejecución en la Figura # 21.

```
module("$.UI.Combobox");
test('Creaci□n de la Clase Combobox', function() {
    equal(checkType($.UI.Combobox), "function", "La Clase Combobox se encuentra definida");
});
test('constructor', function() {
    equal(comboboxConstructor(), "combobox", "Se ha definido correctamente la clase");
});
test('readOnly (set)', function() {
    ok(setReadOnlyCombobox(), "Asignación de readOnly correcta");
});
test('readOnly (get)', function() {
    ok(getReadOnlyCombobox(), "Obtención de readOnly correcta");
});
test('width (set)', function() {
    ok(setWidthCombobox(), "Asignación de width correcta");
});
test('width (get)', function() {
    ok(getWidthCombobox(), "Obtención de width correcta");
});
test('height (set)', function() {
    ok(setHeightCombobox(), "Asignación de height correcta");
});
test('height (get)', function() {
    ok(getHeightCombobox(), "Obtención de height correcta");
});
test('visible (set)', function() {
    ok(setVisibleCombobox(), "Asignación de visible correcta");
});
test('visible (get)', function() {
    ok(getVisibleCombobox(), "Obtención de visible correcta");
});
test('enable (set)', function() {
    ok(setComboboxEnable(), "Asignación de enable correcta");
});
test('enable (get)', function() {
    ok(getComboboxEnable(), "Obtención de enable correcta");
});
```

Figura # 20. Definición de los CP correspondientes al widget Combobox.

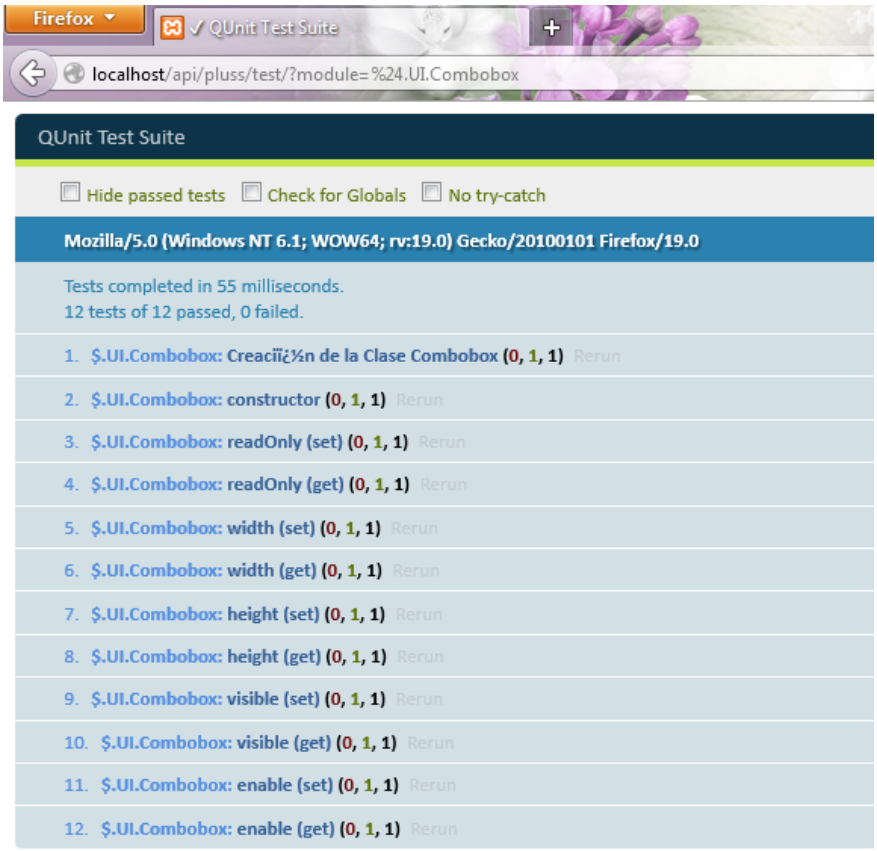


Figura # 21. Resultado de la ejecución de la prueba.

El funcionamiento de esta herramienta se detalla en el Anexo # 7, junto a una captura de pantalla con los resultados obtenidos tras la ejecución de cada una de las pruebas en el Anexo # 8.

Durante el proceso de validación, fueron detectadas 31 no conformidades, las cuales fueron corregidas en la propia iteración donde se detectaron. En el caso de los elementos gráficos, fueron validados mediante su observación directa.

3.3.2 Pruebas de aceptación

"Las pruebas de aceptación también conocidas como pruebas del cliente se enfocan en las características generales y funcionalidades del sistema" [9]. Estas pruebas derivan de las HU que han sido implementadas y su propósito es garantizar el cumplimiento de los requisitos pautados además de asegurar su correcto funcionamiento.

Para la aceptación de la biblioteca se confeccionaron 46 Casos de Prueba (CP) utilizados por el cliente para validar la misma, encontrándose almacenados en el artefacto “Casos de Prueba de Aceptación”. En una primera iteración fueron detectadas 6 no conformidades, sin embargo, la inmediatez en su resolución propició el uso de solo 2 iteraciones para completar las pruebas. La Tabla # 8 muestra la estructura y contenido de uno de estos CP.

Tabla # 8. Caso de Prueba IS-1-10, correspondiente a la HU "Gestionar *widget* Combobox".

Caso de Prueba de Aceptación	
Código Caso de Prueba: IS-1-10	**HU:** Gestionar *widget* Combobox.
Nombre de la persona que realiza la prueba: Ing. Yoander Iñiguez	
Descripción de la Prueba: Dirigida a probar la correcta visualización del *widget* Combobox.	
Condiciones de Ejecución: Incluir el archivo *.js* del *widget* y las dependencias asociadas al componente.	

Entrada / Pasos de ejecución:
Definir una nueva instancia del *widget* Combobox
var combo = new $.UI.Combobox({
fieldLabel: "Opciones",
items: [
{
text: "opción 1"
},
{
text: "opción 2"
}
] })	

Resultado Esperado: Correcta visualización del *widget* Combobox	
Evaluación de la Prueba: Satisfactoria.	

3.4 Conclusiones

El desglose en 28 TI de las 18 HU permitió a lo largo de 3 iteraciones implementar la solución propuesta, al tiempo que la definición de los estándares de codificación propició una escritura uniforme del código. Además, la realización de 119 pruebas unitarias, así como 46 Casos de Prueba de Aceptación, posibilitaron evaluar la solución propuesta, constatando así su calidad y la satisfacción por parte del cliente.

CONCLUSIONES

Una vez concluido el desarrollo de la solución propuesta, se arribó a las siguientes conclusiones:

- La elaboración del marco teórico referencial permitió la correcta selección de la metodológica y herramientas para el desarrollo de la solución.
- Se realizó el análisis de la biblioteca obteniéndose los requisitos funcionales y no funcionales del software.
- El diseño de la biblioteca sentó las bases para la posterior implementación de la misma.
- La implementación de los componentes y clases dio respuesta a los requisitos especificados por el cliente.
- La realización de pruebas unitarias y de aceptación permitió validar el producto.

RECOMENDACIONES

- Promover el resultado de la investigación dentro de la comunidad de desarrolladores, a fin de obtener nuevos componentes e integrarlos a la biblioteca.
- Implementar mecanismos para el posicionamiento de los componentes de acuerdo con diferentes *layouts*.
- Enriquecer el componente G*rid* teniendo en cuenta las funcionalidades brindadas por el *plugin* jQGrid.

REFERENCIAS BIBLIOGRÁFICAS

1. **Marrero, Carlos.** *Interfaz gráfica de usuario: Aproximación semiótica y cognitiva.* Tenerife : s.n., 2006.

2. **Deitel, Paul J y Deitel, Harvey M.** *JavaScript™ for programmers. Deitel® Developer Series.* New Jersey : Pearson Education, 2010. 0-13-700131-2.

3. *RIA (Rich Internet Aplication): La Nueva Web.* **Cabrera, Juan A.** 25, Buenos Aires : MP Ediciones, Users .code, Vol. III.

4. jQuery. [En línea] jQuery Foundation. [Citado el: 18 de Diciembre de 2012.] http://jqueryui.com/about/.

5. **Harmon, James E.** *Using the Dojo JavaScript Library to Build Ajax Applications.* s.l. : Addison Wesley, 2008. 0-13-235804-2.

6. **Techs, W3.** W3 Techs. *W3 Techs.* [En línea] Q-Success, 22 de Diciembre de 2012. [Citado el: 22 de Diciembre de 2012.] http://w3techs.com/technologies/details/js-yui/all/all.

7. *Las Metodologías de Desarrollo Ágil como una Oportunidad para la Ingeniería de Software Educativo .* **Orjuela Duarte, Ailin y Rojas, Mauricio.** 2, Medellín : s.n., 2008, Vol. 5. 1657-7663.

8. **Joskowicz, José.** *Reglas y Prácticas en eXtreme Programming.* Vigo : s.n., 2008.

9. **Pressman, Roger S.** *Ingeniería del Software, Un Enfoque Práctico .* s.l. : McGraw-Hill, 2005. ISBN: 9701054733.

10. **Proyectos, Gerencia de Sistemas y.** Oficina de Proyectos de Informática. [En línea] 26 de Noviembre de 2012. [Citado el: 08 de Enero de 2013.] http://www.pmoinformatica.com/2012/11/los-5-valores-de-la-programacion.html.

11. **Jacobson, Ivar, Booch, Grady y Rumbaugh, James.** *The Unified Software Development Process.* s.l. : Adison-Wesley, 2009.

12. **Stefanov, Stoyan.** *Object-Oriented JavaScript.* Birmingham : Packt Publishing, 2008. 978-1-847194-14-5.

13. **Benítez, Carlos.** etnassoft. [En línea] 1 de Febrero de 2011. [Citado el: 7 de Enero de 2013.] http://www.etnassoft.com/2011/02/01/qunit-testeando-nuestras-aplicaciones-javascript/ .

14. **Eguíluz Pérez, Javier.** *Introducción a CSS.* 2008.

15. **Association, IEEE Standards.** IEEE Standards. [En línea] [Citado el: 5 de Febrero de 2013.] http://standards.ieee.org/findstds/standard/1471-2000.html.

16. **Rojas C, Mauricio y Montilva, Jonás.** *Una arquitectura de software para la integración de objetos de aprendizaje basada en servicios web.* Medellín : s.n., 2011.

17. *Estilo arquitectónico para el sistema integrado de gestión Cedrux.* **Matos Arias, Yusnier y Silega Martínez, Nemury.** 1, La Habana : Universidad de las Ciencias Informáticas, 2013, Vol. 1.

18. **B, Manuel.** Programación en castellano. [En línea] Abril de 2011. [Citado el: 2 de Marzo de 2013.] http://www.programacion.com/articulo/diseno_de_software_con_patrones_114.

19. **Martínez, Ivette.** Diseño con Patrones. *Laboratorio Docente de Computación.* [Online] 2010. [Cited: Febrero 17, 2013.] http://ldc.usb.ve/~martinez/cursos/ci3715/clase9_AJ2010.pdf.

20. **Chromatic.** *Extreme Programming Pocket Guide .* s.l. : O'Reilly Media, 2003. 978-0-596-00485-9.

BIBLIOGRAFÍA

1. **Alonso, Daniel Bolaños.** *Pruebas de Software y JUnit. Un análisis en profundidad y ejemplos prácticos.* s.l.: Pearson, 2010.
2. **Association, IEEE Standards.** IEEE Standards. [En línea] [Citado el: 5 de Febrero de 2013.] http://standards.ieee.org/findstds/standard/1471-2000.html.
3. **B, Manuel.** Programación en castellano. [En línea] Abril de 2011. [Citado el: 2 de Marzo de 2013.] http://www.programacion.com/articulo/diseno_de_software_con_patrones_114.
4. **Beck, Kent.** *Extreme Programming Explained: Embrace Chang.* Longman: Addison Wesley, 2000.
5. **Benítez, Carlos.** etnassoft. [En línea] 1 de Febrero de 2011. [Citado el: 7 de Enero de 2013.] http://www.etnassoft.com/2011/02/01/qunit-testeando-nuestras-aplicaciones-javascript/.
6. **Blanchard, Jay.** *Applied jQuery, DEVELOP AND DESIGN.* Berkeley: Peachpit Press, 2012. 978-0-321-77256-5.
7. **Canós, José H, Letelier, Patricio y Penadés, Carmen.** *Metodologías Ágiles en el Desarrollo de Software.* Valencia: Universidad Politécnica de Valencia.
8. **Castledine, Earle y Sharkie, Craig.** *jQuery: Novice to Ninja.* Collingwood: SitePoint Pty. Ltd., 2010. 978-0-9805768-5-6.
9. **Chromatic.** *Extreme Programming Pocket Guide.* s.l.: O'Reilly Media, 2003. 978-0-596-00485-9.
10. **Darie, Cristian y Brinzarea, Bogdan.** *AJAX and PHP, Building Responsive Web Applications.* Birmingham: Packt Publishing, 2006. 1-904811-82-5.
11. **Deitel, Paul J y Deitel, Harvey M.** *JavaScript™ for programmers. Deitel® Developer Series.* New Jersey: Pearson Education, 2010. 0-13-700131-2.
12. **Echeverry Tobón, Luis M y Delgado Carmona, Luz E.** *Caso Práctico de la Metodología Ágil XP al Desarrollo de Software.* Pereira: Universidad Tecnológica de Pereira, 2007.
13. **Eguíluz Pérez, Javier.** *Introducción a CSS.* 2008.

14. **Enrique Hernández Orallo, José Hernández Orallo, Mari Carmen Juan Lizamora.** *C++ estándar.* s.l.: Editorial Paraninfo, 2011.

15. *Estilo arquitectónico para el sistema integrado de gestión Cedrux.* **Matos Arias, Yusnier y Silega Martínez, Nemury.** 1, La Habana: Universidad de las Ciencias Informáticas, 2013, Vol. 1.

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

17. **Fernández Nodarse, Francisco A.** *Consideraciones sobre el desarrollo basado en componentes en la Red Cubana de Ciencia.* La Habana: s.n., 2007.

18. **Foundation, The Dojo.** dojo. [En línea] http://dojotoolkit.org/.

19. **Garcia, Jesus.** The Component model and lifecycle. *Ext JS in Action.* 2009.

20. **Harmon, James E.** *Using the Dojo JavaScript Library to Build Ajax Applications.* s.l.: Addison Wesley, 2008. 0-13-235804-2.

21. **Jacobson, Ivar, Booch, Grady y Rumbaugh, James.** *The Unified Software Development Process.* s.l.: Adison-Wesley, 2009.

22. **Joskowicz, José.** *Reglas y Prácticas en eXtreme Programming.* Vigo: s.n., 2008.

23. *jQuery UI 1.7, The User Interface Library for jQuery.* Birmingham: Packt Publishing, 2009. 978-1-847199-72-0.

24. jQuery. [En línea] jQuery Foundation. [Citado el: 18 de Diciembre de 2012.] http://jqueryui.com/about/.

25. *Las Metodologías de Desarrollo Ágil como una Oportunidad para la Ingeniería de Software Educativo.* **Orjuela Duarte, Ailin y Rojas, Mauricio.** 2, Medellín: s.n., 2008, Vol. 5. 1657-7663.

26. **Marrero, Carlos.** *Interfaz gráfica de usuario: Aproximación semiótica y cognitiva.* Tenerife: s.n., 2006.

27. **Martínez, Ivette.** Diseño con Patrones. *Laboratorio Docente de Computación.* [En línea] 2010. [Citado el: 17 de Febrero de 2013.] http://ldc.usb.ve/~martinez/cursos/ci3715/clase9_AJ2010.pdf.

28. **Nicolas Marin Ruiz, Olga Pons Capote.** *Introducción A Las Bases De Datos. El Modelo Relacional.* s.l.: thomson, 2011.

29. **PRESSMAN, R.** *Ingeniería del Software: Un Enfoque Práctico. .* s.l.: 7ma edición, 2007.

30. **Pressman, Roger S.** *Ingeniería del Software, Un Enfoque Práctico.* s.l.: McGraw-Hill, 2005. ISBN: 9701054733.

31. **Proyectos, Gerencia de Sistemas y.** Oficina de Proyectos de Informática. [En línea] 26 de Noviembre de 2012. [Citado el: 08 de Enero de 2013.] http://www.pmoinformatica.com/2012/11/los-5-valores-de-la-programacion.html.

32. **Qian, Kai.** *Software Architecture and Design Illuminated.* London: Jones and Bartlett Publishers International, 2010. 978-0-7637-5420-4.

33. *RIA (Rich Internet Aplication): La Nueva Web.* **Cabrera, Juan A.** 25, Buenos Aires: MP Ediciones, Users .code, Vol. III.

34. **Rojas C, Mauricio y Montilva, Jonás.** *Una arquitectura de software para la integración de objetos de aprendizaje basada en servicios web.* Medellín: s.n., 2011.

35. **SISTEL.** *CCES_Especificación de la arquitectura del sistema.* 2013.

36. **Stefanov, Stoyan.** *Object-Oriented JavaScript.* Birmingham: Packt Publishing, 2008. 978-1-847194-14-5.

37. **Techs, W3.** W3 Techs. *W3 Techs.* [En línea] Q-Success, 22 de Diciembre de 2012. [Citado el: 22 de Diciembre de 2012.] http://w3techs.com/technologies/details/js-yui/all/all.

38. **University, Indiana.** University Information Technology Services. [En línea] Indiana University, 2005-2013. [Citado el: 02 de Noviembre de 2012.] http://kb.iu.edu/data/afhv.html.

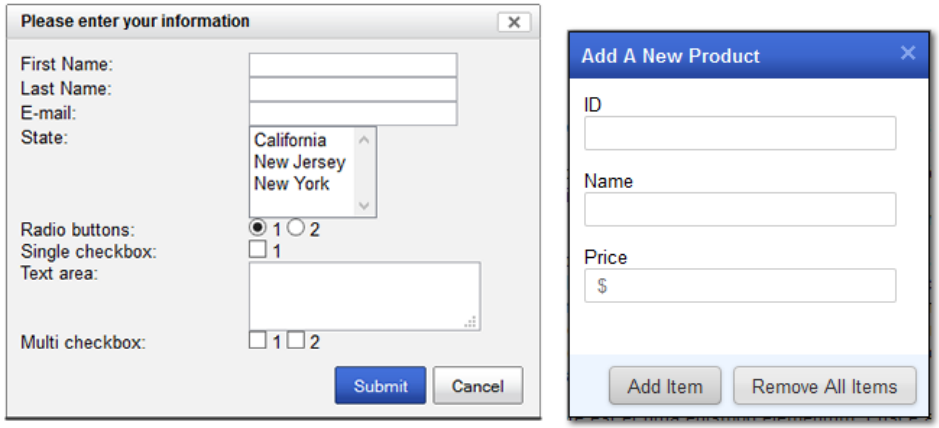
39. **Wellman, Dan.** *Learning the Yahoo! User Interface Library.* Birmingham: Packt Publishing, 2008. 978-1-847192-32-5.

40. **Zakas, Nicholas C.** *Professional JavaScript for Web Developers, 2nd Edition.* Indianapolis: Wiley Publishing, Inc., 2009. 978-0-470-22780-0.

ANEXOS

Anexo # 1. Uso de bibliotecas JavaScript por diferentes compañías.

Biblioteca	Empresa	URL
jQuery UI	AOL	aol.com
	Capcom	capcom-europe.com
	EA Games	ea.com
	ESPN	espn.go.com
	Frontier Airlines	frontierairlines.com
	Oracle	mix.oracle.com
	SourceForge	sourceforge.net
Dijit	IBM	ibm.com
	Sport	sport.es
	Shutterfly	shutterfly.com
	DHL	dhl.com
	Cox	cox.com
	Voila	voila.fr
	Idealista	idealista.com
YUI	Yahoo	mail.yahoo.com
	Flickr	flickr.com
	Reuters	reuters.com
	Aliexpress	aliexpress.com
	Wordreference	wordreference.com



YUI 2 Form YUI 3 Form

Anexo # 2. Diferencias entre el aspecto de YUI 2 respecto a YUI 3.

```
52  if (parentKlass) {
53      emptyParentKlass = function() {
54      };
55      emptyParentKlass.prototype = parentKlass.prototype;
56      Klass.prototype = new emptyParentKlass();
57      Klass.prototype.constructor = Klass;
58      Klass.base = parentKlass;
59      //Especificar "false" para no sobreescribir las propiedades si ya existieran en el hijo
60      extend(Klass, parentKlass, false);
61  }
```

Anexo # 3. Patrón "Prototype".

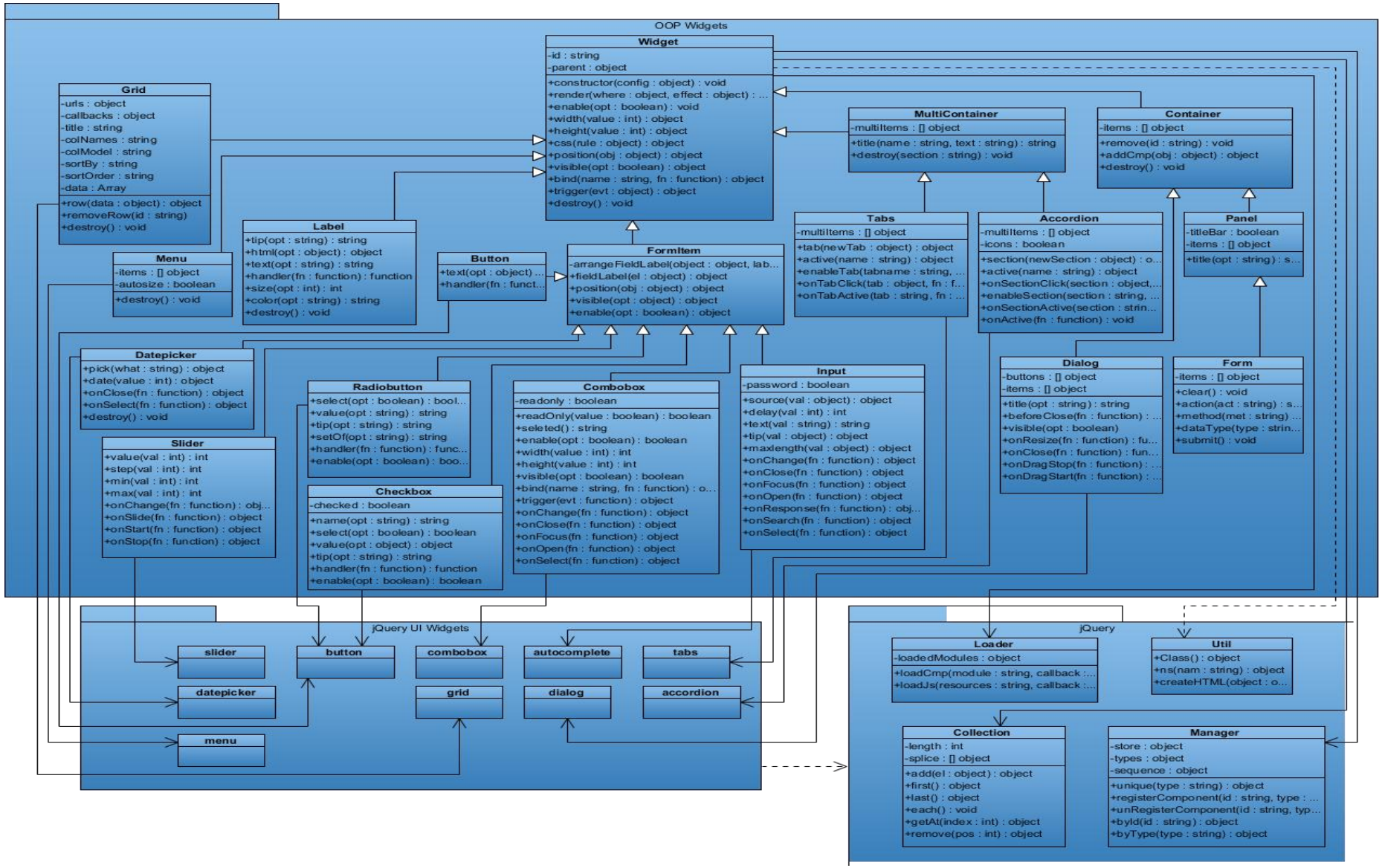
```
(function($, d) {
    //Definición del namespace
    $.util.ns("$.UI.Tabs");
    /*Creación de la clase Tabs
     * Hereda de MultiContainer
     * */
    $.UI.Tabs = $.util.Class($.UI.MultiContainer, {
        /*Constructor de la clase
         * config es el objeto de configuración del widget
         * */
        constructor: function(config) {...},
        /*Permite añadir un nuevo Tab al widget
         * newTab contiene el nombre y el contenido del nuevo Tab*/
        tab: function(newTab) {...},
        /*Elimina una sección del Widget
         * tab contiene el nombre del Tab a remover
         * */
        _removeSection: function(tab) {...},
        /*Método auxiliar utilizado por el constructor para generar
         * la estrucutra HTML necesitada por jQuery UI
         * */
        _objToHtml: function(obj) {...},
        /*Método auxiliar necesitado por _objToHtml para extraer el nombre del Tab*/
        _extractTabsName: function(obj, overridePos) {...},
        /*Método auxiliar necesitado por _objToHtml para extraer el contenido del Tab*/
        _extractTabsContent: function(obj, it) {...},
        /*Cambia el nombre un Tab*/
        _changeName: function(name, text) {...}

    });
})(jQuery, document);
```

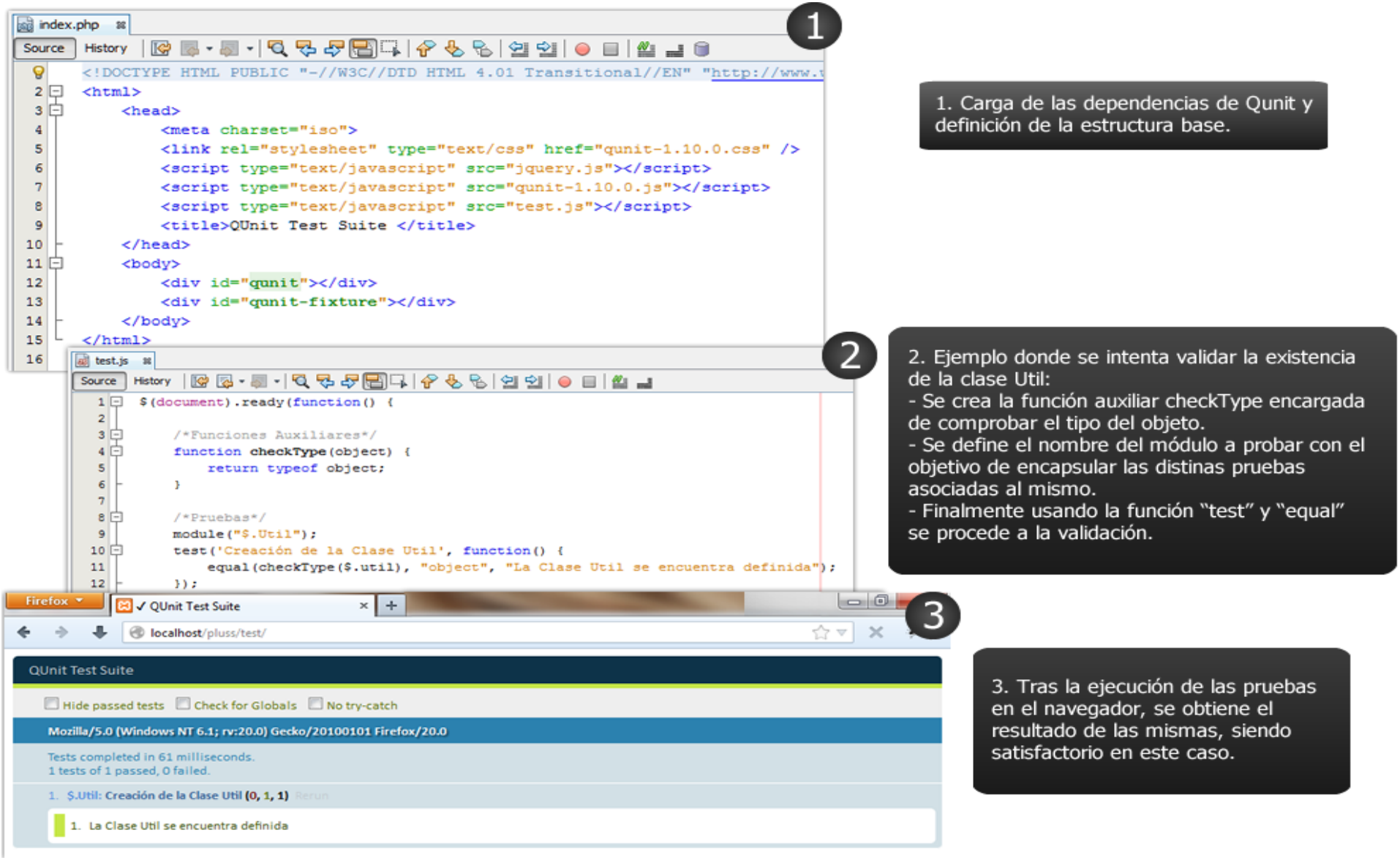
Anexo # 4. Patrón "Adapter".

```
133         /*Destruye el componente
134          * Lo elimina del registro de instancias del Manager
135          * Elimina las referencias en jQuery UI
136          * Lo elimina del DOM
137          * */
138         destroy: function() {
139             $.Manager.unRegister(this.id, this.type);
140             this.self.destroy();
141             this.self = undefined;
142             $(this.baseLayer).remove();
143             delete this;
144             return undefined;
145         }
```

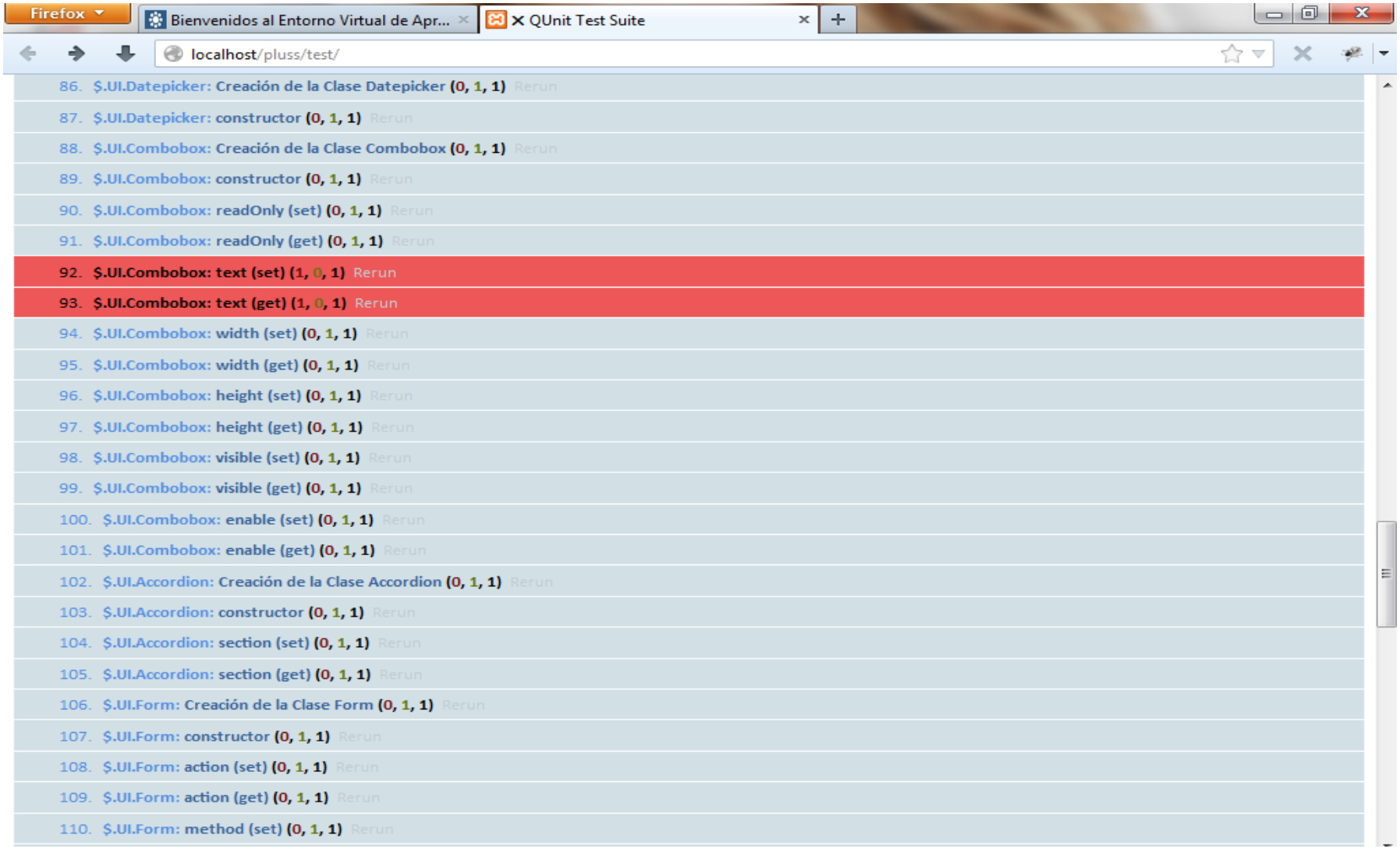
Anexo # 5. Patrón "Facade".



Anexo # 6. Diagrama de Clases.



Anexo # 7. Principio de funcionamiento de la herramienta QUnit.



Anexo # 8. Resultados obtenidos en las pruebas unitarias.

GLOSARIO

API: Siglas de (*Application Programming Interface* - Interfaz de Programación de Aplicaciones). Grupo de rutinas que provee un sistema operativo, una aplicación o una biblioteca, que definen cómo invocar desde un programa un servicio que prestan. En otras palabras, una API representa un interfaz de comunicación entre componentes software.

Biblioteca: Colección o conjunto de subprogramas o procedimientos usados para desarrollar software.

Call-back: Se refiere a aquella función o procedimiento que será ejecutado con posterioridad al completamiento de una acción.

Clase: Contenedor de uno o más datos (variables o propiedades) junto a las operaciones de manipulación de dichos datos (funciones o métodos).

Dependencias: Funcionalidades o funcionalidad requerida por otra para llevar a cabo su labor.

DOM: Siglas de (*Document Object Model* - Modelo de Objetos de Documento). Especificación que determina cómo los objetos (texto, imágenes, enlaces, etc.) en una página web son representados.

ECMAScript: Lenguaje de tipos dinámicos ligeramente inspirado en Java y otros lenguajes del estilo de C. Tiene características de programación orientada a objetos mediante objetos basados en prototipos y pseudoclases.

Experiencia de usuario: Conjunto de factores y elementos relativos a la interacción del usuario con un entorno o dispositivo concretos y cuyo resultado es la generación de una percepción positiva o negativa de dicho servicio, producto o dispositivo.

Find something, manipúlate it: Filosofía de funcionamiento de jQuery, establece que para realizar una acción sobre un elemento este debe ser encontrado primeramente a través del selector CSS de la biblioteca.

Flash: Tecnología desarrollada por *FutureSplash*, adquirida por Macromedia y posteriormente por Adobe (al comprar Macromedia) su uso permite mostrar animaciones vectoriales y crear aplicaciones Web que incluyen flujo de audio y video, así como interactividad.

Namespace: Medio para organizar variables, clases o módulos dentro de un entorno, agrupándolas de un modo más lógico y jerárquico.

OOP: Paradigma de programación. Tipo de lenguaje de programación basado en la idea de encapsular estado y operaciones en objetos.

Propiedad colectiva del código: Filosofía aplicada en la metodología XP, permite que cualquiera contribuya al desarrollo de cualquier parte del proyecto. Cualquier programador podrá cambiar una línea de código para añadir funcionalidad o eliminar algún fallo.

RIA: Siglas de *Rich Internet Application*. Aplicaciones web con características de aplicaciones de escritorio las cuales hacen uso de AJAX para establecer una fluida comunicación entre el cliente y el servidor.

Silverlight: Tecnología que permite la creación de aplicaciones web agregando funciones multimedia como la reproducción de vídeos, gráficos vectoriales, animaciones e interactividad, es propiedad de Microsoft.

Usabilidad: Término que hace referencia al grado de eficacia, eficiencia y satisfacción con la que usuarios específicos alcanzan sus objetivos en un contexto de uso específico.

Web 2.0: Transición que se ha dado de aplicaciones tradicionales hacia aplicaciones que funcionan a través del navegador y están enfocadas al usuario final. Se trata de aplicaciones que generen colaboración y de servicios que reemplacen a las aplicaciones de escritorio.

Widgets: Elemento o componente de una interfaz gráfica que muestra información con la cual el usuario puede interactuar. Por ejemplo: ventanas, cajas de texto, listas, menús desplegables, entre otros.