Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación
Centro de Ingeniería de Software y Sistemas

Desarrollo de una Aplicación Web para la elaboración automatizada del diseño de los exámenes escritos de los cursos de extensión de la Escuela de Idiomas Modernos de la Facultad de Humanidades y Educación de la Universidad Central de Venezuela

Trabajo Especial de Grado
presentado ante la Ilustre
Universidad Central de Venezuela
Por el Bachiller
Andrés Alejandro Viviani Querales
para optar al título de
Licenciado en Computación

Tutores:
Profa. Jossie Zambrano
Prof. Sergio Rivas

Caracas, 21/09/2012

Acta

Quienes suscriben, miembros del jurado designado por el Consejo de la Escuela de Computación de la Facultad de Ciencias de la Universidad Central de Venezuela, para examinar el Trabajo Especial de Grado titulado: Desarrollo de una *Aplicación Web* para la elaboración automatizada del diseño de los exámenes escritos de los cursos de extensión de la *Escuela de Idiomas Modernos* de la *Facultad de Humanidades y Educació*n de la *Universidad Central de Venezuela*, presentado por el bachiller Andrés A. Viviani Q., C.I.: 19.499.931, a los fines de optar por el título de Licenciado en Computación, dejan constancia de lo siguiente: Leído el trabajo por cada uno de los miembros del jurado, se fijó el día Viernes 21 de Septiembre de 2012, a las 10:00 am, para que su autor lo defendiera en forma pública en la Sala Planta Alta 3 de la Escuela de Computación, mediante una presentación oral de su contenido, luego de lo cual respondió a las preguntas formuladas. Finalizada la defensa pública del Trabajo Especial de Grado, el jurado decidió aprobarlo con la nota de _____ puntos. En fe de lo cual se levanta la presente Acta, en Caracas a los veintiún (21) días del mes de Septiembre del año dos mil doce (2012).

_____	_____
Profa. Jossie Zambrano (Tutor)	Profa. Joyce Gutiérrez (Jurado)
_____	_____
Prof. Sergio Rivas (Tutor)	Prof. Carlos Saavedra (Jurado)

Dedicatoria

A ti Rudy.

Que muchas veces te sentaste a mi lado a hacerme compañía para no sentirme solo en el desarrollo de mis actividades. Que siempre me diste la fuerza y el valor para afrontar mis dificultades. Que hiciste que creyera en mí por sobre todas las cosas.

A ti Domenico.

Que junto a mi madre me diste la vida. Que me inculcaste valores para ser una persona de bien, por tu apoyo incondicional, por darme la motivación necesaria para superarme cada día y hacerme entender la importancia y el valor de los estudios.

A ti Viviana.

Que siempre me apoyas y crees en mí, que a pesar de las adversidades, siempre estás ahí. Te amo!

A ti Pedro.

Para que esto te sirva de motivación y tengas la certeza que con esfuerzo todas las cosas se pueden lograr.

A ustedes abuelos.

Que me han visto crecer. Me enorgullece poder dedicarles este logro y sé que ustedes están orgullosos también.

A ustedes suegros.

Que siempre apostaron a mí, aún cuando me caía y todavía no me levantaba.

A ti Jossie.

Que me adoptaste como otro hijo y me impulsaste a lograr esta meta.

A ti Sergio Rivas.

Que creíste en mí y me ayudaste a desarrollar el conocimiento necesario en el área de mi profesión.

Agradecimientos

A ti Rudy.

Gracias por darme el ánimo necesario para lograr todas mis metas. Gracias por siempre hacerme saber que solo yo coloco el límite para lograr lo que quiero.

A ti Domenico.

Gracias por guiarme por el camino correcto. Gracias por aceptar, aún en desacuerdo, todas mis decisiones.

A ti Viviana.

Gracias por no dejar que abandonara cuando lo creía todo perdido. Gracias por no permitirme dejar para mañana lo que podía hacer hoy. Gracias por estar siempre ahí apoyándome. Gracias por ser como eres conmigo.

A mis hermanos.

Gracias por motivarme con sus logros y éxitos. Gracias por motivarme con sus palabras de aliento, que aunque se dieron en pocas ocasiones debido a las distancias, fueron importantes.

A ustedes suegros.

Gracias por tener esa fe en mí. Gracias por siempre hacer del largo camino solo un paso. Gracias por el ánimo.

A ti Sergio Sánchez.

Gracias por el ánimo. Gracias por estar pendiente de todo y de alguna u otra forma involucrarte en lo que hacía.

A mi familia.

Gracias porque a pesar de la distancia, siempre estuvieron pendientes de mis estudios. Siempre creyeron que yo era lo bastante bueno. Gracias!

A ti Jossie.

Gracias por todos tus consejos, tus guías, tu apoyo. Gracias por estar ahí siempre pendiente de mí durante toda mi carrera universitaria. Gracias por todas esas cosas que aprendí. Gracias!.

A ti Sergio Rivas.

Gracias por tomar parte de tu valioso tiempo para enseñarme muchas de las cosas que hoy sé. Gracias por confiar en mí para muchas cosas!.

A FUNDEIM.

Gracias por darme la oportunidad de trabajar con ustedes en los distintos proyectos que se desarrollaron para la fundación. Gracias!

A ti Javier Santos.

Gracias por esos valiosos consejos durante mi carrera. Gracias por brindarme tu ayuda de forma incondicional y sin esperar recibir nada a cambio.

A la DCE.

Gracias a todo el personal de Control de Estudios, que por más de un año me apoyaron de una u otra forma. Gracias Mila! que siempre estuviste pendiente que todo me saliera bien, por regañarme cuando debías, por todo!. Gracias Any! que presionaste bastante para terminar cada una de mis obligaciones. Gracias Dinora, Joa, que siempre estuvieron pendiente de mí. Gracias a la profesora Damaris, por sus buenos deseos y energías positivas. Gracias a todos!.

A mis compañeros.

Gracias a todos ustedes, con los que compartí muchas horas cada día. Gracias porque siempre nos apoyamos y nunca el éxito fue una competencia.

A mis amigos del liceo.

Gracias muchachos. Gracias por compartir parte de esta preparación. Gracias por todos esos momentos. Gracias también, porque a pesar de la distancia, siempre se acuerdan de mí, así como yo de ustedes.

A mis profesores.

Gracias a todos esos profesores que fueron excelentes en cada asignatura que cursé y aquellos que, sin darme clases, se portaron excelente conmigo. Gracias a Eleonora, Eugenio, nuevamente a Jossie y a Sergio, Néstor, María Elena, Inés y muchos otros. Gracias a todos!

A todos.

Gracias a todas esas personas que formaron parte de mi vida durante mi carrera universitaria. Gracias a todos aquellos que alguna vez me apoyaron directa o indirectamente. A todas esas personas, simplemente gracias!.

Resumen

La Fundación Escuela de Idiomas Modernos (FUNDEIM) de la Facultad de Humanidades y Educación de la Universidad Central de Venezuela, actualmente realiza evaluaciones escritas en los distintos cursos que se dictan de diversos idiomas. El proceso a seguir para el desarrollo de los exámenes no se encuentra automatizado, motivo por el cual surge el planteamiento de realizar este *Trabajo Especial de Grado*, que tiene como objetivo general el desarrollo de una *Aplicación Web* para la elaboración automatizada del diseño de los exámenes escritos de los cursos de extensión de la *Escuela de Idiomas Modernos* de la *Facultad de Humanidades y Educació*n de la *Universidad Central de Venezuela*, que se realizan en *FUNDEIM*, con el fin de reducir costos tanto en tiempo como en recursos humanos, para que puedan ser utilizados en labores de coordinación, planificación, entre otros. Para la implementación de este sistema se aplica una instancia del *método de desarrollo de software AgilUs*, que está enfocado en el usuario final y en la incorporación de la usabilidad desde etapas tempranas del desarrollo de software.

Palabras Clave:

Evaluaciones, FUNDEIM, Métodos Ágiles, Aplicación Web, AgilUs, aTesT.

Contactos

Andrés Viviani

andresviviani1@gmail.com

Jossie Zambrano

Escuela de Computación

jossie.zambrano@gmail.com

Sergio Rivas

Escuela de Computación

sergiorivas@gmail.com

Índice general

Índice de imágenes

Índice de Tablas

Introducción

Actualmente, las organizaciones son concebidas como entidades procesadoras de información, independientemente de su actividad, ya que todas tienen necesidad de obtener y analizar información sobre mercados, costos, ventas y procesos de producción a todo nivel. Esta información procede de fuentes internas y externas, y una vez procesada y utilizada, genera nueva información que será difundida dentro y fuera de la organización. Los sistemas de software que son diseñados para la automatización de procesos, son cada vez más demandados para el control de actividades y procesos dentro de cada organización. Esto se debe a la eficiencia y eficacia que ofrecen estos sistemas para la realización de las actividades que conforman el núcleo principal de las empresas.

La *Fundación Escuela de Idiomas Modernos (FUNDEIM)* de la *Universidad Central de Venezuela* es una organización que imparte cursos en cinco (5) idiomas: Inglés, Alemán, Francés, Italiano y Portugués. Todos los cursos están dirigidos al público mayor de quince (15) años de edad, a excepción del idioma inglés, que ofrece cursos para niños entre nueve (9) y once (11) años, y para adolescentes entre doce (12) y catorce (14) años. Cada curso se compone de nueve (9) niveles, a excepción de los cursos para niños y adolescentes que tienen cuatro (4) y seis (6) niveles respectivamente.

Este trabajo se enfoca en el desarrollo de una *Aplicación Web* para diseñar los exámenes de los cursos de extensión de la *Escuela de Idiomas Modernos de la Facultad de Humanidades y Educación de la Universidad Central de Venezuela*, que apoye la realización de los exámenes escritos, con el fin de reducir costos en tiempo y en recursos humanos, para que puedan ser utilizados en labores de coordinación y planificación. Para ello es necesario hacer uso de un conjunto de tecnologías de vanguardia, para el momento de esta investigación, que facilitan el desarrollo de sistemas web. Entre ellas: *HTML, CSS, Javascript, AJAX, JQuery, JQuery UI, Ruby on Rails, HAML, SASS y SCSS, Coffeescript, RVM y MySQL*. Además, es imprescindible implementar y hacer uso de un conjunto de métodos y técnicas que garanticen

la realización de un software de calidad. En este trabajo el método a seguir para el desarrollo de software es *AgilUs,* propuesto por la profesora Alecia Eleonora Acosta del centro de *Ingeniería de Software* y *Sistemas (ISYS)* de la *Escuela de Computación de la Facultad de Ciencias de la Universidad Central de Venezuela,* que divide el desarrollo de un sistema de software en cuatro (4) etapas con distintas técnicas y artefactos para cada una. Además, enfoca el desarrollo en el usuario final y toma en cuenta la usabilidad desde etapas tempranas del desarrollo de software.

Esta investigación se estructura tal como se describe a continuación:

Capítulo 1: *Tecnologías Web.* En este capítulo se describe las *tecnologías web* de vanguardia que son utilizadas para el desarrollo de este trabajo. Se agrupan en *tecnologías del lado del cliente*, conformadas por *HTML, CSS, Javascript, AJAX, JQuery, JQuery UI* y *otros plugins de JQuery*; *tecnologías del lado del servidor,* conformadas por *Ruby on Rails, HAML, SASS y SCSS, Coffeescript*; y tecnologías relacionadas constituidas por *RVM y MySQL* que no son propias de una *Aplicación Web* pero que facilitan el desarrollo de las mismas.

Capítulo 2: *Diseño de las evaluaciones escritas de FUNDEIM.* En este capítulo se presenta la *Fundación Escuela de Idiomas Modernos* como organización y la problemática existente dentro de la misma, así como los distintos sistemas de evaluación de un idioma, los exámenes dentro de la fundación y, por último, el objetivo general, los objetivos específicos y la justificación de esta investigación.

Capítulo 3: *aTesT*. Este capítulo describe el *método de desarrollo de software AgilUs.* Además se detalla la implementación del sistema, denominado *aTesT*. Este sistema se realiza en dos (2) iteraciones en donde se desarrollan las distintas actividades dentro de las cuatro (4) etapas que propone el método *AgilUs*. Las iteraciones que conforman este capítulo son: *Banco de preguntas*, que explica el desarrollo concerniente al módulo de preguntas del sistema; y *generación de exámenes*, que detalla el desarrollo referente a la gestión de exámenes

dentro de *aTesT*.

Finalmente, se exponen las conclusiones y recomendaciones, y se listan las referencias bibliográficas que brindan soporte a este trabajo.

Capítulo 1 *Tecnologías Web*

A partir de la masificación de internet y más aún en los tiempos modernos, la Web es una gran herramienta de acceso a la información. En términos generales, una *Aplicación Web* es un sistema de software que es accedido vía web por una *red* (Mora, Programación en Internet: Clientes Web, 2001). También suele usarse este término para identificar aquellos programas que son ejecutados por un *navegador web*. Una gran ventaja de las *Aplicaciones Web* es la facilidad de mantenimiento y actualización sin la necesidad de instalar un software en muchos *clientes*. Un *cliente* puede ser una computadora, un teléfono celular, una consola de videojuegos, entre otros.

Las *Tecnologías Web* implican un conjunto de *técnicas* que hacen posible lograr mejores resultados a la hora del desarrollo de *Aplicaciones Web*. Estas *técnicas* permiten la distribución de información en hipertexto o hipermedios enlazados y accesibles a través de *Internet* (Mora, Programación en Internet: Clientes Web, 2001). La información que se distribuye puede contener texto, imágenes, videos u otros contenidos multimedia, y la navegación a través de ellas se realiza usando hiperenlaces. Las *Tecnologías Web* se dividen en dos grandes grupos, *tecnologías del lado del cliente* y *tecnologías del lado del servidor*.

En este capítulo se presentan las tecnologías que son utilizadas para desarrollar la *Aplicación Web* de este trabajo. Entre ellas se encuentran: *Tecnologías del lado del cliente*, *tecnologías del lado del servidor* y las *tecnologías relacionadas,* que no son propias de una *Aplicación Web,* pero facilitan su desarrollo.

1.1. Tecnologías del lado del Cliente

Las *tecnologías del lado del cliente* son aquellas que pueden ser ejecutadas directamente por el *navegador web* (Wikipedia, Client-side, 2011). Entre las que se utilizan para esta investigación se encuentran: *HTML*, *CSS*, *Javascript*, *AJAX*, *JQuery*, *JQuery UI* y otros *plugins*

de *Jquery*. A continuación se describe cada una de ellas.

1.1.1. *HTML*

El lenguaje de marcado de hipertexto *HTML* (*HyperText Markup Language*, por sus siglas en inglés), es el lenguaje de marcado más utilizado para la elaboración de *páginas web*. Es usado para describir la estructura y el contenido en forma de texto, así como para complementar el texto con objetos tales como imágenes (W3C, HTML and CSS, 2011). *HTML* se escribe en forma de *etiquetas (< >)*. En la imagen 1.1 puede verse la estructura básica de un archivo en formato *HTML*.

```
1 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML
      4.01//EN" "http://www.w3.org/TR/html4/
      loose.dtd" >
2 <html lang="es">
3   <head>
4     <title> EJEMPLO DE TITULO </title>
5   </head>
6   <body>
7     <p> EJEMPLO DE UN PARRAFO </p>
8   </body>
9 </html>
```

Imagen 1.1: Estructura básica de un archivo en formato HTML.

HTML indica a los *navegadores web* la estructura del contenido de una *página web*. En sus inicios, *HTML* fue diseñado principalmente como un lenguaje para describir semánticamente los documentos científicos. A pesar de su diseño general, las adaptaciones que se han hecho a lo largo de los años le han permitido que sea utilizado para describir una serie de otros tipos de documentos.

HTML 5 es la versión actual de *HTML* al momento de este trabajo, la cual surge como una evolución de las versiones anteriores y pretende dar solución a la mayoría de los problemas con los que un desarrollador web se encuentra.

Las novedades de esta versión se centran en facilitar la implementación de *Aplicaciones Web*, avanzar hacia la *web semántica* y evitar un poco aquellos elementos que rompen con el esquema web. La *web semántica* se refiere a aquellas aplicaciones dotadas de mayor significado en la que cualquier usuario podrá encontrar respuestas a sus inquietudes de forma más eficiente.

HTML 5 incluye diversas *Interfaces de Programación* de *Aplicaciones (API,* del inglés *Application Programming Interface*) que permiten que un *navegador web* pueda interpretar código *HTML* sin necesidad de la instalación de *plugins* adicionales como *flash.* Un *API* es el conjunto de funciones, procedimientos o métodos agrupados en una librería, disponibles para ser utilizadas.

HTML 5 no sólo se trata de incorporar nuevas *etiquetas* o eliminar otras, sino que supone mejoras en áreas que hasta ahora quedaban fuera del lenguaje y para las que se necesitaba utilizar otras tecnologías.

Algunos de los aspectos más interesantes incluidos en esta versión de *HTML* se muestran a continuación:

- **Estructura de la *aplicación web*:** La mayoría de las *aplicaciones web* tienen una estructura común, que suponían un uso abusivo, pero necesario, de la *etiqueta* <*div*>. Esta estructura consta de elementos básicos tales como *cabecera, pie, sección*, etc. *HTML 5* permite agrupar todas estas partes de una *aplicación web* en nuevas *etiquetas* que representarán cada una de ellas como partes típicas de una *página* (ver imagen 1.2). Estas nuevas *etiquetas* no solo buscan eliminar el uso excesivo de la *etiqueta* <*div*>, sino que tratan de introducir semántica a lo que significa cada parte del contenido dentro de la aplicación.

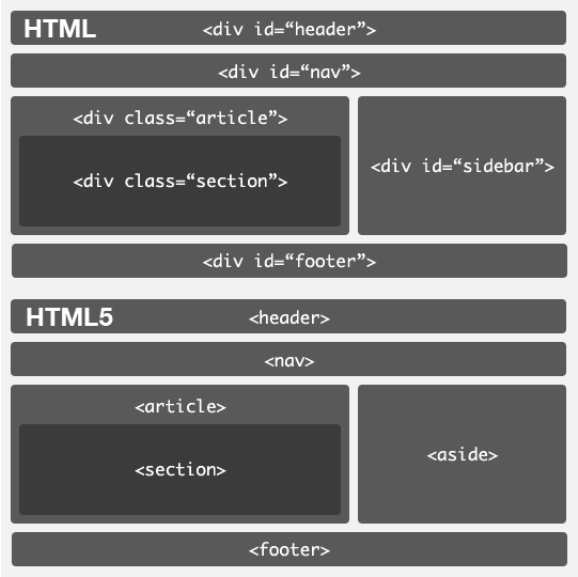


Imagen 1.2: Estructura básica de una *aplicación web* tradicional y su equivalente en *HTML 5.*

- **Etiquetas para contenido específico:** Con las versiones anteriores, se hacía uso de una única *etiqueta* para incorporar contenido multimedia, como animaciones *flash*. Ahora es posible incorporar este tipo de contenido con *etiquetas* específicas como *audio, video*, etc. Esto permite que el *navegador* pueda interpretar código *HTML* e incluir *audio, video,* etc, sin la necesidad de instalar ningún *plugin* adicional.

- **Mejores formularios:** Se incluyen nuevos tipos de datos para el elemento *input* que permiten una mejor interacción con el usuario, así como también validaciones (ver imagen 1.3) sin la necesidad de otro lenguaje como *Javascript*. Algunos de estos tipos de datos son *datetime, datetime-local, date, month, week, time, number, range, email, url,* entre otros.

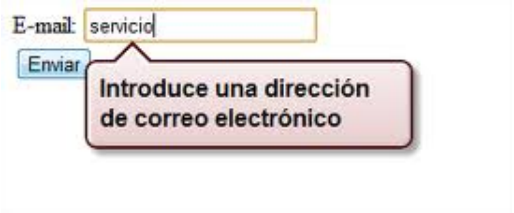


Imagen 1.3: Validación de un campo con *HTML 5.*

Asimismo, se eliminan algunos elementos ó características que a pesar de ser en su mayoría muy utilizados, son puramente de presentación, es decir, no tienen semántica y todo lo referente al tema estético de una página web debe ser tratado con *CSS*, explicado en la sección 1.1.2.

1.1.2. *CSS*

Con la evolución de la tecnología y los avances en el desarrollo de las *Aplicaciones Web*, es común que el aspecto visual de los sitios web sea actualizado constantemente, ya sea para una mejor apariencia, interacción con el usuario final o por el dispositivo tecnológico que se esté utilizando para acceder al *sitio web* (una computadora de escritorio, una laptop, un teléfono móvil, entre otros). Cambiar este aspecto visual directamente en el código de la *Aplicación Web* puede resultar un proceso largo y engorroso, por este motivo se proponen las *Hojas de Estilo en Cascada* (*Cascading Style Sheets*, por sus siglas en inglés).

CSS es un lenguaje usado para definir la presentación de un documento estructurado escrito en *HTML* o *XML* (y por extensión en *XHTML*) (W3C, HTML and CSS, 2011). El objetivo es separar el contenido de la presentación. Por lo tanto, con un lenguaje de marcado (como *HTML*) se indica al navegador como se estructura el documento, mientras que con *CSS* se indica como debe ser mostrado.

Existen diversas formas de adjuntar información de estilo a una página web:

1. **Un estilo en línea (*inline*):** Consiste en agregar propiedades *CSS* dentro de una etiqueta *HTML*, esto hace que dicho estilo sólo tenga alcance dentro de esa etiqueta por lo que resulta engorroso y poco práctico agregar estilo de esta forma. Además, hace que el código *HTML* sea extenso y en algunos casos incomprensible.

2. **Una hoja de estilo interna:** Es un forma de agregar estilo un poco menos intrusiva que la anterior, en donde el estilo se coloca dentro del *head* entre etiquetas *<style>...</style>*. Esta forma, a pesar de ser un poco mejor que la manera intrusiva, sigue

siendo poco práctica ya que se debe copiar todo el estilo de una página a otra lo que puede acarrear problemas de sincronización. Además, hace del código *HTML* un código extenso.

3. **Una hoja de estilo externa:** Es una *hoja de estilo* que está almacenada en un archivo diferente al archivo donde se almacena el código de la *página web*. Esta es la mejor práctica, porque separa completamente las reglas de formato para la página *HTML* de la estructura básica de la página. Basta solamente con hacer referencia a ese archivo para que la *página web* tenga el aspecto deseado. Además trae grandes beneficios en cuanto a la sincronización.

Antes de la existencia de *CSS*, la única forma de componer espacialmente una página era el uso de *tablas*. Aunque es una técnica cómoda y versátil, se está usando un elemento con una semántica particular, que es la de expresar información tabular, solamente por su efecto en la presentación.

La introducción de *CSS* ha permitido en muchos casos reemplazar el uso de *tablas*. Sin embargo, *CSS* no permite aún la versatilidad que ofrecían las tablas, lograr un diagramado de una página compleja suele ser una tarea difícil en *CSS* y las diferencias entre navegadores dificultan aún más la tarea.

Las versiones anteriores de *CSS* tenían muchas utilidades para aplicar estilos a las páginas web. Sin embargo, los desarrolladores usaban diversas técnicas para conseguir efectos que *CSS* por sí solo no proporcionaba. Por este motivo surge la versión actual, al momento de este trabajo, de las *hojas de estilo en cascada CSS 3*, la cual ofrece diversos estilos, muy demandados por el usuario final, sin la necesidad de técnicas adicionales.

Algunas novedades que ofrece *CSS 3* se listan a continuación:

- ***border-radius:*** este efecto permite diseños web con esquinas redondeadas que, a pesar de ser un efecto muy habitual, no tenía manera sencilla de lograrse.

- ***box-shadow:*** permite dibujar una sombra alrededor de cualquier elemento <*div*>. Anteriormente, una de las técnicas que se utilizaban era la de colocar un <*div*> sobre de otro y colocar en negro, para dar el efecto de sombra, aquel que estuviera debajo.
- ***text-shadow:*** al igual que el efecto anterior, permite crear sombras en el texto.
- ***@font-face:*** *CSS*3 permite mostrar textos con fuentes que el usuario no tenga instaladas en su equipo. Bastante útil para diseños en los que se requiera usar una fuente específica.

Estos son apenas cuatro (4) de los muchos efectos que provee *CSS 3*, que además de facilitar el desarrollo de *Aplicaciones Web*, en cuanto a apariencia se trata, hace que los sitios web sean menos pesados, consuman menos tiempo de producción y realicen menos peticiones al servidor.

Smashing Magazine es un *sitio web* y un *blog* que ofrece recursos y asesoramiento a los desarrolladores y diseñadores web. Este sitio evalúa software, aplicaciones y herramientas basadas en web. Una de sus evaluaciones se basó en comparar una versión anterior de *CSS* con la versión tres (3) y los resultados se pueden apreciar en la imagen 1.4.

	CSS	CSS3	Results
Production time	73 minutes	49 minutes	CSS3 33% faster
Size	849.2 KB	767.9 KB	CSS3 9.5% smaller
Requests	22	12	CSS3 45% fewer

Imagen 1.4: Resultados *CSS* vs. *CSS 3* (http://coding.smashingmagazine.com/2011/04/21/*CSS*3-vs-*CSS*-a-speed-benchmark/).

Como se puede apreciar en la imagen 1.4, los resultados son considerablemente notables. El tiempo de producción se optimiza en un *33 %*, el tamaño del archivo es *9.5 %* más ligero y se reducen en un *45 %* las peticiones al servidor.

Una debilidad ligada al uso de las hojas de estilo consiste en que cada *navegador* interpreta de forma independiente los estilos. Esto representa un inconveniente al momento de desarrollar *Aplicaciones Web*, ya que se pueden presentar vistas con el aspecto no deseado. Algunas medidas que se suelen tomar para combatir este inconveniente son las siguientes:

- Colocar el estilo en varias de sus formas según el navegador (ver imagen 1.5).

```
1 .miclase{
2 border-radius: 15px;
3 -moz-border-radius: 15px; #funciona
      en FireFox
4 -webkit-border-radius: 15px; #
      funciona en Safari y Chrome
5 }
```

Imagen 1.5: Estilo en varias de sus formas, para mayor compatibilidad con los navegadores web.

- Verificar que *navegador web* se está utilizando y cargar la hoja de estilo adecuada.

A pesar de esto, el uso de *hojas de estilos* presenta diversas ventajas:

- Permite separar el contenido de la manera en como deben mostrase las *Aplicaciones Web*.
- Facilidad en la actualización en cuanto al aspecto de los *sitios web*.
- Una página puede disponer de diferentes *hojas de estilo* según el dispositivo que la muestre o, incluso, a elección del usuario.
- En la mayoría de los casos, el documento *HTML* en sí mismo es más claro de entender y se consigue reducir considerablemente su tamaño.

1.1.3. *Javascript*

Javascript se propone para brindar a los desarrolladores de *Aplicaciones Web* la facilidad de crear páginas que sean capaces de interactuar con los usuarios de una mejor manera, ya que se necesitaba crear sitios web de mayor complejidad y dinamismo. El *HTML* solo permitía crear páginas estáticas donde se podían mostrar textos con estilos, pero esto no era suficiente para las necesidades por parte de los usuarios.

Javascript es un lenguaje de programación interpretado y diseñado para complementar las capacidades de *HTML* (Wikipedia, Javascript, 2011). El código de *Javascript* es enviado al cliente como parte del código *HTML* de una página, y puede ser utilizado para lograr mejoras en la interfaz de usuario y para crear páginas web dinámicas.

Existen tres (3) formas de introducir código *Javascript* en una página web, explicadas a continuación:

1. **Directamente en las etiquetas *HTML*:** Análogamente a la inclusión *inline* de *CSS*, esta forma consiste en introducir el código *Javascript* dentro de una etiqueta *HTML*. El mayor inconveniente de este método es que complica el mantenimiento del código *Javascript*, así como también incrementa el tamaño del archivo *HTML*.

2. ***Javascript* interno:** El código *Javascript* se encierra entre etiquetas <*script*> <*/script*> y se incluye en cualquier parte del documento.

3. ***Javascript* externo:** Las instrucciones *Javascript* se pueden incluir en un archivo externo para que las *Aplicaciones Web* puedan referenciarlas. Aunque es correcto incluir cualquier bloque de código *Javascript* en cualquier zona de la página, es buena práctica hacerlo dentro de la cabecera del documento, dentro de la etiqueta <*head*>.

Una de las características principales de *Javascript* es que es orientado a eventos, esto quiere decir que la interacción con el usuario se consigue mediante la captura de los eventos que éste produce. Un evento es una acción del usuario ante la cual puede realizarse algún proceso

(por ejemplo, el cambio del valor de un formulario, la pulsación de un enlace, entre otros). La tabla 1.1 muestra algunos de los manejadores de eventos más utilizados en *Javascript*.

Evento	**Objetos para los que está definido**
onAbort	Image
onBlur	Button,Checkbox,FileUpload,Layer, Password, Radio, Reset, Select, Submit, Text, Textarea, window
onChange	FileUpload, Select, Text, Textarea
onClick	Button, Document, Checkbox, Link, Radio, Reset, Submit
onDblClick	Document, Link
onDragDrop	Window
onError	Image, Window
onKeyDown	Document, Image, Link, Textarea
onKeyPress	Document, Image, Link, Textarea
onKeyUp	Document, Image, Link, Textarea
onMouseDown	Button, Document, Link
onMouseOver	Layer, Link
onLoad	Image, Layer, Window
onMouseOut	Layer, Link

Tabla 1.1: Algunos manejadores de eventos más utilizados en *Javascript.*

Algunas ventajas que posee este lenguaje, además de la interactividad que ofrece al cliente, es que tienden a ser pequeños y compactos, no requieren mucha memoria ni mucho tiempo para su ejecución. Además, al incluirse dentro de las mismas páginas *HTML*, se reduce el número de accesos independientes a la *red*.

1.1.4. *AJAX*

Javascript Asíncrono y *XML*, *AJAX* (*Asynchronous Javascript And XML,* por sus siglas en inglés) es una técnica de desarrollo web para crear *Aplicaciones Web* interactivas que mantienen una comunicación asíncrona con el *servidor* en segundo plano (Garret, 2005). Esta comunicación se establece con el fin de lograr cambios sobre una misma página sin necesidad de recargarla completamente, lo que significa aumentar la interactividad, velocidad y usabilidad en las *Aplicaciones Web*.

AJAX es una *técnica* que involucra varias *tecnologías,* de naturaleza *asíncrona*s, en el sentido que los datos adicionales que se requieren al *servidor* se cargan en segundo plano sin interferir con la visualización ni el comportamiento de la página. *Javascript* es el lenguaje interpretado en el que normalmente se efectúan las funciones de llamada de *AJAX* mientras que el acceso a los datos se realiza mediante *XMLHttpRequest*, objeto disponible en los navegadores actuales, que actúa como una interfaz para realizar peticiones *HTTP* y *HTTPS* a *servidores web*. En cualquier caso, no es necesario que el contenido asíncrono esté formateado en *XML.*

AJAX es una combinación de cuatro tecnologías ya existentes:

- X*HTML* (o *HTML*) y *hojas de estilos en cascada* (*CSS*) para el diseño que acompaña a la información.
- Modelo de Objeto del Documento (DOM, por sus siglas en inglés *Document Object Model*) accedido generalmente con *Javascript*, para mostrar e interactuar dinámicamente con la información presentada.
- El objeto *XMLHttpReques*t para intercambiar datos de forma asíncrona con el *servidor web.*
- *XML* es el formato usado generalmente para la transferencia de datos solicitados al servidor, aunque cualquier formato puede funcionar.

En las *Aplicaciones Web* tradicionales, las acciones del usuario en la página desencadenan una serie de llamadas al *servidor*. Una vez procesada la petición del usuario, el *servidor* devuelve una nueva página *HTML* al *navegador web* y este se encarga de mostrar el contenido. En la imagen 1.6, la imagen de la izquierda muestra el modelo tradicional de las *aplicaciones web*. La imagen de la derecha muestra el nuevo modelo propuesto por *AJAX*.

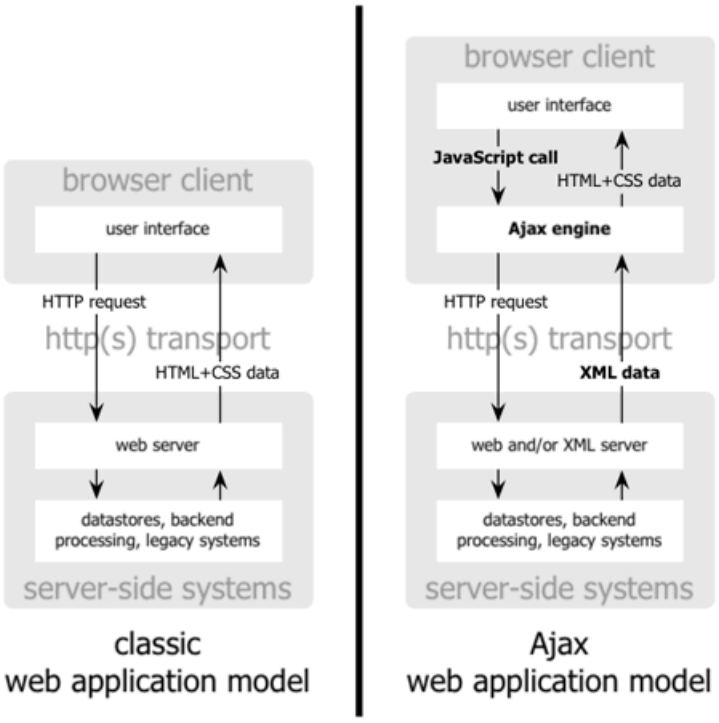


Imagen 1.6: Comparación gráfica del modelo tradicional de *aplicación web* y del nuevo modelo propuesto por *AJAX*. (http://www.librosweb.es/*AJAX*/).

Esta técnica tradicional para la implementación de las *Aplicaciones Web* funciona correctamente, pero carecen de una buena interacción con el usuario. Al realizarse peticiones continuas al *servidor*, el usuario debe esperar a que se recargue la página con los cambios solicitados. Si la aplicación debe realizar muchas peticiones seguidas, el tiempo de espera por parte del usuario puede incrementarse, degradando la usabilidad del sistema y la satisfacción del cliente, entre otras cosas.

Las aplicaciones construidas con *AJAX* eliminan la recarga constante de páginas mediante la creación de un elemento intermedio entre el usuario y el *servidor*. La nueva capa intermedia de *AJAX* mejora la respuesta de la aplicación, ya que para el usuario la comunicación con el servidor es transparente y nunca se encuentra con una ventana del navegador vacía esperando la respuesta.

En la imagen 1.7 se muestra la diferencia más importante entre una *Aplicación Web* tradicional y una *Aplicación Web* creada con *AJAX*. La imagen superior muestra la interacción síncrona propia de las *Aplicaciones Web* tradicionales, mientras que la imagen inferior muestra la comunicación asíncrona de las aplicaciones creadas con *AJAX*.

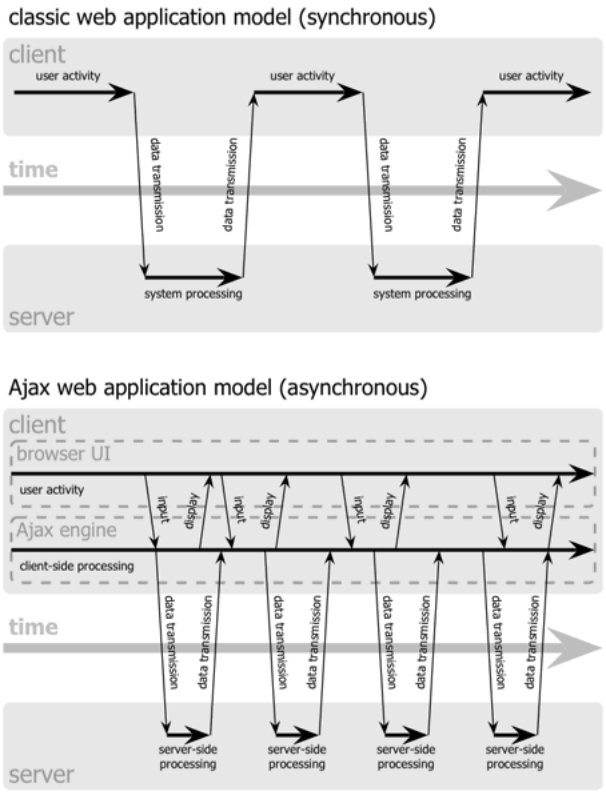


Imagen 1.7: Comparación entre las comunicaciones síncronas y asíncronas de las *aplicaciones web*.

Las *peticiones HTTP* al *servidor* se sustituyen por peticiones *Javascript* que se realizan al elemento encargado de *AJAX*. Las *peticiones* más simples no requieren intervención del *servidor*, por lo que la respuesta es inmediata. Si la interacción requiere una respuesta del servidor, la petición se realiza de forma asíncrona mediante *AJAX*. En este caso, la interacción del usuario tampoco se ve interrumpida por recargas de página o largas esperas por la respuesta del servidor.

Sin embargo, el uso de ésta técnica tiene diversos inconvenientes:

- Las *Aplicaciones Web* con *AJAX* son más difíciles de implementar que las páginas tradicionales.
- El botón de atrás no podrá cumplir su función correctamente debido a que los cambios *AJAX* no son registrados en el historial, por lo que se dificulta regresar a un estado anterior.
- Es posible que páginas con *AJAX* no puedan funcionar en teléfonos móviles u otros aparatos.
- *AJAX* no es compatible con todos los software para ciegos u otras discapacidades.

A pesar de los inconvenientes que acarrea el uso de esta técnica, su uso es cada vez más frecuente. Esto ocurre debido a sus principales ventajas en cuanto a rendimiento, lo que se traduce en una mejor experiencia de usuario. Alguna de ellas son:

- Ya no se refresca la página constantemente al interactuar con ella.
- El tiempo de espera para una petición se reduce. El usuario al hacer una petición (*request*) al *servidor,* no se envía toda la página.
- El tráfico al servidor se reduce.
- *AJAX* es soportado por la mayoría de los navegadores modernos.
- No requiere de ningún *plugin* adicional.

1.1.5. *JQuery*

Las funciones *Javascript* dentro de una *aplicación web* suelen usarse con mucha frecuencia. Las tareas básicas suelen repetirse una y otra vez en cada desarrollo. Por este motivo surgen las *librerías*, que son un producto que sirve como base para el desarrollo de aplicaciones y que aportan una serie de funciones o códigos para realizar tareas habituales. En otras

palabras, una *librería* es un conjunto de funciones, métodos y procedimientos que contienen rutinas listas para usarse, con el fin que los desarrolladores no deban programar estos métodos cada vez que los necesiten.

JQuery es una *librería Javascript* que permite simplificar la manera de interactuar con los documentos *HTML*, manipular el árbol *DOM*, manejar eventos, desarrollar animaciones y agregar interacción con la técnica *AJAX* a *páginas web* (Jquery, 2010). *JQuery* no solo contiene un conjunto de funciones usadas comúnmente, sino que también provee de una forma de escribir código *Javascript* de forma reducida. En la imagen 1.8 se muestra un ejemplo de como se puede lograr un mismo efecto usando *Javascript* simple y *JQuery*.

Javascript

```
var ejemplo = null; // objecto
function mover() {
  ejemplo.style.left = parseInt(
     ejemplo.style.left)+1+'px';
  setTimeout(mover,20); //
     llamado a mover en 20ms
}
function init() {
  //obteniendo el objeto 'ejemplo
     '
  ejemplo = document.
     getElementById('ejemplo');
  //asignando la posicion inicial
      en 0px
  ejemplo.style.left = '0px';
        //empezando animacin
  mover();
}
window.onload = init;
```

JQuery

```
$("#element").animate({ left: 100
   px; });
```

Imagen 1.8: Animación en *Javascript* simple y *JQuery.*

Es importante señalar que existen otros *frameworks Javascript* que tienen el mismo propósito. Sin embargo, *JQuery* se ha convertido en el más usado, al momento de este trabajo, y cada vez es más el apoyo que recibe por parte de todos los desarrolladores de *Aplicaciones*

Web. Además cuenta con muy buena documentación lo que lo hace ser la *librería Javascript* de facto al momento de desarrollar *Aplicaciones Web*.

1.1.6. *JQuery* UI

JQuery UI es una librería de componentes para *JQuery*, desarrollada en gran parte por el mismo equipo de desarrollo de *JQuery*, que añaden un conjunto de funcionalidades para la creación de *Aplicaciones Web interactivas* (Jquery UI, 2010). Esta librería proporciona abstracciones de bajo nivel de interacción y animación, efectos avanzados y de alto nivel, además de un conjunto completo de controles de interfaz de usuario conocido como *widgets*. Cada elemento tiene un conjunto de opciones configurables y se les pueden aplicar estilos *CSS* específicos. *JQuery* UI está dividida en cuatro (4) módulos:

1. **Núcleo:** Que contiene funciones básicas para el resto de los módulos.

2. **Interacciones:** Que consisten en efectos complejos, pero que brindan una mejor experiencia de usuario, tales como *draggable* (hacer un elemento arrastrable), *resizable* (permite redimensionar un elemento), *sortable* (permite ordenar una lista de elementos), entre otros.

3. **Efectos:** Que permiten añadir transiciones animadas y facilidades para interacciones, tales como mostrar, ocultar, color, animación, entre otros.

4. ***Widgets**:* Son un conjunto completo de controles de interfaz de usuario.

Dentro de los *widgets* más utilizados se encuentran los siguientes:

- **Menú con efecto de acordeón (*Accordion*):** Consiste en hacer de un menú clásico (un menú con todas sus opciones visibles) un menú desplegable. Un menú desplegable consta de *secciones*, las cuales están compuestas de un *encabezado* y un *contenido*. Este efecto permite mostrar el *contenido* de una *sección*, ocultando el de las otras *secciones*. En la imagen 1.9 se puede ver la esencia de este efecto, el cual permite ahorrar espacio debido a que permite ver el *contenido* de una sola *sección* a la vez.

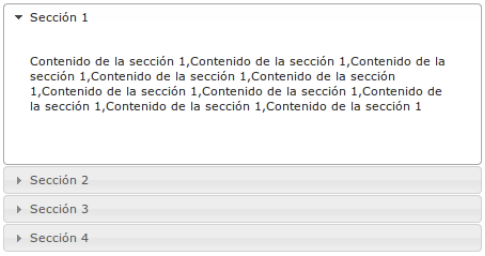


Imagen 1.9: Menú con efecto de acordeón.

- **Diálogo (*Dialog*):** Es una ventana emergente que contiene un título y un área de contenido. Una ventana de diálogo puede ser desplazada, cambiada de tamaño y cerrada. Permite mostrar un mensaje específico, incluir imágenes o inclusive, contenido interactivo. Una de sus variaciones, muy utilizada, es la ventana modal, que consiste en superponer esta ventana sobre el resto del contenido de la página, inhabilitando el uso del resto de la página con propiedades *CSS*, obligando al usuario a realizar alguna acción tales como seleccionar un valor de algún menú, rellenar un formulario, entre otros. En la imagen 1.10 se puede observar el efecto de diálogo en forma de ventana modal.

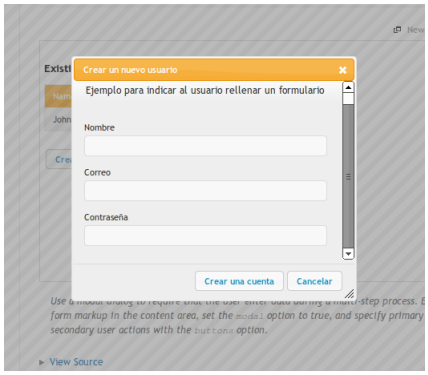


Imagen 1.10: Diálogo en forma de ventana modal.

- **Pestañas (*Tabs*):** Son utilizadas generalmente para dividir el contenido en varias *secciones* que se pueden intercambiar para ahorrar espacio, al igual que un acordeón. Solo se puede mostrar el *contenido* de una *pestaña* al mismo tiempo tal como se observa en la imagen 1.11.

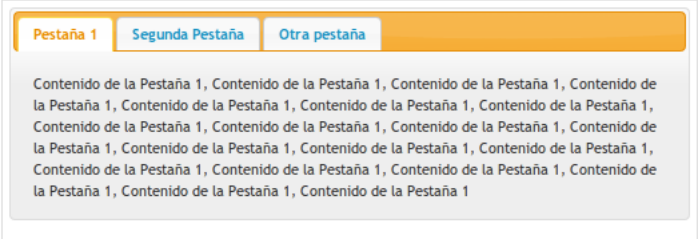


Imagen 1.11: Ejemplo del uso del efecto pestaña.

- **Calendario gráfico (*Datepicker*):** Permite mostrar un calendario para elegir una fecha en particular. Al igual que todos los *plugins* de *JQuery* es altamente configurable, permitiendo restringir ciertas fechas, colocar el calendario en base a semanas, meses, entre otros. Incluso permite mostrar varios meses al mismo tiempo (ver imagen 1.12).

Imagen 1.12: Calendario gráfico con varios meses.

- **Control Deslizante (*Slider*):** Consiste en una barra con un indicador que se puede mover para elegir un valor. Su uso puede ser tan simple como para elegir un número en determinado rango, o tan complejo como para simular un ecualizador o selector de color que permita alterar el gradiente de cada color (ver imagen 1.13).

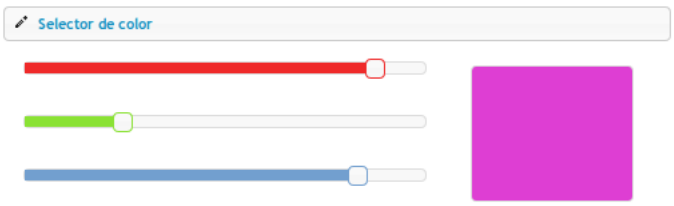


Imagen 1.13: Control deslizante para simular un selector de color.

- **Barra de progreso (*Progressbar*):** Muy utilizada para indicar el estado de distintas operaciones (ver imagen 1.14).

Imagen 1.14: Barra de progreso.

1.1.7. Otros plugins de *JQuery*

Además de *JQuery UI,* existen otros *plugins* para *JQuery* que no son desarrollados por el mismo grupo de programadores de *JQuery*. Muchos de éstos son de libre uso y están disponibles a todo el que los necesite. Algunos de los que pueden ser usados en la aplicación a desarrollar serán descritos a continuación:

- **Tabla ordenable (*Tablesorter*):** es un *plugin* de *JQuery* desarrollado por *Christian Bach* que permite que una tabla *HTML*, con etiquetas <*thead*> y <*tbody*>, sea ordenable sin la necesidad de recargar la página (Bach, 2011). *Tablesorter* permite ordenar varios tipos de datos en una celda, sin necesidad de tratamiento alguno. Además permite ordenar múltiples columnas independientemente del orden (ver imagen 1.15). Como es característico de *JQuery*, permite configurar muchas de sus funcionalidades, como lograr que una columna no sea ordenable, crear un propio criterio de ordenamiento, entre otros. Además tiene un estilo predeterminado (un *CSS* aparte), el cual le da mejor aspecto que las tablas simples de *HTML*.

Primer Nombre	Primer Apellido	Edad	Total	Descuento	Diferencia
Bruce	Almighty	45	$153.19	44.7%	+77
Bruce	Evans	22	$13.19	11%	-100.9
Bruce	Evans	22	$13.19	11%	0
Clark	Kent	18	$15.89	44%	-26
John	Hood	33	$19.99	25%	+12
Peter	Parker	28	$9.99	20.9%	+12.1

Imagen 1.15: Tabla con el efecto tabla ordenable.

- **Filtro de tabla (*Tablefilter*):** Es un *plugin* desarrollado por *Greg Weber*, que permite mostrar aquellas filas de una tabla que cumplan con el patrón de búsqueda, sin necesidad de recargar la página entera (Weber, UI Table Filter, jQuery Plugins, 2007). A medida que el patrón vaya cambiando se irán ocultando o mostrando las distintas filas de la tabla asociada. Un ejemplo de su uso se puede ver en la imagen 1.16.

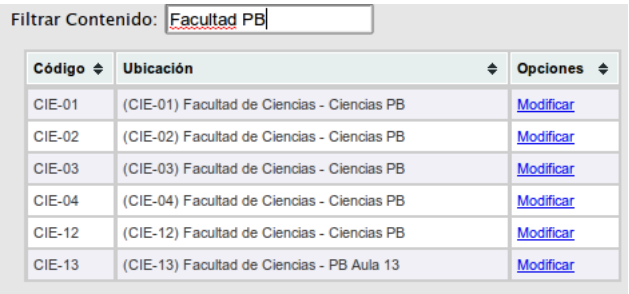


Imagen 1.16: Tabla con el efecto de filtro de tabla y tabla ordenable.

A continuación se describen las principales tecnologías del lado del servidor utilizadas en el desarrollo de *Aplicaciones Web*.

1.2. *Tecnologías del lado del servidor*

Las *tecnologías del lado del servidor* son aquellas tecnologías que son usadas para brindar soporte a los requerimientos del lado del cliente y que son implementadas directamente en el servidor en el contexto del desarrollo de una *Aplicación Web*. A continuación se describe algunas de ellas.

1.2.1. *Ruby on Rails*

Yukihiro "matz" Matsumoto, creador del lenguaje de programación *Ruby*, mezcló partes de diversos lenguajes (*Perl, Smalltalk, Eiffel, Ada, y Lisp*) para formar un nuevo lenguaje que incorporara tanto la programación funcional como la programación imperativa. *Ruby* es un lenguaje pensado para ser natural, más no simple (Ruby Programming Language, 2011).

En *Ruby*, todo es un objeto. Se le puede asignar propiedades y acciones a toda información y código. La programación orientada a objetos llama a las propiedades variables de instancia y las acciones son conocidas como métodos. La orientación a objetos pura de *Ruby* permite hacer cosas que en otro lenguaje causarían una excepción (ver imagen 1.17).

```
1 3.times{print "Ruby"}  #imprime "Ruby Ruby
    Ruby"
2 1.upto(9){|x| print x} #imprime "123456789"
3 a=[1,2,3,4]            #a es un arreglo
4 b=a.select{|x| x%2==0} #selecciona elementos
    pares, b = [2,4]
```

Imagen 1.17: Algunos comandos de *Ruby.*

En muchos lenguajes, los números y otros tipos primitivos no son objetos. El hecho que se puedan invocar algunos métodos a partir de un número es una característica que distingue a *Ruby* de algunos otros lenguajes de programación. Esta característica que ofrece, al tratar todos los elementos como objetos, permite a los programadores alterar el comportamiento de éste lenguaje libremente. Es posible agregar funcionalidades a las clases ya existentes, así como también es posible redefinirlas.

Algunas características de *Ruby:*

- **Multiplataforma**: El intérprete de *Ruby* se puede instalar en distintos sistemas operativos, incluso en sistemas operativos móviles.
- **Extensible**: Permite agregar librerías que no necesariamente deben estar escritas en el mismo lenguaje. Permite implementaciones de éstas en los lenguajes C, Java, entre otros.
- **Manejo de hilos:** Posee manejo de hilos a nivel de aplicación, es decir, de forma independiente al sistema operativo.

- **Herencia simple:** Implementa únicamente herencia simple. Sin embargo, posee mecanismos para simular la herencia múltiple.
- **Manejo de errores:** Brinda soporte para el manejo de errores de forma simple.
- **Gemas**: Las librerías en *Ruby* son denominadas gemas.

Ruby on Rails (*Rails*) es un *framework* para el desarrollo de *Aplicaciones Web* desarrollado en el lenguaje de programación *Ruby*. Un *framework* es un producto que sirve como base para el desarrollo de aplicaciones y que aportan una serie de funciones o códigos para realizar tareas habituales (Wikipedia, Framework , 2012).

Rails es un proyecto de código abierto que sigue el paradigma de la arquitectura *Modelo Vista Controlador (MVC, del inglés Model-View-Controller). MVC* es un *patrón de arquitectura* de software que separa las partes de un sistema de software en tres (3) componentes (ver imagen 1.18) que interactúan entre si (*modelo, vista y controlador*) y que cumplen distintas labores según su propósito (Gamma, 1998). El *modelo* representa los datos y todo aquello relacionado a la lógica de negocios. La *vista* representa al modelo en una interfaz de interacción con el usuario y el *controlador* responde a eventos y establece el control de la comunicación entre el *modelo* y la *vista*.

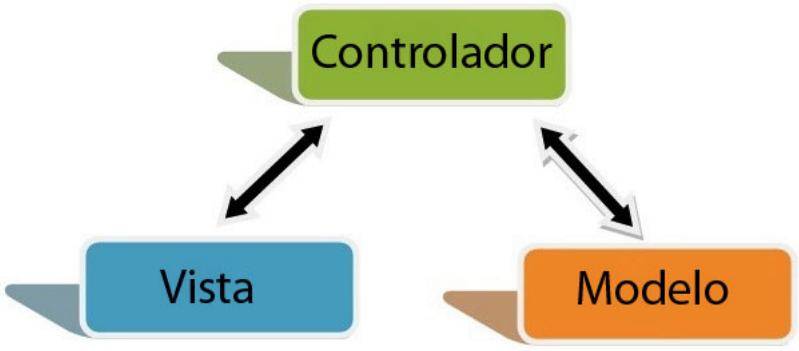


Imagen 1.18: Componentes del Modelo-Vista-Controlador

Rails trata de combinar la simplicidad con la posibilidad de desarrollar aplicaciones del mundo real escribiendo menos código en comparación con otros *frameworks* y con un mínimo de configuración (Hanson, 2011).

Este *framework* está basado en dos (2) principios fundamentales, *"No te repitas"* (del inglés *Don't repeat yourself, DRY*) y *Convención sobre Configuración* (del inglés *Convention over Configuration, CoC*). El primer principio establece que un elemento de software no debe repetirse, ya que la duplicación incrementa la dificultad en los cambios y la evolución posterior. Además, esto garantiza entre otras cosas la integridad de la aplicación, ya que lo cambios en una funcionalidad de software se realizarán en un único lugar.

El segundo principio se basa en un conjunto de convenciones que están establecidas y que no requieren configurarse. De este modo, el desarrollador deberá especificar solamente aquellas configuraciones que están fuera de la convención. Cuando se establece una solución siguiendo la convención, se hace innecesario realizar aquellas tareas para las que la convención ya ha definido un comportamiento, de este modo se ahorra un tiempo importante de desarrollo.

Rails provee diversos módulos que permiten implementar una solución para el desarrollo de aplicaciones basadas principalmente en la arquitectura *MV*C. Algunos de ellos son propios de *Ruby*.

En las *Aplicaciones Web*, los modelos son principalmente clases que representan a las tablas en la base de datos. En *Rails*, las clases del modelo son gestionadas por *ActiveRecord*. *ActiveRecord* es una implementación del patrón arquitectónico *ActiveRecord* que lleva el mismo nombre y que permite mantener relación entre un objeto y una base de datos relacional. Esta librería asocia un modelo a una tabla en la base de datos y crea automáticamente los métodos de acceso y manipulación de los datos. *ActiveRecord* permite que el programador indique las relaciones existentes entre las tablas por medio del modelo y crea automáticamen-

te los métodos necesarios, esto permite, entre otras cosas, crear relaciones lógicas a nivel de aplicación.

Las *vistas* son implementadas mediante la librería *ActionView*, que se basa en un sistema para definir plantillas de presentación para mostrar la información. Las *vistas* en *Rails* suelen contener código *Ruby* embebido y son almacenadas en formato *.html.erb*. Además esta librería provee de ciertas etiquetas (denominados *helpers*) que permiten escribir ciertos elementos en *HTML* de una forma más simple. La manera de escribir código *Ruby* dentro de una vista es mediante el uso de las *etiquetas* <%%>. Para utilizar una plantilla o colocar determinado valor directamente en *HTML* es necesario agregar un *igual (=)* al momento de abrir la etiqueta de código *Ruby* (<%=%>).

El *controlador* en *Rails* es implementado por la librería *ActionController*. Esta librería es la base de las peticiones web en *Rails*. Se componen de una o más acciones o métodos que ejecutan una petición y luego muestran una *vista* que, por convención, lleva el mismo nombre del método, o redireccionan a otra acción.

La versión estable actual de *Rails,* al momento de esta investigación*,* es la 3.2.2, que incluye otras tecnologías como *JQuery*, *SASS y SCSS* para el desarrollo de las hojas de estilo*, Coffescript* para desarrollar código interpretado *Javascript*, entre otros. Entre los beneficios que ofrece *SASS* y *SCSS* se encuentra el uso de variables, reglas de anidamiento, herencia, entre otros, que permiten un desarrollo más eficiente de las *Aplicaciones Web* y con mejor calidad en los efectos *CSS*. Asimismo, *Coffescript* permite escribir código *Javascript* de manera eficiente, reduciendo el número de líneas de código e incluyendo ciertas características de otros lenguajes de programación que facilitan el desarrollo de código script.

1.2.2. *HAML*

Lenguaje de marcado de abstracción *HTML*, *HAML* (*HTML* Abstraction Markup Language, por sus siglas es inglés) es un lenguaje ligero de marcado que es usado para describir el *XHTML* de cualquier documento web sin el uso de la codificación tradicional (Catlin, HAML , 2011). *HAML* está diseñado para expresar la estructura de documentos *XHTML* o *XML* en una forma elegante, fácil y no repetitiva, usando indentación en lugar del cierre de etiquetas (</ >). Inicialmente fue creado como un *plugin* para *Rails*, pero actualmente puede ser usado con muchos *frameworks*. La extensión de los archivos *HAML* debe ser *.html.haml*.

Debido a que *HAML* utiliza espacios en blanco como forma de organizar el documento, el número de líneas en un documento web se reduce, en promedio, a la mitad ya que no se requieren las etiquetas de cierre. En la imagen 1.19 se puede observar la estructura básica de un documento *HTML* y su equivalente en *HAML*.

HTML

```
1 <!DOCTYPE HTML PUBLIC "-//W3C//
    DTD HTML 4.01//EN" "http://
    www.w3.org/TR/html4/loose.dtd
    " >
2 <html lang="es">
3   <head>
4     <title> EJEMPLO DE TITULO </
        title>
5   </head>
6   <body>
7     <p> EJEMPLO DE UN PARRAFO </p
        >
8   </body>
9 </html>
```

HAML

```
1 !!!
2 %html{:lang => "es"}
3   %head
4     %title EJEMPLO DE TITULO
5   %body
6     %p EJEMPLO DE UN PARRAFO
```

Imagen 1.19: Estructura básica de un documento HTML y su versión en HAML.

HAML, en su uso con *Rails*, permite incluir código *Ruby* de una forma más simple que trata de reducir la cantidad de código escrito. Mientras que en la forma tradicional se debe incluir código *Ruby* entre *etiquetas* <%%>, en *HAML* solo debe colocarse un *guión (-)*.

Asimismo, para ingresar valores en variables *Ruby* solo debe colocarse un símbolo de igual (=). En la imagen 1.20 se muestra un formulario simple en código *ERB* y su equivalente en *HAML*.

ERB

```
<div id="formulario">
  <%= form_for @usuario do |u| %>
    <div class="izquierda">
        Usuario: </div>
    <div class="derecha"> <%= f.
        text_field :usuario %> </
        div>
    <div class="izquierda"> Clave
        : </div>
    <div class="derecha"> <%= f.
        text_field :clave %> </
        div>
    <%= f.submit %>
  <% end %>
</div>
```

HAML

```
#formulario
  =form_for @usuario do |u|
    .izquierda Usuario:
    .derecha= f.text_field :
        usuario
    .izquierda Clave:
    .derecha=f.text_field :clave
    =f.submit
```

Imagen 1.20: Formulario en código ERB y su equivalente en HAML.

La ventaja principal que involucra el uso de *HAML* es el ahorro considerable en el tiempo de desarrollo de las vistas de una aplicación. Además permite construir documentos *HTML* bien formados (aquellos que contienen todas sus etiquetas de apertura y cierre respectivamente), lo que es importante al momento de visualizar una página en un *navegador web,* ya que los navegadores actuales suelen mostrar documentos *HTML* aún si éstos no están bien formados, pudiendo ocasionar resultados no deseados en las *vistas*.

1.2.3. *SASS y SCSS*

Las *Hojas de Estilo Sintácticamente Impresionantes (SASS*, del inglés *Syntactically Awesome Style Sheets*) fueron creadas originalmente por *Hampton Catlin* y *Nathan Weizenbaum. SASS* es un metalenguaje (lenguaje utilizado para definir otros lenguajes) para *CSS* que es usado para describir el estilo de un documento de una forma simple y estructurada con más características de las que permite *CSS* (Catlin, Sass - Syntactically Awesome Stylesheets,

2011). *SASS* proporciona una sintaxis más simple y elegante para *CSS* e implementa algunas características que son bastante útiles para el desarrollo de *hojas de estilo*.

SASS es una extensión de *CSS 3* que agrega *reglas de anidamiento*, uso de *variables*, *mixins* que son un equivalente a las funciones en cualquier lenguaje de programación, *herencia*, entre otros. La sintaxis de este lenguaje está basada en la indentación.

Las hojas de estilo en cascada con estilo o elegantes (*SCSS,* del inglés *Sassy Cascading Style Sheets),* no son más que la forma actual de escribir documentos *SASS* con otra sintaxis igual a la de *CSS 3*. Esto quiere decir que emplean el uso de *llaves ({ })* y *punto y coma* (;) para determinar los bloques de código. Por lo tanto, cualquier documento escrito en *CSS 3* será válido para *SCSS.*

La forma indentada, a pesar de ser la forma anterior de escribir documentos *SASS,* sigue teniendo validez debido a la cantidad de desarrolladores que prefieren usar la indentación como forma de escribir un documento. Estos archivos deben tener la extensión .sass, mientras que de la nueva manera deben llevar la extensión .scss.

Las principales características que ofrecen estas nuevas *hojas de estilo* son las siguientes:

- **Uso de *variables*:** Permite almacenar valores en variables que pueden ser usadas en distintas partes del código. Además permite el uso de operaciones matemáticas con las mismas. En la imagen 1.21 se puede observar la diferencia entre el uso de variables y la forma tradicional. Esta característica es útil al momento de usar un mismo color, borde, ancho, etc, en diversas partes del código de las hojas de estilo.
- **Reglas de anidamiento:** *SASS* (o *SCSS*) permite anidar selectores de *CSS* para evitar su repetición en otro bloque de código. En la imagen 1.22 se muestra gráficamente un ejemplo de esta característica. Sass evita la repetición de los selectores de anidación dentro de otro.

SASS

```
$azul: #3bbfce
$margen: 16px
.contenido
  border-color: $azul
.borde
  padding: $margen / 2
```

SCSS

```
$azul: #3bbfce;
$margen: 16px;
.contenido {border-color: $azul;}
.borde {padding: $margen / 2;}
```

CSS 3

```
.contenido {border-color: #3bbfce
    ;}
.borde {padding: 8px;}
```

Imagen 1.21: Uso de variables en *SASS* y *SCSS* y su equivalencia en *CSS*.

SASS

```
table.hl
  margin: 2em 0
  td.ln
    text-align: right
li
  font:
    family: serif
```

SCSS

```
table.hl {
  margin: 2em 0;
  td.ln {
    text-align: right;}
}
li {
  font: {
    family: serif;}
}
```

CSS

```
table.hl {
  margin: 2em 0;
}
table.hl td.ln {
  text-align: right;
}
li {
  font-family: serif;
}
```

Imagen 1.22: Uso de las *reglas de anidamiento* en *SASS* y *SCSS* y su equivalencia en *CSS*.

- ***Mixins:*** Aún más útil que las variables, los *mixins* permiten volver a utilizar bloques enteros de *código CSS*, propiedades o selectores. Al igual que una función en cualquier

lenguaje de programación, los *mixins* permiten el pase de parámetros. En la imagen 1.23 se muestra el uso éstos.

SASS

```
@mixin tabla
  th
    text-align: center
    font-weight: bold
  td, th
    padding: 2px

@mixin izquierda($distancia)
  float: left
  margin-left: $distancia

#data
  @include izquierda(10px)
  @include tabla
```

SCSS

```
@mixin tabla {
  th {
    text-align: center;
    font-weight: bold;
  }
  td, th {padding: 2px}
}

@mixin izquierda($distancia) {
  float: left;
  margin-left: $distancia;
}

#data {
  @include izquierda(10px);
  @include tabla;
}
```

CSS

```
#data {
  float: left;
  margin-left: 10px;
}
#data th {
  text-align: center;
  font-weight: bold;
}
#data td, #data th {
  padding: 2px;
}
```

Imagen 1.23: Uso de *mixins* en *SASS* y *SCSS* y su equivalencia en *CSS*.

- **Herencia:** *SASS* (o *SCSS*) permite heredar todos los estilos de un *selector* sin necesidad de duplicar todos los estilos. En la imagen 1.24 se puede observar el beneficio de usar esta propiedad.

SASS

```
.error
  border: 1px #f00
  background: #fdd

.error.intrusion
  font-size: 1.3em
  font-weight: bold

.badError
  @extend .error
  border-width: 3px
```

SCSS

```
.error {
  border: 1px #f00;
  background: #fdd;
}
.error.intrusion {
  font-size: 1.3em;
  font-weight: bold;
}

.badError {
  @extend .error;
  border-width: 3px;
}
```

CSS

```
.error, .badError {
  border: 1px #f00;
  background: #fdd;
}

.error.intrusion,
.badError.intrusion {
  font-size: 1.3em;
  font-weight: bold;
}

.badError {
  border-width: 3px;
}
```

Imagen 1.24: Uso de *herencia* en *SASS* y *SCSS* y su equivalencia en *CSS.*

1.2.4. Coffeescript

Coffeescript es un lenguaje de programación que se compila en *Javascript*, creado por *Jeremy Ashkenas* (Burnham, 2011). La sintaxis de este lenguaje está basada en lenguajes como *Ruby* o *Python* y trata de tomar lo mejor de *Javascript* y llevarlo a una forma simple de escribir. No solo elimina el uso de *llaves ({ }),* debido a que es indentado, sino que incluye características de comprensión de arreglos y reconocimiento de patrones.

Coffeescript tiene una correspondencia uno a uno con *Javascript*, esto quiere decir que no hay degradación del rendimiento. En algunos casos, muchos de los códigos escritos en *Javascript* terminan siendo más eficientes luego de ser migrados a *Coffeescript,* debido a ciertas optimizaciones que el *compilador* puede realizar.

Algunas de las mejoras básicas que incluye este lenguaje se mencionan a continuación:

- **Funciones:** Son definidas por una lista de parámetros entre paréntesis (de forma opcional), una *flecha (->)* y el cuerpo de la función. En la imagen 1.25 se observa una función escrita en *Coffeescript* y su equivalente en *Javascript.*

Coffeescript

```
1 cuadrado = (x) -> x * x
2 cubo   = (x) -> cuadrado(x) * x
```

Javascript

```
1 var cube, square;
2 square = function(x) {
3   return x * x;
4 };
5 cube = function(x) {
6   return square(x) *
```

Imagen 1.25: Función escrita en *Coffeescript* y su equivalente en *Javascript.*

- **Sentencia *switch/when/else*:** Las sentencias switch en *Javascript* son un poco incómodas, debido a que se debe colocar la instrucción de quiebre (*break*) al final de cada *case* para evitar caer en el caso por defecto. *Coffeescript* evita esto convirtiendo cada *switch* en una expresión *retornable/asignable.* En la imagen 1.26 se refleja esta mejora.

Coffeescript

```
switch dia
  when "Ju" then ir pesca
  when "Vi", "Sa"
    if dia is dia_de_bingo
      ir bingo
      ir discoteca
  when "Do" then ir iglesia
  else ir trabajo
```

Javascript

```
switch (dia) {
  case "Ju":
    ir(pesca);
    break;
  case "Vi":
  case "Sa":
    if (day === dia_de_bingo) {
      ir(bingo);
      ir(discoteca);
    }
    break;
  case "Do":
    ir(iglesia);
    break;
  default:
    ir(trabajo);
}
```

Imagen 1.26: Sentencia *switch* en *Coffeescript* y su equivalente en *Javascript*.

- **Comparaciones encadenadas (*chained comparisons*):** Hace más natural la comparación de un valor dentro de un rango, permitiendo colocar el valor a comparar entre los valores con los que será comparado (ver imagen 1.27).

Coffeescript

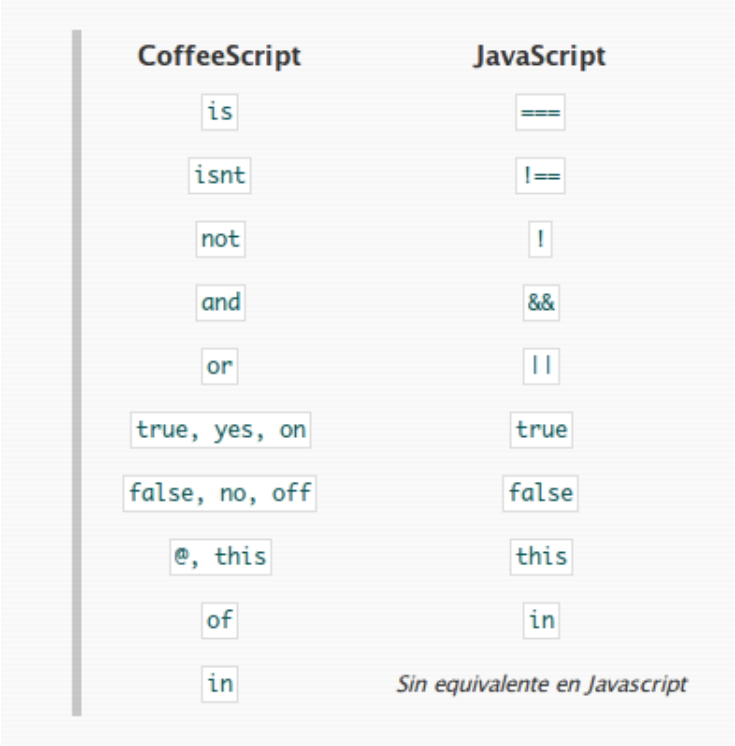
```
nota = 15
nota_valida = 20 >= cholesterol
   >= 0
```

Javascript

```
var nota, nota_valida;
nota = 15;
nota_valida = (20 >= nota && nota
    >= 0);
```

Imagen 1.27: Ejemplo de una *comparación encadenada* escrita en *Coffeescript* y su equivalente en *Javascript.*

- **Operadores y alias:** *Coffeescript* permite el uso de alias en algunos operadores. Por ejemplo, el operador *&&* de *Javascript* puede ser usado por el alias *and* en *Coffeescript*. En la imagen 1.28 se muestran algunos de los más utilizados.



CoffeeScript	JavaScript
`is`	`===`
`isnt`	`!==`
`not`	`!`
`and`	`&&`
`or`	`||`
`true, yes, on`	`true`
`false, no, off`	`false`
`@, this`	`this`
`of`	`in`
`in`	*Sin equivalente en Javascript*

Imagen 1.28: Operadores y alias más utilizados.

Éstas son apenas cuatro (4) de las muchas mejoras que involucra el uso de *Coffeescript*. En la mayoría de los casos, se escribe aproximadamente un tercio del código, en cuanto a líneas se refiere, en comparación con las líneas de código escritas usando *Javascript.* Esto permite obtener mejores resultados durante el desarrollo de una *aplicación web*.

1.3. Tecnologías Relacionadas

En esta sección se describirán algunas tecnologías que no son propias de una *Aplicación Web.* Sin embargo, todas ellas son utilizadas con frecuencia durante el desarrollo de las mismas y ayudan a obtener mejores resultados durante su elaboración. Estas tecnologías permiten resolver problemas de persistencia, manejo de versiones, manejo de archivos, entre otros. A continuación se describen algunas de ellas:

1.3.1. *RVM*

El *manejador de versiones de Ruby* (*RVM*, *Ruby Version Manager*, por sus siglas en inglés), es un herramienta multiplataforma que permite instalar, gestionar y trabajar con múltiples ambientes de *Ruby* en un mismo sistema (Seguin, 2011).

RVM permite crear proyectos con entornos diferentes totalmente independientes. Además permite crear directorios de gemas (*gemset*) para cada versión de *Ruby* instalada. Esto permite incluso tener proyectos con la misma versión de *Ruby* pero con diferentes versiones de *gemas* sin ningún inconveniente, lo que conlleva a instalar solo las *gemas* necesarias por aplicación. Además, cada *gemset* es totalmente independiente, lo que permite usar distintas versiones de *Ruby* con un mismo *gemset* sin problema alguno.

Es importante indicarle al manejador de versiones de *Ruby*, cada vez que se cambie de proyecto, cual versión de Ruby se va a utilizar y con cual *gemset*. Es común que este paso se omita y se cometan errores. Sin embargo, *RVM* permite crear *flujos de trabajo*, los cuales se encargan de establecer todo esto de manera automática.

RVM facilita muchas de las cosas con las que un desarrollador se encuentra, como probar distintas versiones de una *gema* sin alterar el ambiente de trabajo, probar distintas versiones de *Ruby* en un mismo sistema, probar la compatibilidad entre versiones de *gemas* y versiones de *Ruby* sin dañar el ambiente de trabajo, entre otros. Es importante señalar que con el uso de esta herramienta, las gemas son instaladas en un repositorio específico y no directamente en el sistema, lo que lo hace mas flexible en cuanto a roles de usuario, esto quiere decir que cualquier usuario sin permisos de administrador puede instalar, eliminar o actualizar tanto *gemas* como versiones de *Ruby* sin ningún problema (salvo por aquellas gemas que deben ser compiladas e instaladas directamente en el sistema).

En la imagen 1.29 se muestra un listado de los principales comandos provistos por RVM ejecutados en una terminal:

```
$ rvm list #muestra las versiones de ruby instaldas, indicando cual se esta usando.
$ rvm install 1.8.7 #Instala la version 1.8.7 de ruby
$ rvm use system #Selecciona la version de ruby del sistema
$ rvm use 1.8.7 #Selecciona la version de ruby 1.8.7
$ rvm gemset create ejemplo #Crea un gemset llamado ejemplo
$ rvm gemset list #muestra los gemsets creados, indicando cual se esta usando.
$ rvm use 1.8.7@ejemplo #Selecciona la version 1.8.7 de ruby con el gemset ejemplo
$ rvm use --default 1.8.7@ejemplo #Indica usar la version 1.8.7 de ruby con el gemset ejemplo de forma automatica cada vez que se inicie el sistema.
```

Imagen 1.29: Principales comandos de *RVM*.

1.3.2. *MySQL*

MySQL es un sistema de administración de bases de datos relacional que almacena y distribuye una gran cantidad de datos, típicos de una aplicación (Oracle Corporation, 2012). Está basado en la arquitectura cliente-servidor, por lo que el servidor de base de datos puede estar asociado a múltiples clientes. Utiliza el lenguaje de consulta estructurado (SQL, del ingles Structured Query Language) para el acceso y manipulación de los datos.

Algunas características de *MySQL* son las siguientes:

- **Velocidad de acceso:** Debido a que está basado en una arquitectura multi-hilos, por lo que diversos clientes pueden acceder simultáneamente de forma concurrente.

- **Velocidad de respuestas:** *MySQL* implementa un almacenamiento parcial de los resultados de las consultas comunes (caché), por lo que una vez que se ejecuta una consulta, su resultado es almacenado de forma temporal, por si se requieren esos datos nuevamente, aumentando la velocidad de respuesta.

- **Fácil uso:** provee una interfaz de línea de comandos. Además cuenta con muchas herramientas gráficas para su manejo, desde aplicaciones de escritorio hasta aplicaciones web.

- **Portabilidad:** Puede ejecutarse en diversos sistemas operativos sin ningún problema.

- **Internacionalización:** Brinda soporte para varios idiomas y diversos juegos de caracteres.

- **Código abierto:** Cualquier desarrollador tiene acceso al código libremente.

Algunas limitaciones de *MySQL* son las siguientes:

- Para algunas transacciones, según el tipo de tablas que se esté utilizando, al momento de consultar o modificar la información de la base de datos, toda la tabla donde se encuentra la información es bloqueada.

- MySQL no puede llevar a cabo copias de seguridad "en caliente" para todas las tablas que maneja. Las copias de seguridad "en caliente" son copias de la información que se realizan durante una operación sin bloquear tablas.

- Muchos manejadores de bases de datos permiten definir tipos de datos por el usuario. MySQL no lo permite.

A continuación se describe la *Fundación Escuela de Idiomas Modernos* y el proceso para la realización de los exámenes escritos dentro de la organización.

Capítulo 2 Diseño de las evaluaciones escritas de *FUNDEIM*

La *Fundación Escuela de Idiomas Modernos (FUNDEIM)* fue concebida como una herramienta de apoyo y promoción para todas las actividades académicas y profesionales que desarrolla la *Escuela de Idiomas Modernos de la Facultad de Humanidades y Educación de la Universidad Central de Venezuela*. La fundación se encarga de instrumentar proyectos para la generación de recursos financieros necesarios en defensa y promoción de las actividades y objetivos que se propone alcanzar la *Escuela de Idiomas Modernos* (FUMDEIM, 2005).

Dentro de las actividades académicas que realiza *FUNDEIM* se encuentra la impartición de cursos de diversos idiomas, entre ellos: alemán, francés, italiano, portugués e inglés, siendo este último el más demandado. Cada curso tiene una duración aproximada de tres (3) meses y son ofertados a estudiantes de la *Universidad Central de Venezuela* y al público en general. Actualmente se dictan cursos en cuatro periodos cada año.

2.1. Organización de los cursos

Los cursos impartidos en *FUNDEIM* son ofrecidos a niños, adolescentes y adultos en el caso de inglés, y solo para adultos en el caso de los demás idiomas. Adicionalmente constan de nueve (9) niveles y una única categoría (adultos), salvo el idioma inglés que consta de tres (3) categorías: niños, adolescentes (teens) y adultos; cada uno con sus distintas cantidades de niveles, tal como se ve en la imagen 2.1.

Aproximadamente, se dictan cincuenta y cinco (55) cursos por periodo, con sus distintos horarios y secciones. Al año se abren cuatro (4) periodos, lo que implica una impartición de al menos doscientos veinte (220) cursos anualmente.

Para la aprobación de un curso debe obtenerse una nota mínima de diez (10) puntos en el caso de los niveles niños y teens; y una nota mínima de quince (15) en el nivel adultos.

CATEGORÍAS

NIVELES	Niños	Adolescentes (Teens)	Adultos
	Beginners	Básico I	Básico I
	Elementary	Básico II	Básico II
	Intermediate	Conversación Básico	Básico III
	Advanced	Intermedio I	Conversación Básico
	---	Intermedio II	Intermedio I
	---	Conversación Intermedio	Intermedio II
	---	---	Intermedio III
	---	---	Conversación Intermedio
	---	---	Avanzado I
	---	---	Avanzado II
	---	---	Avanzado III
	---	---	Conversación Avanzado

(a) Niveles y categorías para el idioma inglés.

CATEGORÍA

NIVELES	Adultos
	Básico I
	Básico II
	Básico III
	Conversación Básico
	Intermedio I
	Intermedio II
	Intermedio III
	Conversación Intermedio
	Avanzado I
	Avanzado II
	Avanzado III
	Conversación Avanzado

(b) Niveles y categorías para los idiomas alemán,francés, italiano y portugués.

Imagen 2.1: Niveles y categorías para los diferentes idiomas.

Los cursos son dictados de lunes a jueves en horas de la tarde y los días sábados en la mañana. La distribución de los días y los horarios se muestra a continuación:

- Lunes y Miércoles de 4:00 pm a 6:15 pm.
- Lunes y Miércoles de 6:15 pm a 8:30 pm.
- Martes y Jueves de 4:00 pm a 6:15 pm.
- Martes y Jueves de 6:15 pm a 8:30 pm.
- Sábados de 8:30 am a 1:15 pm.

Además, no todos los cursos están disponibles en todos los horarios. Esto es debido a ciertas reglas de la fundación, por lo que los horarios quedan establecidos de la siguiente manera:

- Los cursos de inglés en las categorías niños y teens, así como los idiomas alemán y portugués son dictados sólo los días sábados.

- Los cursos de italiano son dictados en el horario de martes y jueves, así como también los días sábados.

- Los cursos de francés son dictados en el horario de lunes y miércoles, así como también los días sábados.

- Los cursos de inglés en la categoría adultos se dictan en todos los horarios.

Este trabajo se centra en la propuesta de una solución a la problemática que acarrea la realización de los exámenes escritos de los distintos niveles de inglés, en la categoría adultos. Para ello, es necesario explicar los distintos sistemas para la evaluación de un idioma, entre ellos están: Pruebas diagnósticas, de nivelación, de progreso, acumulativas y de aptitud. A continuación se describe cada una de ellas.

2.2. Sistemas de evaluación

Actualmente, existen diferentes tipos de exámenes para evaluar los conocimientos en un idioma. Entre ellos se encuentran (Spratt, M., Alan, P., & Williams, M., 2005):

- **Exámenes diagnósticos**: Son utilizados para medir el conocimiento general sobre un idioma y ayudan a definir en que áreas el aspirante requiere de mayor ayuda. Este tipo de evaluación es realizado, generalmente, al momento de comenzar algún curso y los resultados sirven para determinar que objetivos deben ser tomados en cuenta con mayor dedicación.

- **Pruebas de nivelación:** Son tomados por los aspirantes con la finalidad de determinar el nivel al que deben ingresar en algún curso, según los conocimientos previos en el idioma.

- **Pruebas de progreso:** Son aquellas destinadas a evaluar una parte de la enseñanza en un curso, con el objetivo de determinar que tanto han aprendido los estudiantes.

- **Pruebas acumulativas:** Son aquellas que miden el conocimiento luego de realización de un curso. Evalúan todo el contenido desarrollado y, en algunos casos, ofrecen una retroalimentación del desempeño del tomador del curso.

- **Pruebas de aptitud:** Son aquellas que miden los conocimientos generales que tiene una persona en algún idioma. Los contenidos a evaluar en estas pruebas no se basan en un curso o en un plan de evaluación.

Actualmente, en *FUNDEIM* se realizan pruebas de nivelación y pruebas de progreso. Además, existen diversas instituciones que cuentan con aplicaciones para el desarrollo de exámenes escritos, una de ellas es el *Servicio de Pruebas Educativas* (*ETS*, del ingles *Educational Testing Service*) con el *Examen de Inglés como Lengua Extranjera (TOEFL,* del inglés *Test of English as a Foreign Language)*. Esta organización busca promover la calidad y equidad en la educación, ofreciendo evaluaciones de calidad, con el fin de promover el aprendizaje y el rendimiento educativo para el desarrollo profesional de todas las personas en el mundo. Otra institución que desarrolla exámenes escritos para evaluar el conocimiento del idioma inglés, es la *Universidad de Cambridge*, en conjunto con otras organizaciones. Esta universidad desarrolla el *Sistema Internacional de Evaluación del Idioma Inglés (*IELTS*,* del inglés *International English Language Testing System)*, con el fin de evaluar los conocimientos generales en el inglés.

Ambos exámenes toman en cuenta cuatro (4) aspectos fundamentales en la evaluación de conocimientos en un idioma:

- **Listening**: Consiste en escuchar un audio con la finalidad de realizar preguntas referentes al contenido de esa grabación.

- **Writing**: Su objetivo es evaluar la capacidad de redacción del tomador de la prueba. Generalmente, se procede a escuchar una grabación o leer algunos párrafos para luego

realizar un resumen sobre ellos.

- **Reading**: Tiene como finalidad determinar las capacidades en la lectura del tomador de la prueba.

- **Speaking**: En este tipo de prueba, los aspirantes deben leer pequeñas lecturas o escuchar algunas grabaciones y, luego de esto, deben discutir acerca de los temas puntuados en las grabaciones o lecturas.

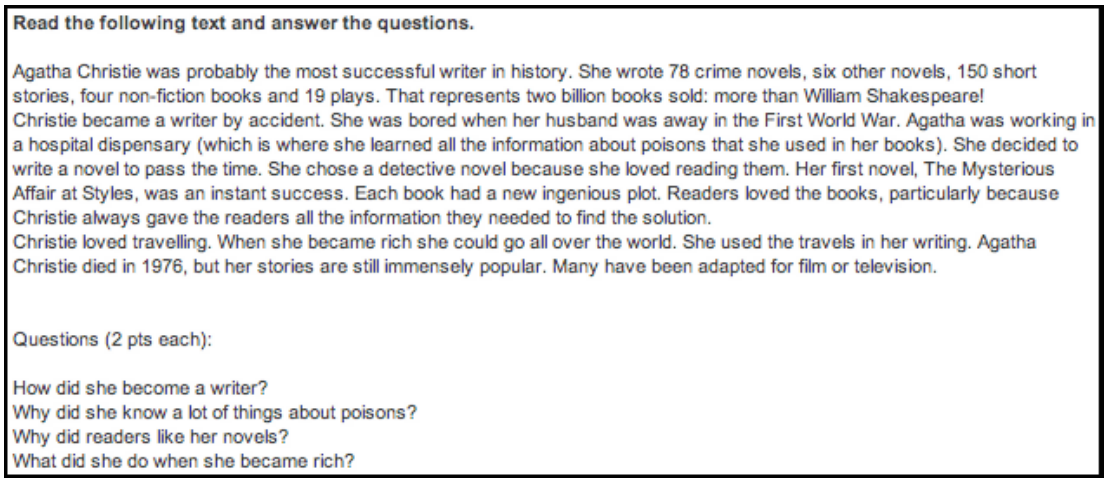
En ambos sistemas de evaluación, los resultados son otorgados en una escala diferente y cada organización establece el mínimo requerido para tomar como aprobado el examen.

2.2.1. Exámenes de *FUNDEIM*

Dentro de los cursos que imparte *FUNDEIM*, cada nivel del idioma inglés cuenta con un plan de evaluación denominado *Syllabus*, el cual contiene toda la planificación del contenido que será dictado y evaluado.

Todos los niveles, exceptuando los de conversación, dentro del idioma inglés en la categoría adultos, cuentan con dos (2) exámenes escritos, los cuales se realizan a mediados y al final del curso. La estructura general de la primera evaluación escrita consta de cuatro partes: Comprensión lectora, gramática, vocabulario y redacción, que se detallan a continuación.

- **Comprensión lectora:** Este tipo de preguntas consisten en una lectura breve sobre un tópico específico. Posterior a esto, se deben contestar una serie de preguntas relacionadas con el texto (ver imagen 2.2). Las respuestas pueden estar explícitas en la lectura o de manera que se deba inferir la respuesta a través de un análisis.



Read the following text and answer the questions.

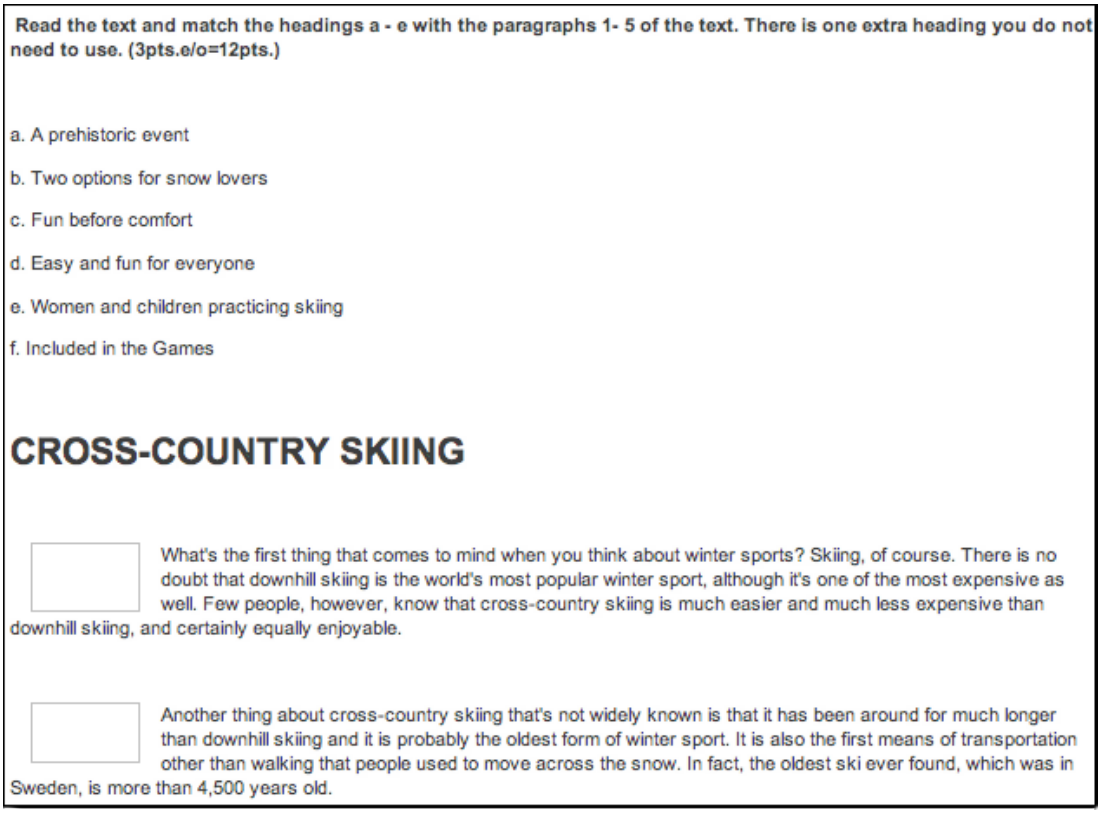
Agatha Christie was probably the most successful writer in history. She wrote 78 crime novels, six other novels, 150 short stories, four non-fiction books and 19 plays. That represents two billion books sold: more than William Shakespeare!
Christie became a writer by accident. She was bored when her husband was away in the First World War. Agatha was working in a hospital dispensary (which is where she learned all the information about poisons that she used in her books). She decided to write a novel to pass the time. She chose a detective novel because she loved reading them. Her first novel, The Mysterious Affair at Styles, was an instant success. Each book had a new ingenious plot. Readers loved the books, particularly because Christie always gave the readers all the information they needed to find the solution.
Christie loved travelling. When she became rich she could go all over the world. She used the travels in her writing. Agatha Christie died in 1976, but her stories are still immensely popular. Many have been adapted for film or television.

Questions (2 pts each):

How did she become a writer?
Why did she know a lot of things about poisons?
Why did readers like her novels?
What did she do when she became rich?

Imagen 2.2: Ejemplo de comprensión lectora para responder preguntas.

También pueden consistir en la lectura de párrafos para asociarlos con un título en particular, a fin de establecer una relación entre dicho título y el párrafo (ver imagen2.3).



Read the text and match the headings a - e with the paragraphs 1- 5 of the text. There is one extra heading you do not need to use. (3pts.e/o=12pts.)

a. A prehistoric event

b. Two options for snow lovers

c. Fun before comfort

d. Easy and fun for everyone

e. Women and children practicing skiing

f. Included in the Games

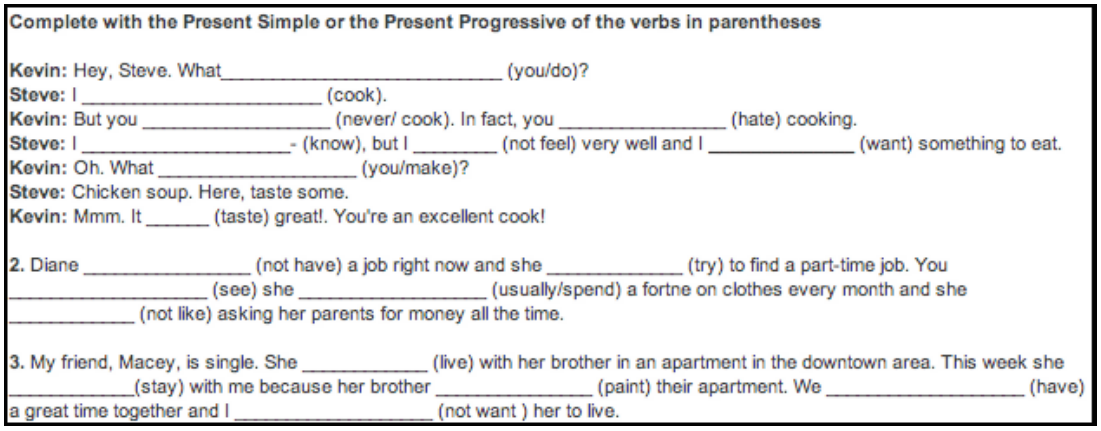
CROSS-COUNTRY SKIING

What's the first thing that comes to mind when you think about winter sports? Skiing, of course. There is no doubt that downhill skiing is the world's most popular winter sport, although it's one of the most expensive as well. Few people, however, know that cross-country skiing is much easier and much less expensive than downhill skiing, and certainly equally enjoyable.

Another thing about cross-country skiing that's not widely known is that it has been around for much longer than downhill skiing and it is probably the oldest form of winter sport. It is also the first means of transportation other than walking that people used to move across the snow. In fact, the oldest ski ever found, which was in Sweden, is more than 4,500 years old.

Imagen 2.3: Ejemplo de comprensión lectora para asociar títulos a párrafos.

- **Gramática:** En este tipo de preguntas, el estudiante debe completar en alguna forma verbal un conjunto de oraciones (ver imagen 2.4).



Complete with the Present Simple or the Present Progressive of the verbs in parentheses

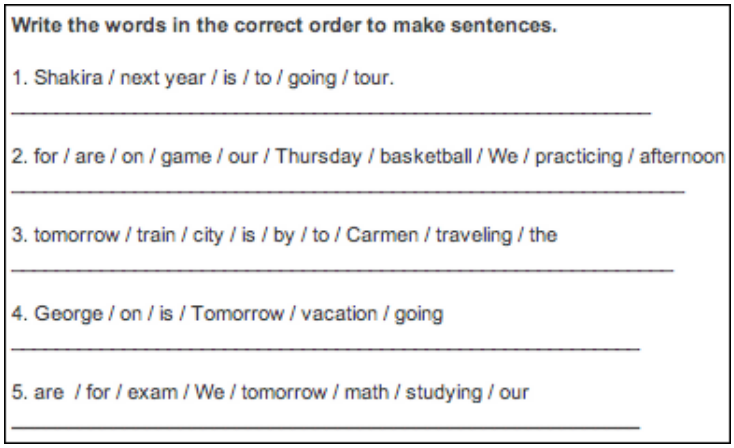
Kevin: Hey, Steve. What_____ (you/do)?
Steve: I _____ (cook).
Kevin: But you _____ (never/ cook). In fact, you _____ (hate) cooking.
Steve: I _____- (know), but I _____ (not feel) very well and I _____ (want) something to eat.
Kevin: Oh. What _____ (you/make)?
Steve: Chicken soup. Here, taste some.
Kevin: Mmm. It _____ (taste) great!. You're an excellent cook!

2. Diane _____ (not have) a job right now and she _____ (try) to find a part-time job. You _____ (see) she _____ (usually/spend) a fortne on clothes every month and she _____ (not like) asking her parents for money all the time.

3. My friend, Macey, is single. She _____ (live) with her brother in an apartment in the downtown area. This week she _____(stay) with me because her brother _____ (paint) their apartment. We _____ (have) a great time together and I _____ (not want) her to live.

Imagen 2.4: Ejemplo de una pregunta de gramática, de completación de oraciones.

También existen las preguntas en donde el aspirante debe re-ordenar un conjunto de oraciones de tal forma que tengan coherencia (ver imagen 2.5).



Write the words in the correct order to make sentences.

1. Shakira / next year / is / to / going / tour.
_____

2. for / are / on / game / our / Thursday / basketball / We / practicing / afternoon
_____

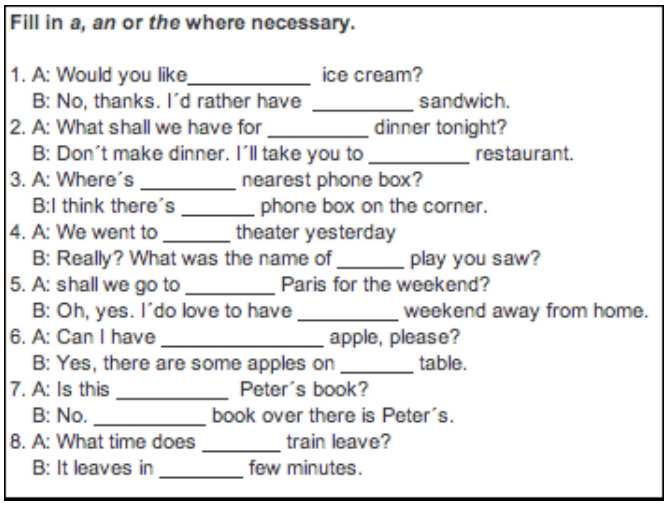
3. tomorrow / train / city / is / by / to / Carmen / traveling / the
_____

4. George / on / is / Tomorrow / vacation / going
_____

5. are / for / exam / We / tomorrow / math / studying / our
_____

Imagen 2.5: Ejemplo de una pregunta de gramática, re-ordenando oraciones.

Igualmente, están aquellas en las que se debe colocar una palabra específica de una lista dada, con el objetivo que la oración esté escrita correctamente (ver imagen 2.6).

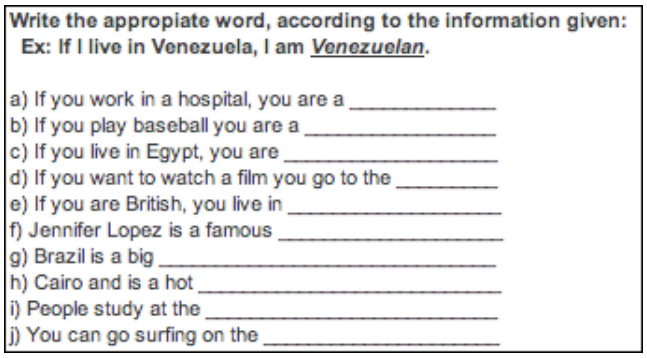


Fill in *a, an* or *the* where necessary.

1. A: Would you like_____ ice cream?
   B: No, thanks. I´d rather have _____ sandwich.
2. A: What shall we have for _____ dinner tonight?
   B: Don´t make dinner. I´ll take you to _____ restaurant.
3. A: Where´s _____ nearest phone box?
   B:I think there´s _____ phone box on the corner.
4. A: We went to _____ theater yesterday
   B: Really? What was the name of _____ play you saw?
5. A: shall we go to _____ Paris for the weekend?
   B: Oh, yes. I´do love to have _____ weekend away from home.
6. A: Can I have _____ apple, please?
   B: Yes, there are some apples on _____ table.
7. A: Is this _____ Peter´s book?
   B: No. _____ book over there is Peter´s.
8. A: What time does _____ train leave?
   B: It leaves in _____ few minutes.

Imagen 2.6: Ejemplo de una pregunta de gramática, completando oraciones.

- **Vocabulario:** En estas preguntas, el estudiante debe escribir la palabra correcta de acuerdo a una información previa (ver imagen 2.7).

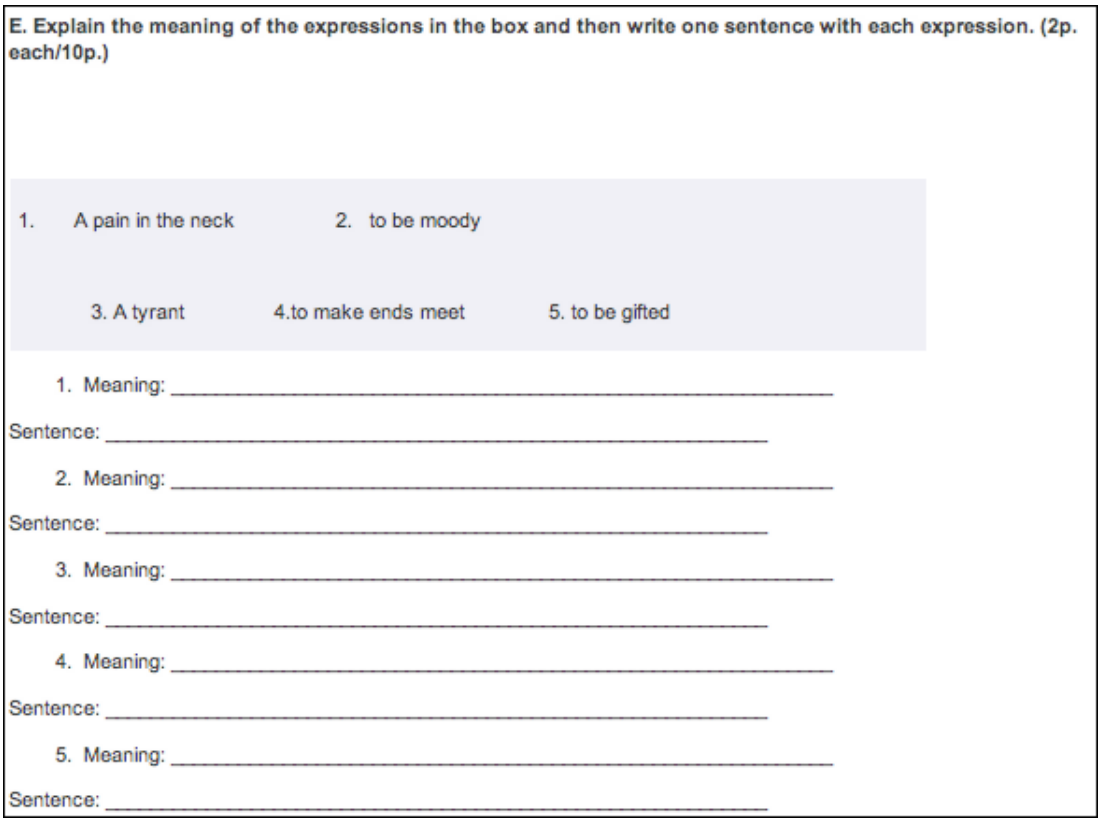


Write the appropiate word, according to the information given:
Ex: If I live in Venezuela, I am *Venezuelan*.

a) If you work in a hospital, you are a _____
b) If you play baseball you are a _____
c) If you live in Egypt, you are _____
d) If you want to watch a film you go to the _____
e) If you are British, you live in _____
f) Jennifer Lopez is a famous _____
g) Brazil is a big _____
h) Cairo and is a hot _____
i) People study at the _____
j) You can go surfing on the _____

Imagen 2.7: Ejemplo de una pregunta de vocabulario, escribiendo la palabra correcta.

También existen preguntas en donde se debe explicar el significado de una palabra o expresión y luego escribir una oración relacionada (ver imagen 2.8).



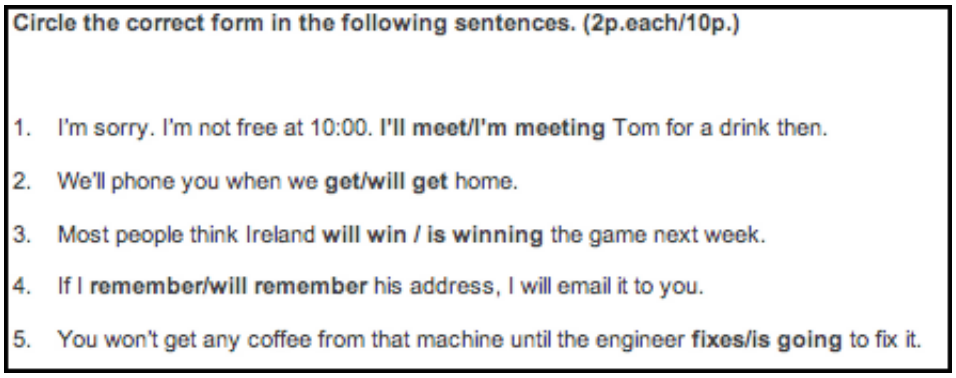
E. Explain the meaning of the expressions in the box and then write one sentence with each expression. (2p. each/10p.)

1. A pain in the neck
2. to be moody
3. A tyrant
4. to make ends meet
5. to be gifted

1. Meaning: _____
Sentence: _____
2. Meaning: _____
Sentence: _____
3. Meaning: _____
Sentence: _____
4. Meaning: _____
Sentence: _____
5. Meaning: _____
Sentence: _____

Imagen 2.8: Ejemplo de una pregunta de vocabulario, explicando expresiones y escribiendo oraciones.

Igualmente, se encuentran preguntas en donde se debe encerrar en un círculo la forma correcta de un verbo o expresión (ver imagen 2.9).

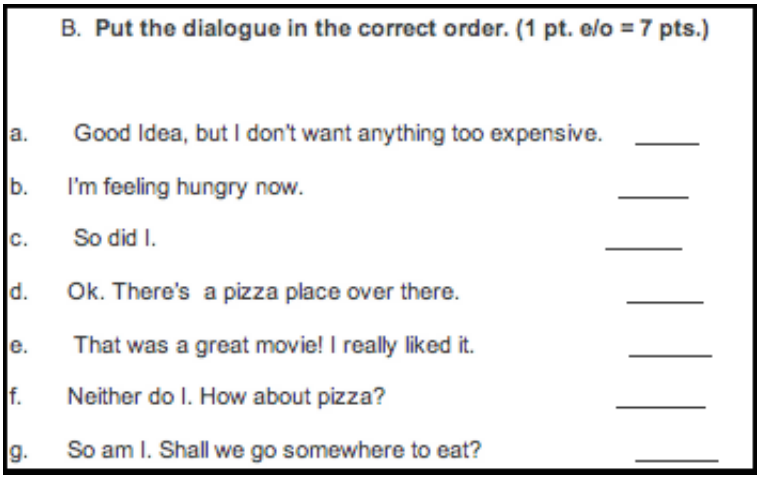


Circle the correct form in the following sentences. (2p.each/10p.)

1. I'm sorry. I'm not free at 10:00. **I'll meet/I'm meeting** Tom for a drink then.
2. We'll phone you when we **get/will get** home.
3. Most people think Ireland **will win / is winning** the game next week.
4. If I **remember/will remember** his address, I will email it to you.
5. You won't get any coffee from that machine until the engineer **fixes/is going** to fix it.

Imagen 2.9: Ejemplo de una pregunta de vocabulario, encerrando en un círculo.

También, están aquellas en las que se debe establecer un orden en ciertas oraciones con el objetivo de formar un párrafo coherente (ver imagen 2.10).



B. **Put the dialogue in the correct order. (1 pt. e/o = 7 pts.)**

a. Good Idea, but I don't want anything too expensive. _____

b. I'm feeling hungry now. _____

c. So did I. _____

d. Ok. There's a pizza place over there. _____

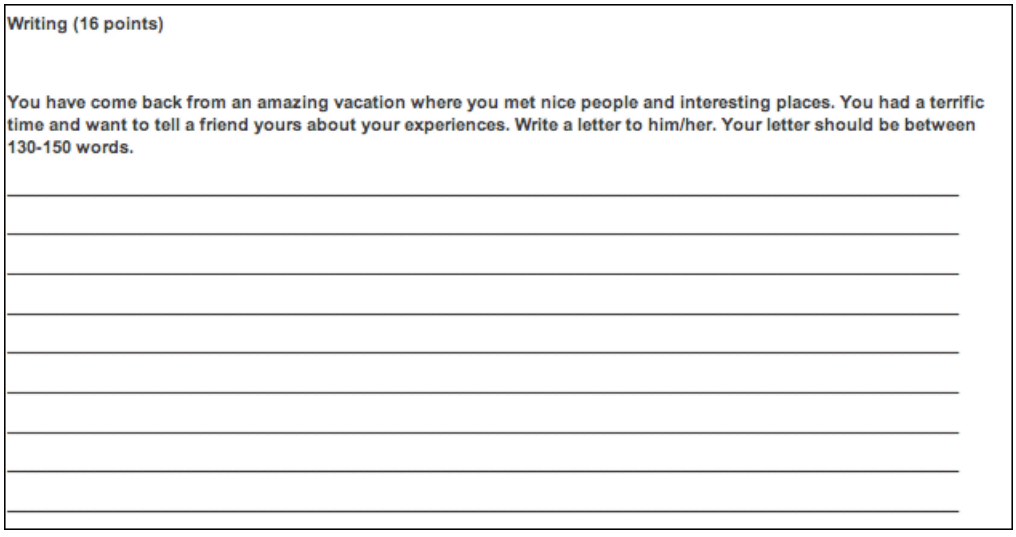
e. That was a great movie! I really liked it. _____

f. Neither do I. How about pizza? _____

g. So am I. Shall we go somewhere to eat? _____

Imagen 2.10: Ejemplo de una pregunta de vocabulario, ordenando oraciones.

- **Redacción:** En estas preguntas, el estudiante debe redactar un párrafo sobre algún tópico específico. El escrito debe tener por lo menos cien (100) palabras y, en algunas ocasiones, un máximo de ciento cincuenta (150) (ver imagen 2.11).



Writing (16 points)

You have come back from an amazing vacation where you met nice people and interesting places. You had a terrific time and want to tell a friend yours about your experiences. Write a letter to him/her. Your letter should be between 130-150 words.

Imagen 2.11: Ejemplo de pregunta de redacción.

La segunda evaluación agrega una parte de *listening* a la evaluación. Esto ocurre solo hasta el nivel Intermedio I, por lo tanto, en lo que resta de niveles se mantiene la estructura básica del primer examen.

La ponderación de cada evaluación es de treinta por ciento (30 %) para cada examen y cada uno consta de ochenta (80) puntos, repartidos de forma desigual por cada pregunta. El criterio para elegir el puntuación de cada pregunta viene dado por la dificultad que acarrea la resolución de la misma, medida en tiempo y en contenido a evaluar. La manera de escoger las preguntas se puede apreciar en la imagen 2.12.

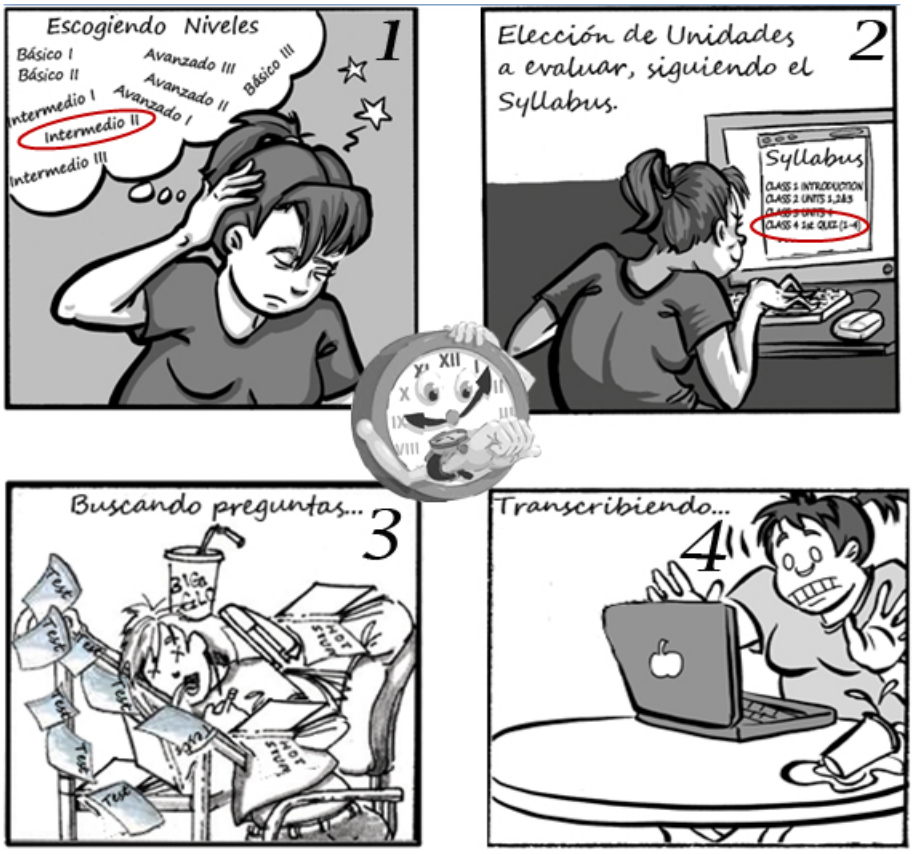


Imagen 2.12: Pasos para la realización de los exámenes escritos.

En primera instancia se debe escoger el nivel de la evaluación que se desea realizar, luego de esto se identifican las unidades a evaluar siguiendo el *Syllabus*. Posteriormente, se eligen

de un conjunto de preguntas sin ningún tipo de orden, aquellas que forman parte de la evaluación. El conjunto de preguntas a elegir es tomado de los libros *American Cutting Edge* del nivel uno (1) al cuatro (4), el cual contiene exámenes de muestra agrupados por módulos, en donde cada módulo no hace correspondencia directa con las unidades descritas en el *Syllabus*, ya que un módulo puede tener un conjunto más grande de éstas. También se toman preguntas de exámenes anteriores almacenados en formato *.doc* y de otros libros de cursos anteriores. Finalmente, se procede a transcribir las preguntas y desarrollar el examen a realizar. Dentro de los problemas identificados en la forma de diseñar estos exámenes se encuentran:

- Gran cantidad de tiempo invertido, lo que afecta el tiempo de planificación y administración de los cursos.
- Inexistencia de un repositorio único de preguntas.
- Las preguntas que se tienen, no cumplen ningún tipo de categorización, lo que dificulta la selección de las mismas para la elaboración de un examen.
- Se genera problemas en la transcripción de preguntas, que deben corregirse por cada examen.
- No existe un control de las preguntas existentes.

La realización de estas evaluaciones es un proceso largo que se realiza de forma manual por la coordinadora de los cursos. Este proceso debe hacerse dieciocho (18) veces cada curso y toma un tiempo valioso de dos (2) horas aproximadamente.

Es importante señalar que, por falta de tiempo, no es posible realizar un examen diferente para cada horario de los distintos cursos en los diferentes niveles. Sin embargo, es ideal que esto pueda ocurrir, lo que implica realizar noventa (90) exámenes escritos por periodo. Si se realizan noventa (90) exámenes por periodo, es equivalente a invertir setecientas veinte (720) horas al año aproximadamente, es decir, un mes sin horas de descanso. Además, estos exámenes son realizados por una sola persona, la coordinadora de los cursos. Es por esto que

surge el planteamiento de realizar una *Aplicación Web* que facilite este trabajo, para que el desarrollo de las evaluaciones escritas no sea un proceso que requiera de mucho tiempo ni recurso humano. A continuación se describe el objetivo general y los objetivos específicos de esta investigación.

2.3. Objetivo General

Desarrollo de una *Aplicación Web* para la elaboración automatizada del diseño de los exámenes escritos de los cursos de extensión de la Escuela de Idiomas Modernos de la Facultad de Humanidades y Educación de la Universidad Central de Venezuela, que apoye la realización de los exámenes escritos.

2.4. Objetivos específicos

Los objetivos específicos para dar cumplimiento al objetivo general son:

- Realizar el levantamiento de la información y los requerimientos de la institución.
- Seguir un método de desarrollo de software ágil.
- Diseñar una base de datos para el almacenamiento de las preguntas y exámenes del idioma inglés en la categoría adultos.
- Diseñar una interfaz usable que permita agregar preguntas banco de preguntas.
- Implementar un módulo que permita gestionar el banco de preguntas.
- Desarrollar un módulo que permita la gestión y generación de exámenes escritos y su posterior impresión.

2.5. Justificación

El desarrollo de la *Aplicación Web* para la elaboración automatizada del diseño de los exámenes escritos de los cursos de extensión de la Escuela de Idiomas Modernos de la Facultad de Humanidades y Educación de la Universidad Central de Venezuela, permite a la fundación contar con un repositorio único de preguntas que facilita la búsqueda de las mismas. Además brinda la posibilidad de administrar esas preguntas de forma rápida. También permite reducir costos en tiempo y recursos humanos, en cuanto al diseño de los exámenes escritos, y permite realizar distintos exámenes para los diferentes horarios de un curso.

A continuación se describe el desarrollo de la *Aplicación Web,* siguiendo el *método de desarrollo de software AgilUs*.

Capítulo 3 *aTesT*

En este capítulo se explica el *método de desarrollo de software*, denominado *AgilUs,* que se utiliza para la implementación de la *Aplicación Web,* que recibe por nombre *aTesT*. Además, se detalla el desarrollo del sistema haciendo seguimiento a las etapas y actividades que propone el *método de desarrollo de software*. A continuación se detalla *AgilUs.*

3.1. *AgilUs*

AgilUs es un método de desarrollo ágil que toma en cuenta la usabilidad a tempranas etapas del desarrollo de software (Acosta, 2011). Este método está fundamentado en el análisis centrado en el usuario y en la participación de especialistas con el objetivo de evolucionar el software, a fin de producir un software usable.

AgilUs propone incluir la usabilidad como un aspecto fundamental durante el ciclo de desarrollo de software. Para esto, es necesario que los usuarios finales sean los que guíen el desarrollo del proyecto.

El método *AgilUs* busca proporcionar un conjunto de actividades para construir la usabilidad en el diseño de interfaces de usuario durante el desarrollo de un producto de software. El proceso de desarrollo de software engloba las actividades de requisitos, análisis, prototipaje y entrega; así como las evaluaciones de usabilidad correspondientes a cada etapa del proceso. Estas actividades se realizan en ciclos iterativos hasta alcanzar el producto final.

El desarrollo de un producto de software en *AgilUs* está centrado en varios principios:

- Integra la *Interacción Humano Computador (IHC)* y la *Ingeniería del Software (IS)* como actividades o disciplinas complementarias.
- La usabilidad debe considerarse desde el principio del desarrollo. Esto con el fin de aumentar la calidad del producto de software.

- La usabilidad determina la utilidad. Un software se considera útil en la medida que pueda ser usado a fin de producir resultados, en forma eficiente, intuitiva y satisfactoria para los usuarios.

- El usuario determina la usabilidad. La usabilidad no puede ser medida de igual forma para todos los sistemas existentes. Es por esto que un sistema se puede considerar usable en un contexto determinado para un tipo de usuario específico y es éste último quien determina si el producto final es usable o no.

En *AgilUs* se propone aplicar buenas prácticas de desarrollo de software que están enfocadas en satisfacer las demandas del usuario y el desarrollo iterativo e incremental, haciendo énfasis en lograr la usabilidad en cada paso del desarrollo. Estas prácticas son:

- **Diseño centrado en el usuario:** Este enfoque de diseño y desarrollo está centrado en los deseos, limitaciones y necesidades de los usuarios finales para el desarrollo de un sistema de software. Para lograr esto, es necesario realizar pruebas constantes para verificar el curso que lleva el desarrollo del sistema y la interfaz de usuario. De esta forma, el usuario influye notablemente en el proceso de desarrollo del sistema.

- **Diseño basado en prototipos:** El desarrollo de software está guiado por la construcción de prototipos. En *AgilUs* estos prototipos son evaluados por los usuarios y por especialistas en usabilidad, con el objetivo de ir desarrollando prototipos basados en la retroalimentación de prototipos anteriores a fin de lograr un producto listo para la entrega.

- **Desarrollo ágil, iterativo e incremental:** Este principio recomienda desarrollar un sistema simple que satisfaga la necesidades primordiales o actuales del usuario, preparándose para cambios futuros. El desarrollo incremental permite dar resultados sin la necesidad de especificar todos los requerimientos del sistema al inicio del desarrollo. La iteratividad permite regresar a etapas anteriores una vez recibida la retroalimentación producto de las evaluaciones realizadas.

- **Usabilidad como atributo de la calidad:** Uno de los atributos que es considerado para determinar la calidad de un sistema de software es la usabilidad. *AgilUs* hace énfasis en la producción de software usable a fin de aumentar la calidad del producto de software.

- **Interacción continua con el usuario:** Para *AgilUs* la presencia constante y participativa del usuario es fundamental. El equipo de desarrollo sólo puede tomar decisiones tras realizar evaluaciones de usabilidad, y la usabilidad del sistema sólo puede ser determinada por el usuario.

AgilUs está basado en el desarrollo iterativo e incremental de prototipos que son aprobados por el usuario final hasta que se convierten en el producto final. La modificación de este producto final se puede realizar mediante el mantenimiento correctivo y/o evolutivo del sistema. Esto último no está contemplado como parte del método.

Cada etapa del ciclo de vida de *AgilUs* comprende un conjunto de actividades que, en lo posible, garantizan la usabilidad. El objetivo de estas etapas es proporcionar una manera organizada de desarrollar un producto de software tomando en cuenta la usabilidad en cada actividad. El ciclo de vida engloba la definición de requisitos, análisis, prototipaje y entrega (ver imagen 3.1).

Imagen 3.1: El método AgilUs: etapas, actividades y artefactos.

Esta imagen muestra un diagrama de la relación entre cada una de las etapas del ciclo de vida de *AgilUs*, con las actividades que se realizan y artefactos que se generan en cada etapa. A continuación se describen brevemente las etapas de este método:

- **Requisitos:** Se hace un análisis global del problema que se quiere solucionar, se estudian productos de software similares existentes, se genera un perfil de usuario y, por último, se define la lista de requerimientos a desarrollar.

- **Análisis:** En esta etapa se realiza un análisis de la solución a desarrollar, se emplean diagramas de casos de uso y modelo de objetos del dominio, siguiendo la notación del *lenguaje unificado de modelado* (*UML*, del inglés *Unified Modeling Language*), para definir el conjunto de funcionalidades que tendrá el sistema.

- **Prototipaje:** Se realiza un prototipo rápido de la interfaz de usuario a partir de los patrones de interacción, el cual va evolucionando hasta convertirse en el producto final, se

genera la guía de estilo y se realizan evaluaciones heurísticas y listas de comprobación a fin de verificar la usabilidad del sistema.

- **Entrega:** Se aplican las pruebas al sistema para certificar que la aplicación desarrollada sea un software usable y sin errores. Finalmente se pone en producción la aplicación.

A continuación se describe la *Fundación Escuela de Idiomas Modernos.*

3.2. Aplicación del método AgilUs

El desarrollo del sistema *aTesT* se realiza en dos (2) iteraciones que consisten en la implementación de un módulo para la creación y gestión de preguntas en un banco de preguntas y un módulo para la generación de exámenes. En cada iteración se desarrolla las cuatro (4) etapas descritas en la sección 3.1 y se aplica un conjunto de técnicas que permiten la generación de artefactos de manera incremental. Además, *AgilUs* permite iterar dentro de cada etapa tantas veces como sea necesario, hasta conseguir el producto deseado. A continuación se describe cada iteración:

3.2.1. Iteración I: Banco de preguntas

Esta iteración consta de diversas funcionalidades para la gestión del banco de preguntas. A continuación se explica las técnicas y artefactos generados en cada etapa de *AgilUs* aplicados a esta iteración.

3.2.1.1. Etapa I: Requisitos

En esta etapa se utiliza una serie de técnicas de indagación, los cuales permiten conocer las necesidades y requerimientos del usuario. Alguno de ellos son:

- **Entrevistas.** Se planifican entrevistas con el usuario periódicamente con el objetivo de indagar en cuanto a requerimientos del sistema se refiere. Se proponen soluciones y se discute sobre ellas.

- **Tormenta de ideas.** Se logra intercambiar ideas entre el equipo de desarrollo para dar solución al problema. En la imagen 3.2 se muestra un resumen de las propuestas que surgieron al aplicar esta técnica.

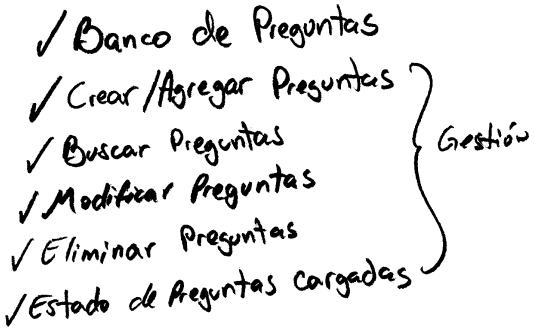


Imagen 3.2: Tormenta de ideas de la primera iteración de *aTesT.*

De aquí se extrajo las ideas principales acerca de las funcionalidades que deben conformar el módulo de preguntas del sistema.

- **Perfiles de usuario.** La comunidad a la que está dirigido el módulo de creación y gestión de preguntas en el banco de preguntas, está constituida por los instructores y la coordinadora de los cursos de idiomas que se imparten en *FUNDEIM*. Estos usuarios requieren conocimientos en el uso del computador para navegar en internet.

- **Requerimientos funcionales y no funcionales.** Dentro de los requerimientos funcionales de esta iteración se tienen los siguientes: Agregar pregunta, modificar pregunta, buscar pregunta, eliminar pregunta, consultar el estado de carga de las preguntas. Entre los requerimientos no funcionales se encuentran: Usabilidad, de manera que el usuario pueda comprender y usar con facilidad el módulo; Disponibilidad, se debe permitir usar el módulo en cualquier momento y desde cualquier lugar con acceso a internet; Mantenibilidad, con el objetivo de poder realizar los cambios necesarios en el menor tiempo posible a fin de garantizar la disponibilidad del sistema; Portabilidad, para que pueda ser utilizado independientemente de la plataforma que se esté utilizando.

3.2.1.2. Etapa II: Análisis

En esta etapa se lleva a cabo el análisis de los requerimientos obtenidos en la etapa anterior y se generaron algunos artefactos, entre ellos:

- **Prototipo en papel.** Se presenta un prototipo en papel, que simula una posible interfaz con la que el usuario debe interactuar para agregar una pregunta. En la imagen 3.3 se observa el prototipo en papel.

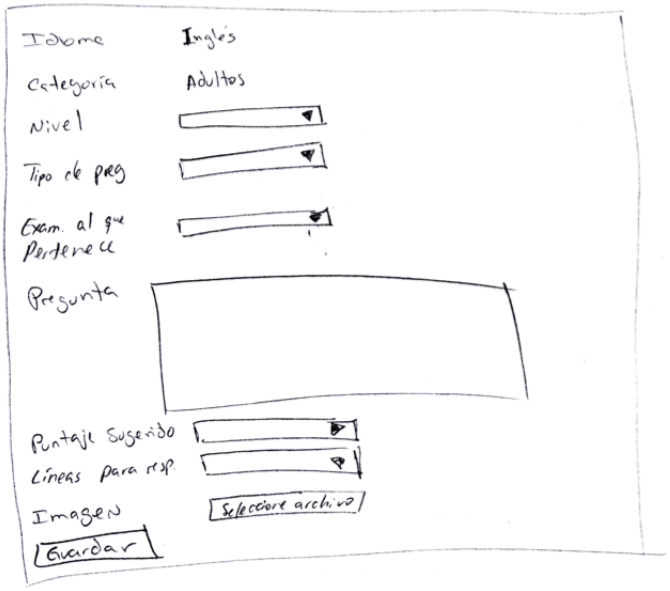


Imagen 3.3: Prototipo en papel para agregar preguntas.

- **Modelo de casos de uso.** Se diseña un diagrama de casos de uso para describir las funcionalidades de esta iteración y la relación existente con los usuarios del mismo. En la imagen 3.4 se muestra el caso de uso generado.

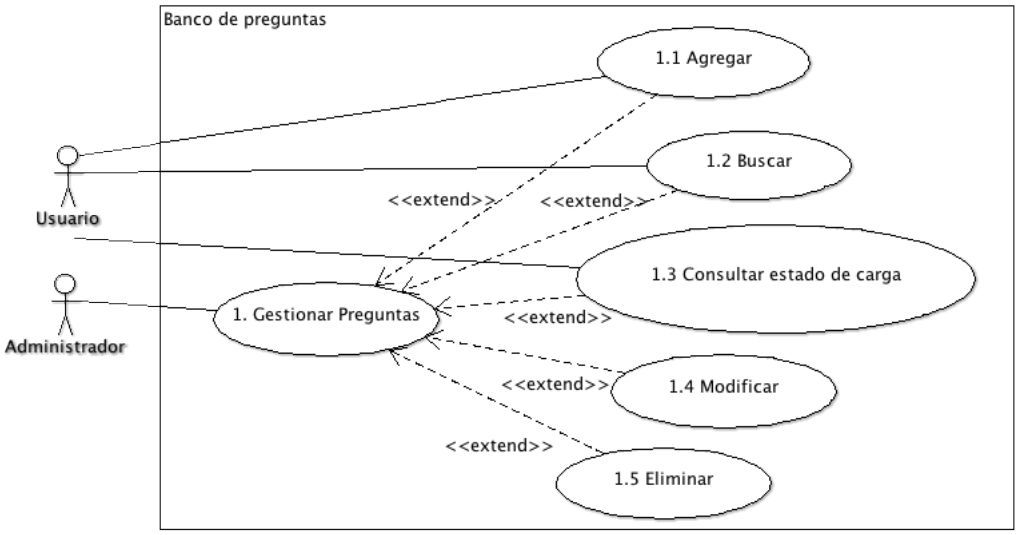


Imagen 3.4: Modelo de casos de uso de la primera iteración de *aTesT*.

Se definen dos (2) actores principales, el usuario, que representa a los instructores de los cursos que se imparten en *FUNDEIM* y el administrador, que representa a la coordinadora de los cursos. Dentro de las funcionalidades se encuentran:

1. Gestionar preguntas. Permite realizar diversas opciones para la gestión de las preguntas. Entre ellas están: agregar pregunta, buscar pregunta, consultar estado de preguntas cargadas, modificar y eliminar preguntas.

1.1 Agregar. Brinda la posibilidad ingresar preguntas al sistema. Para ello, se debe indicar el nivel, la categoría, el tipo de pregunta, el examen al que pertenece, el contenido de la pregunta, el puntaje sugerido, entre otros.

1.2 Buscar. Permite visualizar un conjunto de preguntas de acuerdo a un criterio de búsqueda establecido. Para ello, se debe indicar al menos uno de los siguientes criterios de búsqueda: Nivel, tipo de pregunta, examen al que pertenece o contenido.

1.3 Consultar estado de carga. Permite conocer cuantas preguntas han sido cargadas en el sistema, indicando el usuario y la cantidad de preguntas agregadas por dicho

usuario.

1.4 Modificar. Brinda la posibilidad de modificar una pregunta existente en el sistema, con la finalidad de poder corregir posibles errores.

1.5 Eliminar. Esta funcionalidad permite eliminar una pregunta del sistema.

- **Modelo de objetos del dominio.** Para la elaboración del modelo de objetos del dominio, se analizó . De esta manera, en la imagen 3.5 se aprecia el modelo de objetos del dominio generado.

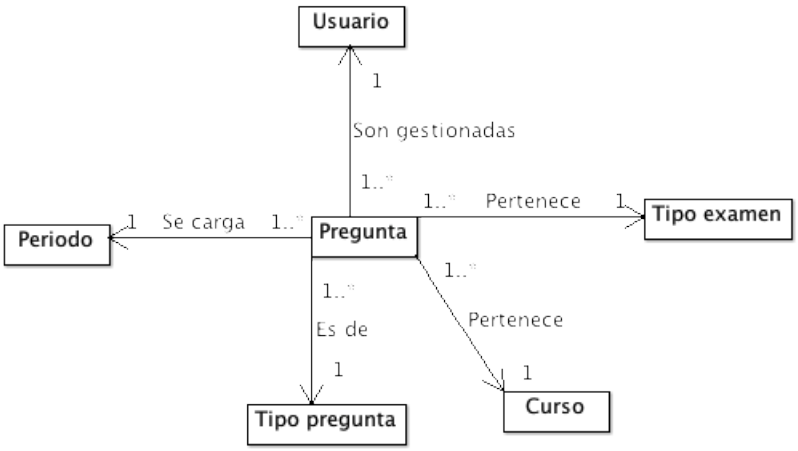


Imagen 3.5: Modelo de objetos del dominio de la primera iteración de *aTesT.*

Los objetos que conforman este diagrama son los siguientes: Pregunta, que representa una pregunta dentro del sistema; Periodo, que representa los periodos en que se dictan los cursos de idiomas; Usuario, que es quien gestiona las preguntas dentro del sistema; Tipo examen, para indicar si la pregunta pertenece al primer o segundo examen; Curso, que representa cada curso en sus distintos niveles y categorías*;* Tipo pregunta, para indicar si la pregunta es de redacción, lectura, gramática, entre otros.

3.2.1.3. Etapa III: Prototipaje

En esta etapa, se obtienen algunos artefactos y se realizan evaluaciones con el fin que el usuario no tenga la necesidad de esperar hasta el final del desarrollo del sistema para comen-

zar a usarlo y comunicar sus aportes como miembro del equipo de desarrollo de software. Alguno de ellos son:

- **Prototipo ejecutable.** Se desarrolla un prototipo ejecutable a partir de los artefactos generados en las etapas anteriores, con la finalidad que el usuario pueda interactuar con los módulos que conforman el sistema, sin la necesidad que todos hayan sido desarrollados completamente. En una primera iteración sobre esta etapa se desarrolla un prototipo ejecutable para la funcionalidad de agregar pregunta (ver imagen 3.6).

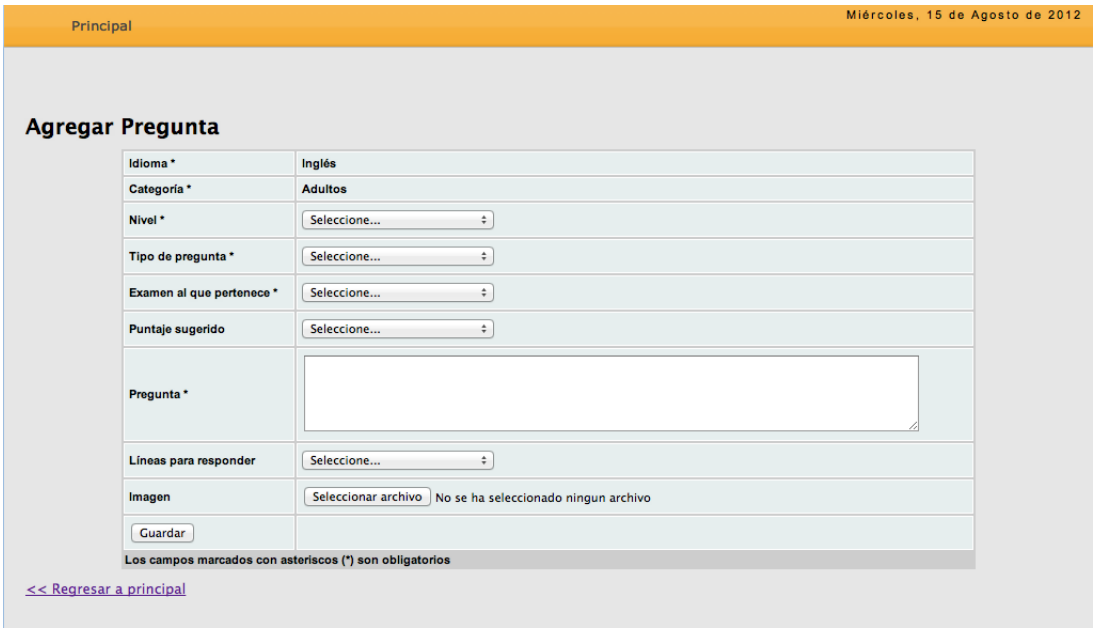


Imagen 3.6: Primer prototipo ejecutable para agregar preguntas.

Para una segunda iteración sobre este prototipo, el usuario final realiza sugerencias que son aplicadas y mostradas para su aprobación. Además, se realizan los prototipos para el resto de funcionalidades que comprenden la primera iteración del desarrollo de *aTesT*. En la imágenes 3.7, 3.8 y 3.9 se muestra algunas de las interfaces del prototipo ejecutable.

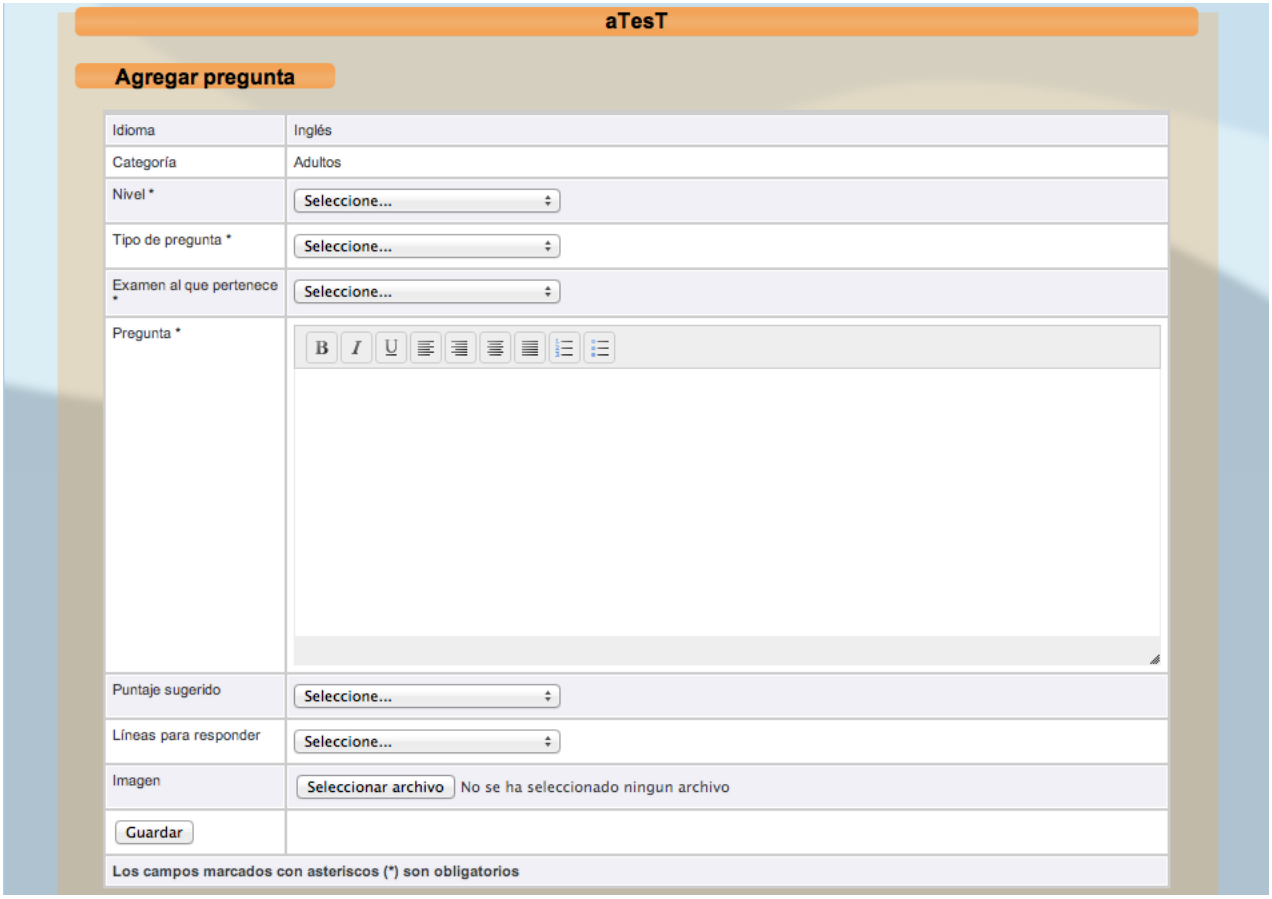
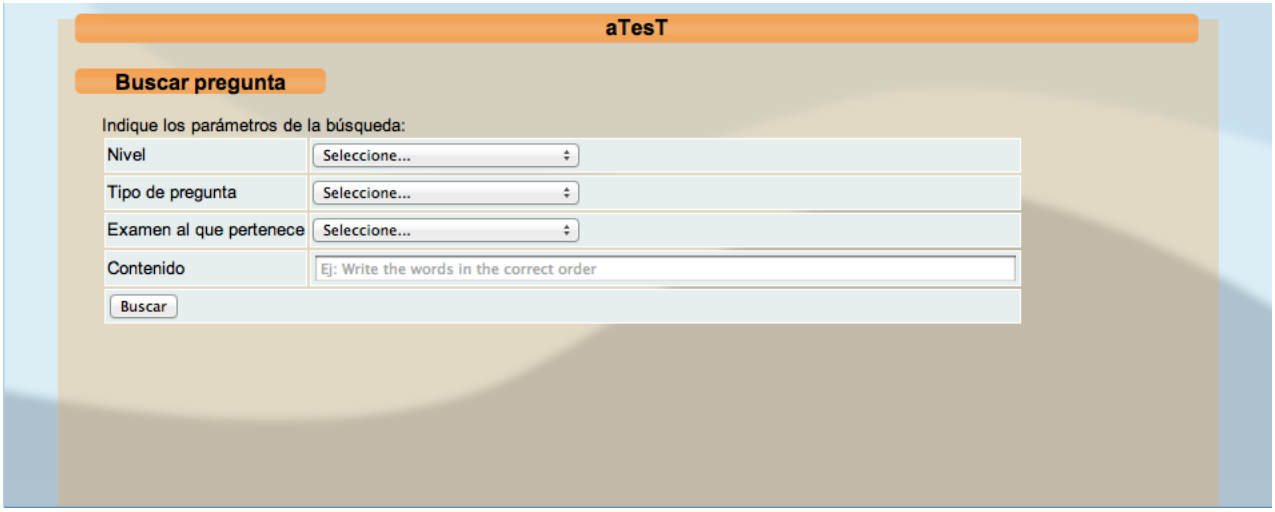
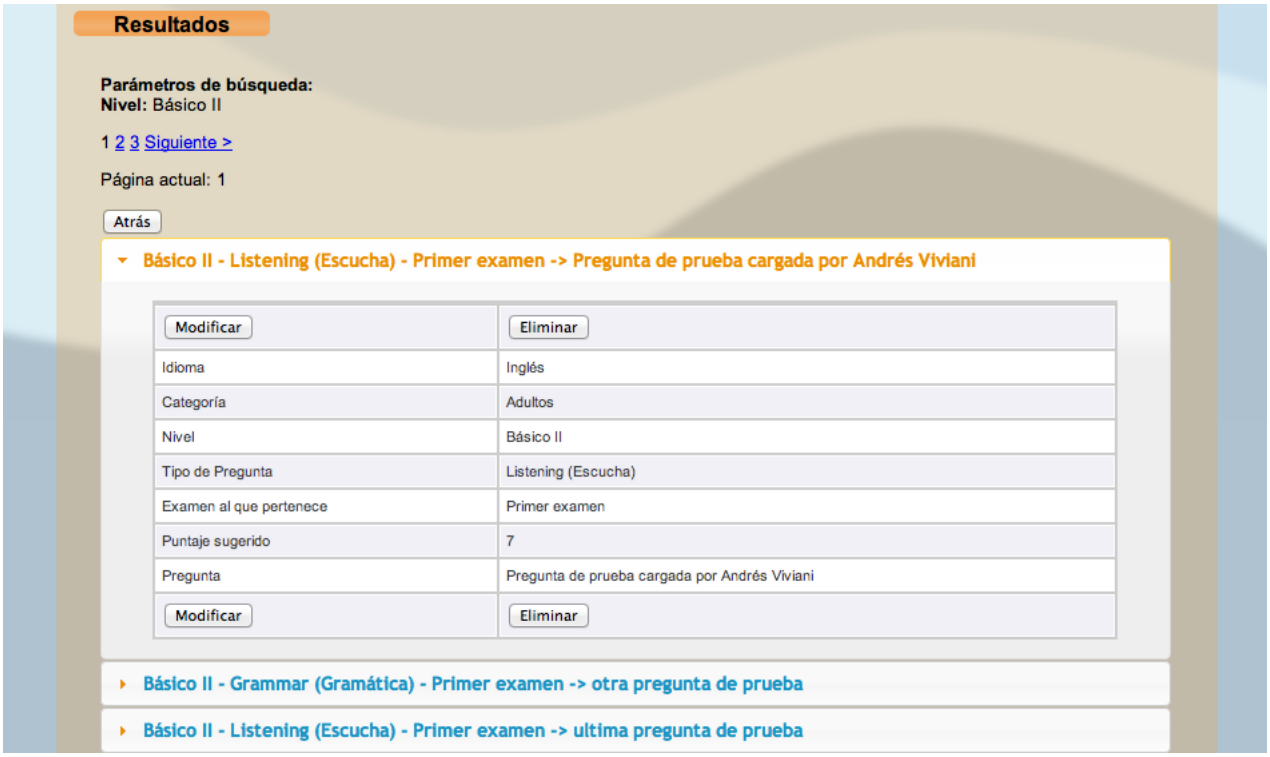


Imagen 3.7: Prototipo ejecutable para agregar preguntas al banco de preguntas.

Esta interfaz permite a los usuarios agregar preguntas al banco de preguntas. Para lograr ingresar una nueva pregunta, es necesario completar al menos todos los campos obligatorios, marcados con asteriscos (*).



(a) Prototipo para ingresar los criterios de búsqueda.



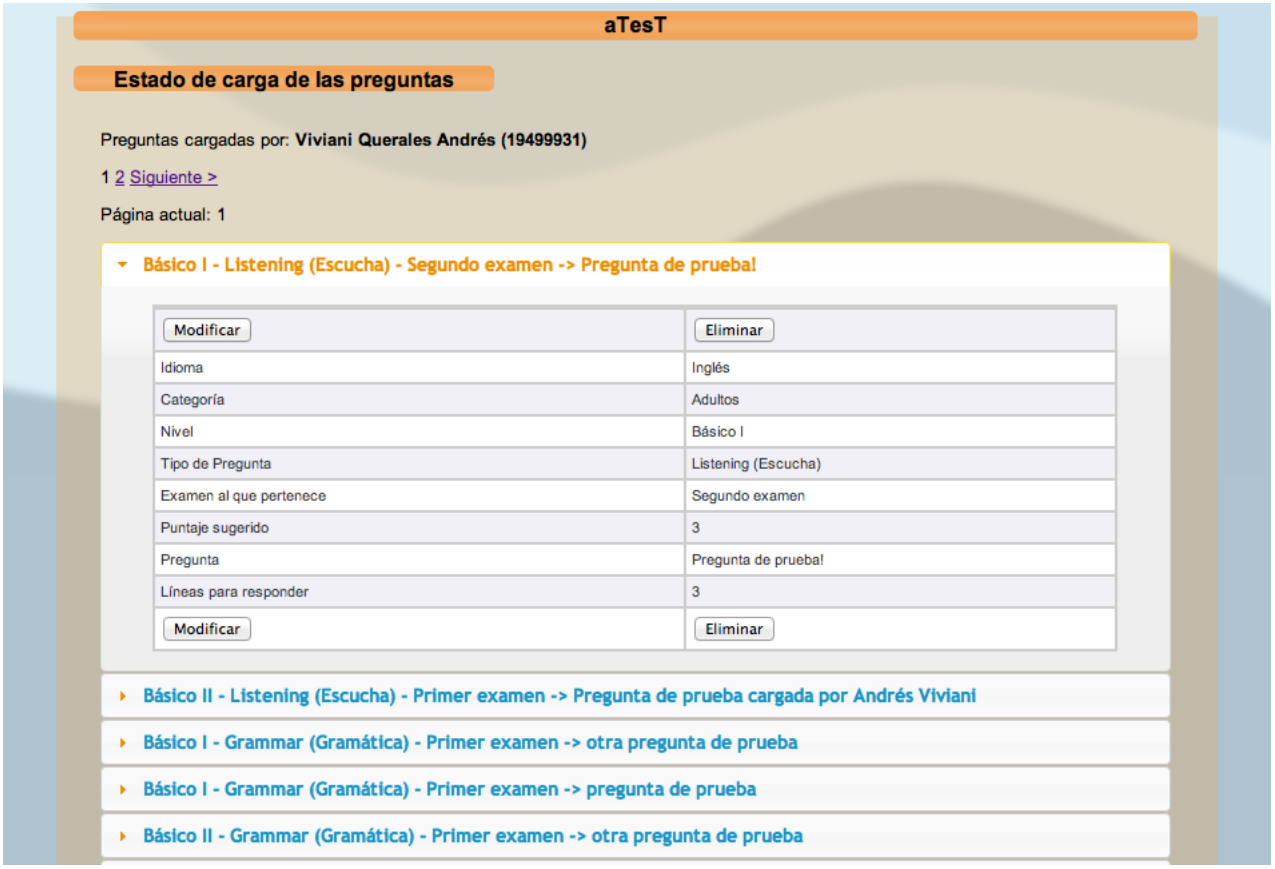
(b) Prototipo para mostrar los resultados de la búsqueda.

Imagen 3.8: Prototipo ejecutable para la búsqueda de preguntas.

Estas interfaces permiten realizar la búsqueda de preguntas en el banco con cuatro (4) criterios diferentes: el tipo de pregunta, el nivel al cual están destinadas las preguntas, el examen al que pertenecen y un filtro que permite colocar parte del contenido deseado

en la pregunta.

(a) Prototipo para ver el listado de preguntas general.



(b) Prototipo para ver el listado de preguntas individual.

Imagen 3.9: Prototipo ejecutable para consultar el estado de las preguntas cargadas.

La primera interfaz permite consultar el estado de carga de las preguntas en un resumen, indicando para todos los usuarios el nombre y la cantidad de preguntas cargadas. La

segunda vista muestra el detalle de cada pregunta cargada por cada usuario.

- **Evaluación Heurística.** Este tipo de pruebas se realizan con el fin de encontrar problemas de usabilidad en la interfaz de usuario. Para ello, es necesario utilizar las heurísticas creadas por Jakob Nielsen, ellas son (Wikipedia, Heurísticas de Nielsen, , 2012):

  H1. Visibilidad de sistema. El sistema debe informar a los usuarios del estado del sistema, dando una retroalimentación apropiada en un tiempo razonable.

  H2. Utilizar el lenguaje de los usuarios. El sistema debe utilizar el lenguaje de los usuarios, con palabras o frases que le sean conocidas, en lugar de los términos que se utilizan en el sistema, para que al usuario no se le dificulte utilizar el sistema.

  H3. Control y libertad para el usuario. En caso que los usuarios elijan una opción del sistema por error, éste debe contar con las opciones de deshacer y rehacer para proveer al usuario de una salida fácil sin tener que utilizar diálogo extendido.

  H4. Consistencia y estándares. El usuario debe seguir las normas y convenciones de la plataforma sobre la que está implementando el sistema, para que no se tenga que preguntar el significado de las palabras, situaciones o acciones del sistema.

  H5. Prevención de errores. Es más importante prevenir la aparición de errores que de generar buenos mensajes de error.

  H6. Minimizar la carga de la memoria del usuario. En la interfaz se deben mantener los objetos, acciones y opciones visibles para que el usuario no pierda el tiempo recordando la información de una parte del diálogo a otra.

  H7. Flexibilidad y eficiencia de uso. Los aceleradores permiten aumentar la velocidad de interacción para el usuario experto tal que el sistema pueda atraer a usuarios principiantes y experimentados. Las instrucciones para el uso del sistema tienen que ser visibles o fácilmente accesibles cada vez que se necesiten, para que al usuario inexperto se le facilite aprender a utilizar el sistema. Es importante que el sistema permita personalizar acciones frecuentes, para acelerar el uso del sistema.

H8. Los diálogos estéticos y diseño minimalista. La interfaz no debe contener información que no sea relevante o se utilice raramente, pues cada unidad adicional de información en un diálogo compite con las unidades relevantes de la información y disminuye su visibilidad relativa.

H9. Ayudar a los usuarios a reconocer, diagnosticar y recuperarse de los errores. Los mensajes de error deben expresarse en un lenguaje claro, indicar exactamente el problema, y deben ser constructivos.

H10. Ayuda y documentación. En caso que el sistema necesite disponer de ayuda y documentación, ésta tiene que ser fácil de encontrar, centrada en las tareas del usuario, tener información de las etapas a realizar y que no sea muy extensa.

En la tabla 3.1 se muestran los problemas encontrados durante la evaluación aplicada a cuatro (4) personas.

Problema	**Heurísticas**	**Valoración**	**Solución**
El puntaje sugerido debe preguntarse una vez escrita la pregunta	H5	0	Colocar el puntaje sugerido luego de la pregunta
Modificar pregunta luego de crearla	H9	2	Colocar la opción de modificar una vez creada la pregunta
Opción de devolverse al buscar una pregunta	H3	1	Colocar el botón de atrás luego de buscar una pregunta
Eliminar pregunta luego de una búsqueda	H6	4	Colocar la opción de eliminar pregunta en los resultados de una búsqueda

Tabla 3.1: Resultados de la evaluación heurística de la primera iteración de *aTesT*.

3.2.1.4. Etapa IV: Entrega

En esta etapa se prepara el ambiente para colocar en sistema en producción. Para ello, se hace uso de una herramienta para el control de versiones denominada *Git*. Esta herramienta se utiliza como mecanismo del grupo de desarrollo para manejar versiones y mantener la

consistencia en el código fuente de la aplicación. Con la instalación del *Manejador de Versiones de Ruby,* se instala la versión 1.9.3 de *Ruby* con la versión 3.2.2 de *Rails,* además de un conjunto de gemas necesarias para el funcionamiento de la aplicación.

Esta etapa, además permite identificar problemas de consistencia entre los ambientes de desarrollo y producción para que sean corregidos y evitar fallos en la puesta en producción.

3.2.2. Iteración II: Generación de exámenes

Esta iteración consta de diversas funcionalidades para la generación y gestión de exámenes. A continuación se explica las técnicas y artefactos generados en cada etapa de *AgilUs* aplicados a esta iteración.

3.2.2.1. Etapa I: Requisitos

En esta etapa se utiliza una serie de técnicas de indagación, los cuales permiten conocer las necesidades y requerimientos del usuario. Alguno de ellos son:

- **Entrevistas.** Se realizan entrevistas con el usuario final y se plantean ideas para la realización de exámenes.
- **Tormenta de ideas.** Se intercambian ideas para dar solución al problema. En la imagen 3.10 se muestra un resumen de las propuestas que surgieron al aplicar esta técnica.

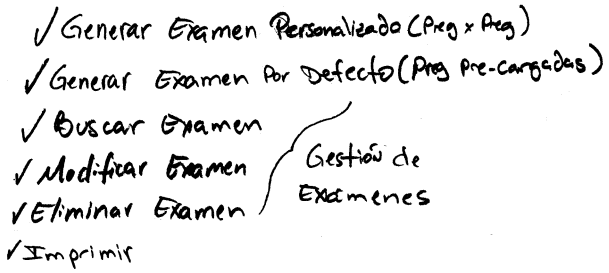


Imagen 3.10: Tormenta de ideas de la segunda iteración de *aTesT.*

De esta tormenta de ideas, se obtienen las principales funcionalidades que debe tener el módulo de generación de exámenes.

- **Lista de requerimientos.** Dentro de los requerimientos de esta iteración se tienen los siguientes:

  - Generar examen personalizado.
  - Generar examen por defecto.
  - Buscar examen.
  - Eliminar examen.
  - Modificar examen.

3.2.2.2. Etapa II: Análisis

En esta etapa se lleva a cabo el análisis de los requerimientos obtenidos en la etapa anterior y se generaron algunos artefactos, entre ellos:

- **Modelo de casos de uso.** Se diseña un diagrama de casos de uso para describir las funcionalidades del módulo y la relación existente con los usuarios del mismo. En la imagen 3.11 se muestra el caso de uso generado.

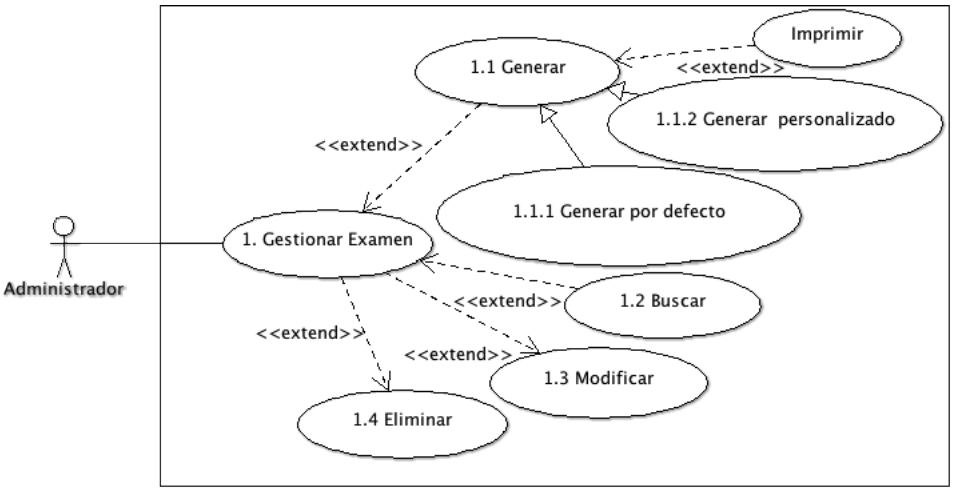


Imagen 3.11: Modelo de casos de uso de la segunda iteración de *aTesT*.

Se define un (1) actor principal, el administrador, que representa a la coordinadora de los cursos. Dentro de las funcionalidades se encuentran:

1. Gestionar examen. Esta funcionalidad permite la toma de decisiones sobre las acciones a ejecutar en este módulo.

1.1 Generar. Brinda la posibilidad de generar un examen a partir de un conjunto de preguntas almacenadas en el banco de preguntas.

1.1.1 Generar personalizado. Esta funcionalidad permite generar una prueba partiendo de una evaluación sin preguntas, con el objetivo que el usuario seleccione una a una cada pregunta.

1.1.2 Generar por defecto. Con esta funcionalidad se genera una evaluación de forma automática, con el objetivo que el usuario solo invierta tiempo en corregir y no en desarrollar el examen.

1.2 Buscar. Esta opción permite introducir criterios de búsqueda para recuperar exámenes realizados previamente.

1.3 Modificar. Permite modificar un examen existente.

1.4 Eliminar. Esta opción permite al usuario eliminar un examen del sistema.

- **Prototipo en papel.** Se presenta un prototipo en papel, que simula una posible interfaz con la que el usuario debe interactuar para generar un examen de forma manual. En la imagen 3.12 se observa el prototipo en papel.

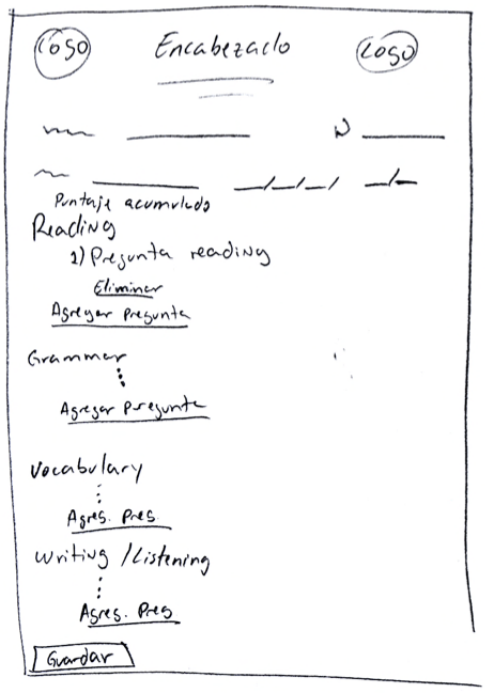


Imagen 3.12: Prototipo en papel para generar un examen.

- **Modelo de objetos del dominio.** En la imagen 3.13 se muestra el modelo de objetos del dominio que se genera en esta etapa en función al módulo de generación de exámenes.

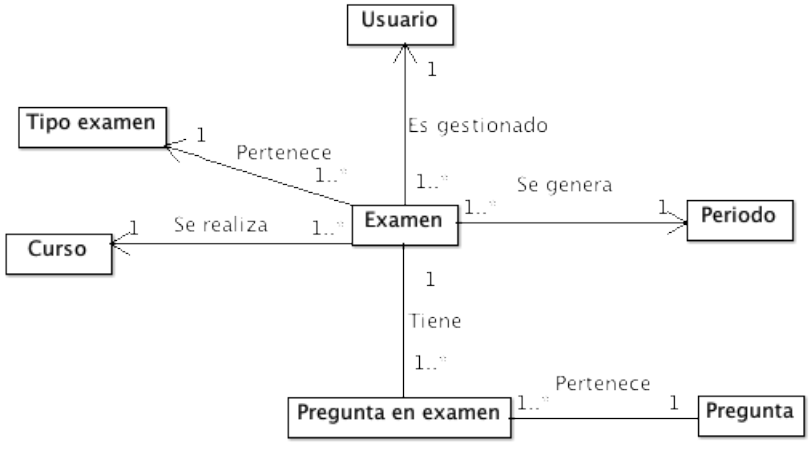


Imagen 3.13: Modelo de objetos del dominio de la segunda iteración de *aTesT*.

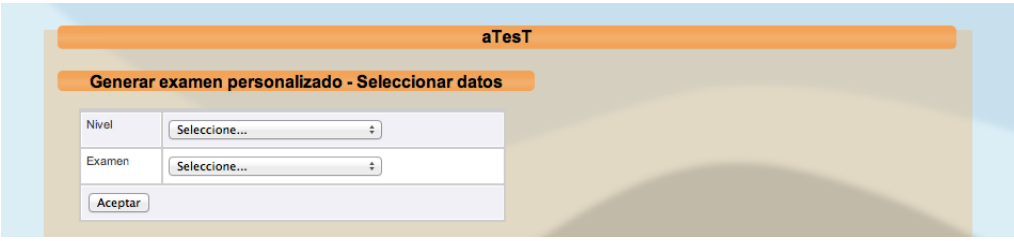
Los objetos que conforman este diagrama son los siguientes: Examen, que representa un examen dentro del sistema; Periodo, que representa los periodos en que se dictan

los cursos de idiomas; Usuario, que es quien gestiona los exámenes dentro del sistema con el rol de administrador; Tipo examen, para indicar si la evaluación es el primer o segundo examen; Curso, que representa cada curso en sus distintos niveles y categorías dentro de *ACEIM;* Pregunta en examen, que permite mantener un conjunto de preguntas relacionadas a un examen; Pregunta, que representa al modelo de objetos del dominio de la primera iteración.

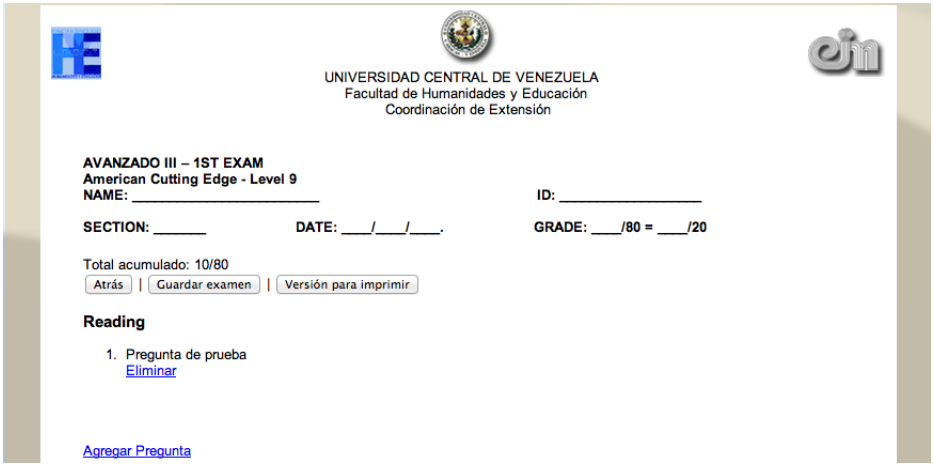
3.2.2.3. Etapa III: Prototipaje

En esta iteración se desarrollan técnicas y se generan artefactos comprendidos en método de desarrollo *AgilUs.* Algunos de ellos son:

- **Prototipo ejecutable.** Una vez analizados los requerimientos, se realiza un prototipo ejecutable que permita interactuar con el módulo de desarrollo de exámenes. En las imágenes 3.14, 3.15 y 3.16 se pueden apreciar algunas de las interfaces desarrolladas.



(a) Prototipo para ingresar los criterios para generar un examen manualmente.

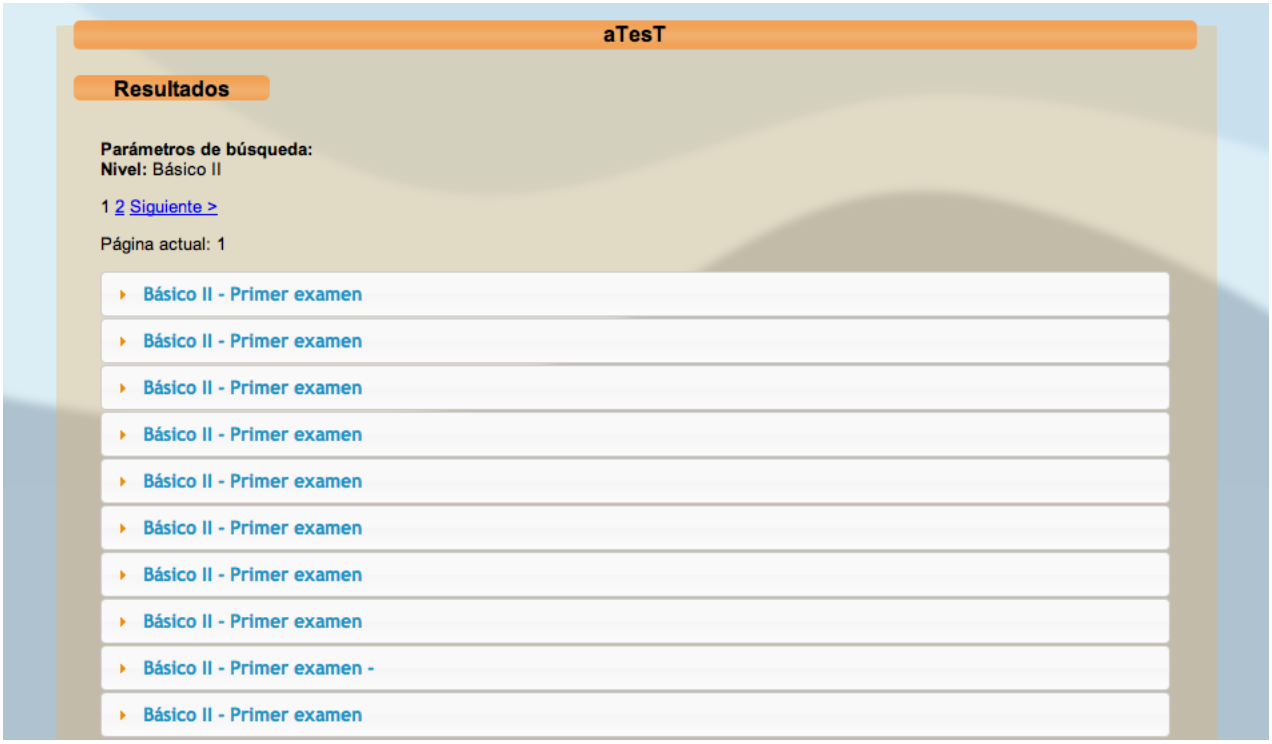


(b) Prototipo para generar un examen.

Imagen 3.14: Prototipo ejecutable para la generación de exámenes.

Estas interfaces permiten la generación de un examen de forma personalizada. La primera interfaz permite ingresar el nivel y el examen que se desea generar y la segunda vista muestra una interfaz para ir agregando y eliminando preguntas dentro del examen. Para la generación de un examen por defecto, la interfaz es similar, con la particularidad que el examen se muestra con preguntas pre-cargadas y el usuario final puede agregar o eliminar preguntas.

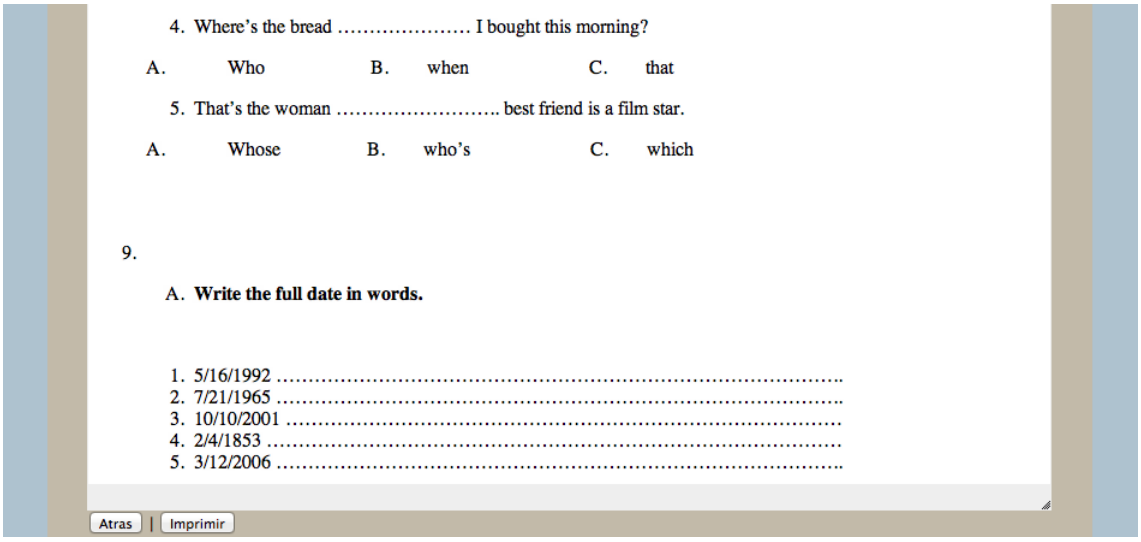
(a) Prototipo para ingresar los criterios de búsqueda de exámenes.



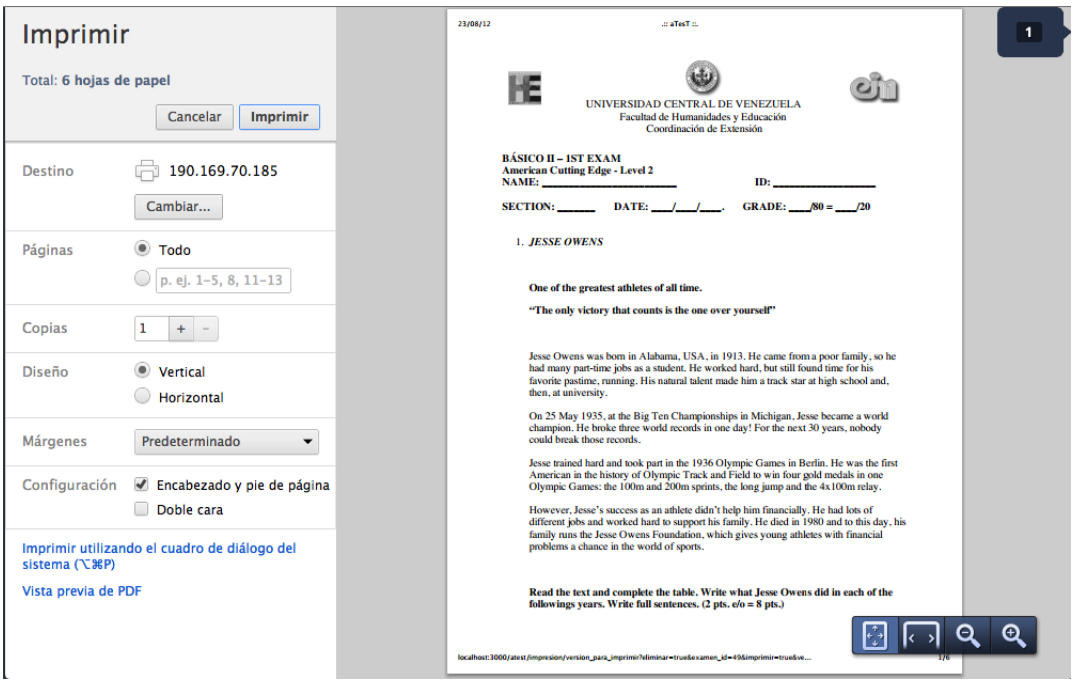
(b) Prototipo para mostrar los resultados de la búsqueda de exámenes.

Imagen 3.15: Prototipo ejecutable para la búsqueda de preguntas.

Estas vistas permiten la búsqueda de exámenes según diversos criterios, una vez encontrados, se muestra una lista desplegable en forma de acordeón.



(a) Interfaz que permite la edición del documento a imprimir.



(b) Vista del *navegador web Google Chrome* que permite la impresión del documento generado.

Imagen 3.16: Prototipo ejecutable para la impresión de exámenes.

Una vez generado el examen, la opción para imprimir muestra una interfaz que permite editar el examen completo y su posterior impresión. En una primera iteración sobre este prototipo solo se permite la impresión del documento actual. En una segunda iteración, por recomendación del usuario final, se agrega la posibilidad de almacenar las versiones para imprimir de un examen, de manera que se pueda tener varias versiones para imprimir de un mismo examen.

- **Lista de comprobación.** Se aplicó esta técnica a posibles usuarios del sistema, así como también a personas del área de educación que realizan evaluaciones progresivas. Ésta permitió identificar fallas en el cumplimiento del *control y libertad para el usuario; y* en *minimizar la carga de la memoria del usuario.* A las personas a las que se le aplicó esta técnica, se les facilitó la interfaz de usuario de la aplicación para la realización de distintas actividades relacionadas con la generación de exámenes. Los participantes sugirieron colocar el puntaje original del examen en la versión para imprimir, con el objetivo de evitar recordarlo de la interfaz anterior. También solicitaron la posibilidad de retornar a vistas anteriores en cada paso de la generación de un examen.

3.2.2.4. Etapa IV: Entrega

En esta etapa se actualizan las versiones del proyecto en el repositorio *Git* y se actualiza la versión de la aplicación que está en el servidor con la finalidad de agregar las nuevas funcionalidades al sistema. Finalmente, se entrega el software y es puesto en producción con todas las funcionalidades que se enmarcan en este *trabajo especial de grado,* con la finalidad de realizar pruebas piloto para la generación de algunos exámenes del periodo C-2012 de los cursos de inglés.

Conclusiones y recomendaciones

Las *tecnologías de la información y comunicación* ofrecen beneficios y ventajas a todo nivel. En el ámbito educativo, permiten el aprendizaje cooperativo, la alfabetización tecnológica, el aprovechamiento de recursos, mayor comunicación, el aprendizaje colectivo, mejor administración y distribución del conocimiento, entre otros. Al igual que muchas organizaciones, *FUNDEIM* se ha visto beneficiada con el uso de *Aplicaciones Web* para mejorar sus servicios. Sin embargo, existen procesos dentro de la fundación que se realizan de forma no automatizada como la generación de exámenes escritos, razón por la cual se desarrolla *aTesT*.

El desarrollo de *aTesT* siguiendo el método *AgilUs* permitió implementar un conjunto de técnicas y actividades de forma organizada y rápida. Además, *AgilUs* fue posible adaptar a las necesidades por parte del equipo de desarrollo y del usuario para la implementación de esta aplicación. Por otra parte, involucrar al usuario final desde el principio del desarrollo del sistema, es un factor clave de este método que permite adaptar el software a las necesidades de éste y no viceversa. La característica iterativa e incremental de este método también es un factor relevante que permite dividir el sistema en subsistemas que agrupan distintas funcionalidades que se entregan progresivamente para que puedan ser utilizadas, sin la necesidad de implementar la aplicación en su totalidad.

Las tecnologías del lado cliente que fueron utilizadas para el desarrollo de esta aplicación fueron apropiadas y de vanguardia; y facilitaron la implementación de la misma. No hubo inconvenientes en cuanto al uso de estas tecnologías.

Las tecnologías del lado del servidor también fueron apropiadas y de vanguardia. Con *Rails,* se generaron problemas en cuanto a fallas que presentaba el *framework*, pero que fue posible resolver tras la instalación de algunas *gemas.* Otro problema encontrado, fue la discrepancia existente entre los ambientes de ejecución de la aplicación, específicamente entre el ambiente de desarrollo y producción.

El objetivo general de esta investigación se cumplió. Actualmente *FUNDEIM* cuenta con un nuevo sistema para el diseño de los exámenes escritos de forma automatizada.

Se logró realizar el levantamiento de la información y los requerimientos de la institución, desde el momento en que se elaboró el seminario previo a este a *Trabajo Especial de Grado*. Se estudió la situación que tenía *FUNDEIM* en relación a la generación de exámenes escritos.

Como método de desarrollo ágil se empleó *AgilUs.* Se pudo adaptar a las necesidades del equipo de desarrollo y del usuario final para la elaboración de *aTesT.* Para implementar el módulo que permita gestionar el banco de preguntas, así como la gestión y generación de exámenes escritos y su posterior impresión, se elaboraron listas de requerimientos que engloban todas las funcionalidades para estas actividades. Además, se desarrolla cada requerimiento como una funcionalidad del sistema y se muestra al usuario para determinar su aprobación. Cada módulo, tras varias iteraciones, fue aceptado por el usuario final.

Actualmente *aTesT* cuenta con un banco de preguntas con doscientos sesenta y cuatro (264) preguntas, cuya distribución se aprecia en las tablas 3.2 y 3.3, correspondientes a las preguntas pertenecientes al primer y segundo examen respectivamente.

Nivel	Reading	Grammar	Vocabulary	Wrinting	Listening
Básico I	0	12	5	1	1
Básico II	4	9	6	4	0
Básico III	4	12	3	2	0
Intermedio I	1	5	1	1	0
Intermedio II	2	4	2	1	0
Intermedio III	0	3	1	3	0
Avanzado I	5	8	1	1	0
Avanzado II	2	9	3	2	0
Avanzado III	1	6	3	2	0

Tabla 3.2: Distribución de las preguntas pertenecientes al primer examen.

Con estas preguntas se puede generar al menos un (1) primer examen para los siguientes niveles: Básico II, básico III, avanzado II y avanzado III.

Nivel	Reading	Grammar	Vocabulary	Wrinting	Listening
Básico I	3	11	2	3	5
Básico II	0	6	3	2	1
Básico III	4	16	2	2	2
Intermedio I	3	14	3	4	2
Intermedio II	1	5	0	1	0
Intermedio III	2	5	1	1	0
Avanzado I	4	5	1	1	0
Avanzado II	1	5	2	1	0
Avanzado III	2	4	2	2	0

Tabla 3.3: Distribución de las preguntas pertenecientes al segundo examen.

Con estas preguntas se puede generar al menos un (1) segundo examen para el siguiente nivel: Intermedio I. Sin embargo, se siguen agregando preguntas al banco de preguntas. En poco tiempo, se podrá generar al menos un (1) examen para todos los niveles. Para los cursos en los que aún no se pueden generar exámenes, faltan entre una (1) y dos (2) preguntas para que se pueda realizar el diseño de los mismos.

Otro beneficio importante que se deriva del desarrollo de *aTesT*, es que se pueden realizar exámenes diferentes para los distintos horarios de un curso. Esto representaba un problema que, por razones de tiempo, no podía ser solventado.

En cuanto a la usabilidad, *AgilUs* propone técnicas que permiten medir este factor de calidad en un sistema. Se aplicaron evaluaciones heurísticas a distintos usuarios con el perfil necesario para usar la aplicación y se encontraron problemas de usabilidad que fueron solventados. Además se facilitó la interfaz de usuario de los distintos módulos a los potenciales usuarios del sistema y éstos aceptaron la usabilidad de los mismos.

Recomendaciones

La aplicación *aTesT* es el comienzo de una idea motivada por el esfuerzo requerido para la realización de exámenes escritos en la *Fundación Escuela de Idiomas Modernos.* Como la mayoría de las nuevas ideas, siempre surgen posibles mejoras que con el tiempo se deben ir desarrollando. Todo software de calidad requiere mantenimiento adaptativo, correctivo y preventivo para garantizar su calidad en el tiempo. A continuación se listan las posibles mejoras y recomendaciones que se pueden hacer en función a este trabajo:

- Agregar más preguntas al banco de preguntas.
- Elaborar un módulo que permita incluir más idiomas para agregar preguntas.
- Posibilitar la configuración del encabezado de los exámenes.
- Elaborar un módulo que permita gestionar las categorizaciones de las preguntas en el banco de preguntas.
- Posibilitar la configuración de variables como lo son: puntaje total de los exámenes, mínimo de preguntas requerido para generar un examen, entre otros.

Este trabajo permite la automatización de un proceso importante dentro de la *Fundación Escuela de Idiomas Modernos*, logrando así un ahorro en tiempo y recurso humano que puede ser utilizado en otros procesos como la gestión y planificación de los cursos. Además, la fundación cuenta con un repositorio de preguntas que permite una rápida búsqueda y manipulación de las mismas, así como un repositorio de exámenes que pueden ser consultados, modificados e incluso impresos nuevamente. Además, este trabajo puede ser base para otro tipo de evaluaciones que se realizan en la fundación que pueden ser aplicadas en este dominio, como las pruebas de nivelación, entre otras.

Referencias bibliográficas

Acosta, A. E. (2011). AgilUs: Construcción ágil de la Usabilidad.

Bach, C. (2011). jQuery plugin: Tablesorter 2.0. Consultado el 12 de Diciembre de 2011, en http://tablesorter.com/docs/

Burnham, T. (2011). CoffeeScript: Accelerated JavaScript Development.

Catlin, H. (2011). #haml. Consultado el 12 de Diciembre de 2011, en http://haml-lang.com/

Catlin, H. (2011). Sass - Syntactically Awesome Stylesheets. Consultado el 12 de Diciembre de 2011, en http://sass-lang.com/

CoffeeScript. Consultado el 5 de Enero de 2012, en http://coffeescript.org/

Educational Testing Service. (2012). ETS. Consultado el 12 de Enero de 2012, en http://www.ets.org/

Educational Testing Service. (2012). TOEFL. Consultado el 12 de Enero de 2012, en http://www.ets.org/toefl/

FUNDEIM. (2005). Consultado el 11 de Enero de 2012, en http://www.fundeim.org/

Flanagan D., Matsumoto Y. (2008). The Ruby programming language. Sebastopol: O´Reilly.

Gamma, E., Helm R., Johnson R., Vlissides, J. (1998). Design Patterns CD, Elements of Reusable Object-Oriented Software.

Garrett, J. Ajax: A New Approach to Web Applications. Consultado el 12 de Diciembre de 2012, en http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications

Hansson, D. H. (2011). Ruby on Rails. Consultado el 12 de Diciembre de 2011, en http://weblog.rubyonrails.org/

IELTS Partners. (2011). IELTS - International English Language Testing System. Consultado el 12 de Enero de 2012, en http://www.ielts.org/

Jquery. (2010). jQuery: The Write Less, Do More, JavaScript Library. Consultado el 12 de Diciembre de 2011, en http://jquery.com/

Jquery UI. (2010). jQuery UI. Consultado el 12 de Diciembre de 2011, en http://jqueryui.com/

Mora, S. L. (2002). Programación de aplicaciones web: historia, principios básicos y clientes web. (E. C. Universitario, Ed.) España.

Mora, S. L. (2001). Programación en Internet: Clientes WEB. España: Imprenta Gamma.

Oracle Corporation. (2012). MySQL. Consultado el 7 de Enero de 2012, en http://www.mysql.com/about/

Rescorla, E. HTTP Over TLS. Consultado el 10 de Enero de 2012, http://www.ietf.org/rfc/rfc2818.txt

Ruby Programming Language. (2011). Consultado el 12 de Diciembre de 2011, en http://www.ruby-lang.org/en/

Seguin, W. E. (2011). RVM. Consultado el 6 de Enero de 2012, en https://rvm.beginrescueend.com/

Sommerville, I. (2005). Ingeniería del Software (7ma. Edición ed.). Madrid, España.

Spratt, M., Alan, P., & Williams, M. (2005). The Teaching Knowledge Test Course.

Stallings, W. (2004). Comunicaciones y Redes de Computadoras (7ma. Edición ed.). (P. P. Hall, Ed.) Madrid, España: Pearson Educación.

W3C. (2011). HTML & CSS. Consultado el 12 de 12 de 2011, en http://www.w3.org/standards/webdesign/htmlcss

W3C. (2011). W3C Semantic Web Activity. Consultado el 12 de 12 de 2011, en http://www.w3.org/2001/sw/

Weber, G. jquery-uitablefilter - GitHub. Consultado el 12 de Diciembre de 2011, en https://github.com/gregwebs/jquery-uitablefilter

Weber, G. UI Table Filter | jQuery Plugins. Consultado el 12 de Diciembre de 2011, en http://archive.plugins.jquery.com/project/uiTableFilter

Wells, D. Extreme Programming: A Gentle Introduction. Consultado el 10 de Enero de 2012, en http://www.extremeprogramming.org/

Wikipedia. Ajax (programming). Consultado el 12 de Diciembre en 2012, de http://en.wikipedia.org/w/index.php?title=Ajax_(programming)

Wikipedia. Client-side. Consultado el 15 de 12 de 2011, en http://en.wikipedia.org/wiki/Client-side

Wikipedia. CoffeeScript. Consultado el 5 de Enero de 2012, en http://en.wikipedia.org/wiki/CoffeeScript

Wikipedia. Extreme programming. Consultado el 10 de Enero de 2012, en http://en.wikipedia.org/wiki/Extreme_programming

Wikipedia. Framework. Consultado el 29 de Julio de 2012, en http://es.wikipedia.org/wiki/Framework

Wikipedia. Heurísticas de Nielsen. Consultado el 29 de Julio de 2012, en http://es.wikipedia.org/wiki/Heur%C3%ADsticas_de_Nielsen

Wikipedia. IELTS. Consultado el 12 de Enero de 2012, en http://es.wikipedia.org/wiki/IELTS

Wikipedia. JavaScript. Consultado el 12 de 12 de 2012, en http://en.wikipedia.org/wiki/JavaScript

Wikipedia. Manifiesto ágil. Consultado el 8 de Enero de 2012, en http://es.wikipedia.org/wiki/Manifiesto_ágil

Wikipedia. MySQL. Consultado el 7 de Enero de 2012, en http://en.wikipedia.org/wiki/MySQL

Wikipedia. (10 de Noviembre de 2011). Sass (stylesheet language). Consultado el 12 de Diciembre de 2011, en http://en.wikipedia.org/wiki/Sass_(stylesheet_language)

Wikipedia. Scrum (development). Consultado el 10 de Enero de 2012, en http://en.wikipedia.org/wiki/Scrum_(development)

Wikipedia. Server- side. Consultado el 2011 de Diciembre de 2011, en http://en.wikipedia.org/wiki/Server-side

Wikipedia. TOEFL. Consultado el 12 de Enero de 2012, en http://es.wikipedia.org/wiki/TOEFL

Wikipedia. World Wide Web. Consultado el 12 de 12 de 2011, en http://es.wikipedia.org/wiki/World_Wide_Web