UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

FACULTAD 3

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Herramienta para el diseño y generación de formularios web para la Ventanilla Única Aduanera.

Autor(es):

Osniel Ramos Avalos

Oslen Alvarez Carmona

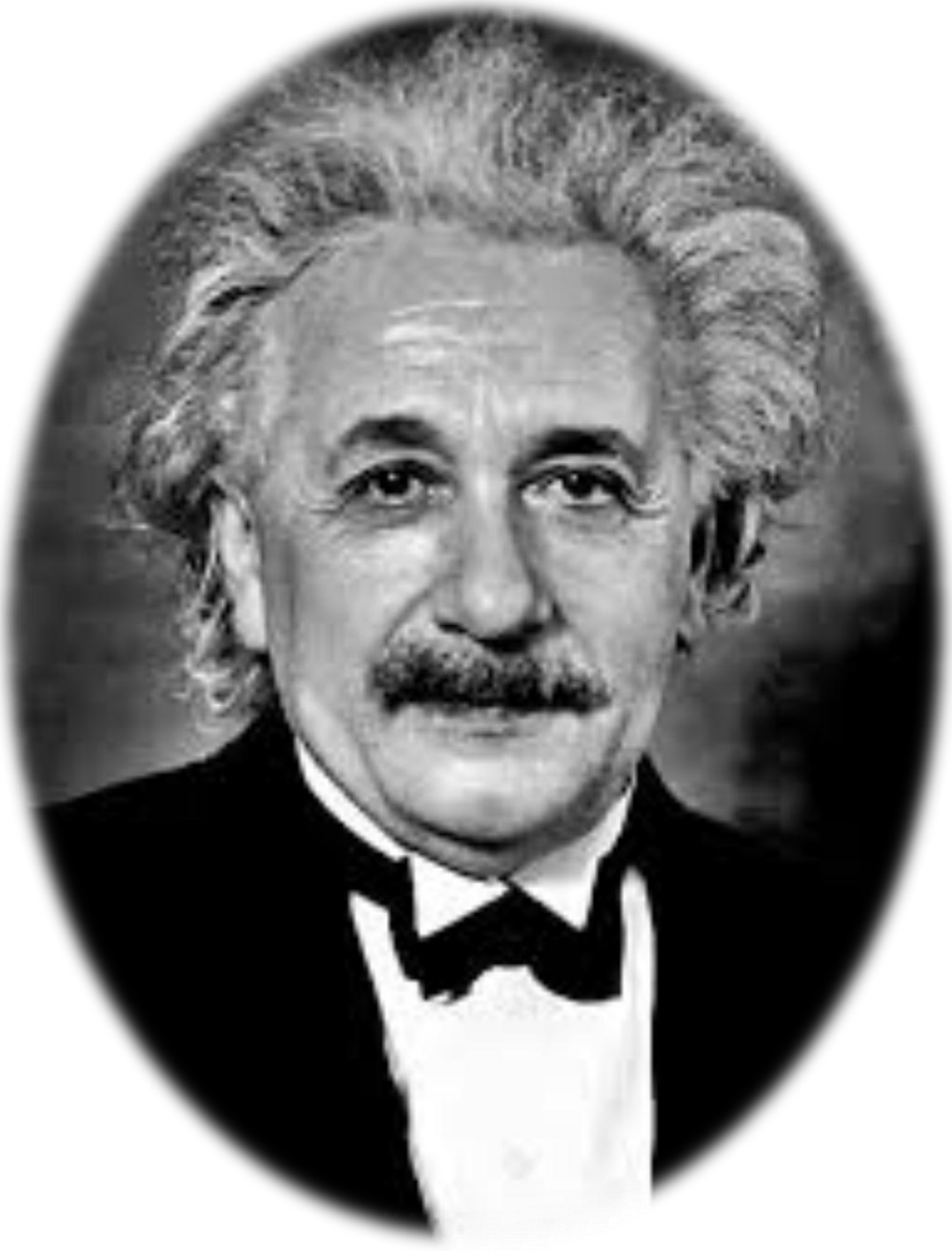
Tutor(es):

Ing. Leduan Flores Riera

Ing. Leansi Vega Rouco

La Habana, junio de 2018

"Año 60 de la Revolución"



"Cada día sabemos más y entendemos menos".

Albert Einstein

DECLARACIÓN DE AUTORÍA

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

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

_____	_____
Osniel Ramos Avalos	**Oslen Alvarez Carmona**
Autor	Autor
_____	_____
Ing. Leansi Vega Rouco	**Ing. Leduan Flores Riera**
Tutor	Tutor

A toda mi familia, en especial…

A mis padres, mi hermano, mi tío y mi abuelo que sin ellos no estuviera aquí hoy.

Osniel Ramos Avalos.

A toda mi familia, en especial.…

A mi abuela, mis padres, mis hermanos y mis tíos que sin ellos esto no fuera posible.

Oslen Alvarez Carmona.

Quiero agradecer a mis amistades, a los del aula, a los que vinieron desde tan lejos a compartir este día tan especial conmigo y a los que no pudieron venir, pero igual se acordaron de mí.

A toda mi familia por su apoyo incondicional, pero sobre todo quiero agradecer…

A mis padres, por estar ahí siempre para mí y cumplir mis caprichos aun cuando no los merezco.

A mi tío Duniesky por darme todo lo que le he pedido sin pensarlo, y lo que no le he pedido también, por ser un ejemplo a seguir, por sus consejos… aunque no tenga mucha paciencia. Duni tal vez no lo sepas porque no lo demuestro como mi hermano, pero siempre estaré ahí para ti. Muchas Gracias tío.

A mi hermano, por ser un apoyo y la única persona en la que confío, por enseñarme a ser más "pillo" en la vida, One sabes que siempre, por mucho que discutamos estaré ahí para ti, sin importar cuándo, dónde, cómo, o que obstáculos existan. Muchas Gracias mi hermano.

A mi papá por ser un ejemplo a seguir, papá lo único que pienso cuando me propongo una meta en la vida, es; "ojalá pueda hacerlo como lo haría mi

papá", sin importa que sea, porque lo mejor que tienes es que no tienes límites, para ti no hay trabajo que no se pueda hacer, tal vez nunca te hayas dado cuenta, pero siempre que haces algo intento prestar atención para aprender porque para mí nadie tiene ese talento, esa dedicación, esa perfección que tienes cuando te propones algo. Por esto, por enseñarme tanto papá, muchas Gracias.

A mi mamá por ser tan especial, mamá quiero que sepas que estos años lejos de ustedes no han sido fáciles, no han sido cómo crees, te he mentido, te he ocultado cosas, siempre pensando que hago bien, pero como ya sabes aun después de tantos años no soy nada sin tu apoyo y tus consejos y siempre termino contándote mis problemas, la mayoría de las veces porque no me queda más remedio, pero tal vez es porque sepa que no importa cuando, ni porque, ni que tan grave sea, siempre ustedes estarán conmigo… aunque tengas que venir desde Villa Clara a las 11:00 pm tu solita. Por estas cosas mamá quisiera que fueras eterna, muchas Gracias.

A mis padres más recientes, gracias por hacerme sentir como un hijo para ustedes, por darme tanta confianza, por creer en mí, no los defraudaré nunca y siempre cuidaré a su princesa más que a mi vida.

A mi tata por hacerme la persona más feliz del mundo. Por estar a mi lado en los momentos más difíciles, en los momentos en los que no quiero saber de nadie más. Creo que ya lo sabes, pero lo diré porque sé que te gusta oírlo, una sola sonrisa tuya me da las energías que necesito para seguir adelante en cualquier situación. Tata quiero que sepas que estos meses a tu lado los siento como si fueran años por la confianza que me brindas y la que me haces sentir. Contigo a mi lado siento que no hay nada que no pueda lograr. Te amo mi angelito adorable, y por tu amor, muchas Gracias.

Osniel Ramos Avalos.

Quiero agradecer a mi abuela (nene) que ya no se encuentra entre nosotros pero que siempre la llevo conmigo. Gracias a ella soy la persona que soy hoy en día. Sé que estaría muy orgullosa de verme aquí. Estás palabras no alcanzan para agradecerle todo lo que hiso por mí. Muchas Gracias.

A mi mamá que es la persona más especial de mi vida. Gracias por siempre estar en las buenas y en las malas. Por ayudarme a que siempre que me caigo levantarme e impulsarme a seguir adelante. Por enseñarme hacer una mejor persona cada día. Mami te amo, muchas gracias.

A mi papá por ser la persona que es y ser mi guía en la vida. Por siempre sacarme de todos mis problemas en los cuales me he metido. Por enseñarme hacer un hombre de bien. Aunque no te lo diga mucho te quiero. Muchas gracias.

A mi hermano Osmel por estar siempre junto a mi lado tanto en las buenas como en las malas. Por tener la paciencia de siempre escucharme y de aconsejarme. Quiero que sepas que eres la personita que más quiero en mi vida y que puedes contar conmigo para lo que sea. Muchas Gracias mi hermanito.

A mi tío Danilo por ser como un padre para mí y dame la confianza de un hijo más. Por tapar y escuchar todas mis trastadas. Por aconsejarme y regañarme cuando lo he necesitado. Sabes que te quiero. Muchas Gracias.

A mi segunda madre, mi tía Virginia por haberme acogido como el hijo varón que no tiene. Te agradezco todo lo que has hecho por mí. Te quiero mucho. Muchas Gracias.

A Mis tíos Isidoro y Martin que, aunque estén lejos siempre sacan un tiempo para escucharme y aconsejarme. Por nunca decirme un no y ayudarme en todo lo que he necesitado. Muchas Gracias.

A mi tata, la hermanita que no tengo y que quiero con locura, aunque sea muy histérica. Sabes que te quiero. Muchas Gracias.

A mi compañero de tesis que en los últimos años ha sido como un hermano. Muchas Gracias.

A mis amigos Adonito, Pedro, Alexito, Copita y Pocholo que siempre han estado a mi lado y que saben que los quiero como si fueran mis hermanos.

Gracias a Eileen y a Lalo por siempre ayudarme y aguantarme. Saben que las quiero con locura, aunque sean muy estresantes. Muchas Gracias.

Muchas gracias a mis hermanos Osley y Osniel, a todos mis tíos y primos que no mencione pero que no olvido. A todos ellos muchísimas gracias. Los quiero a todos.

Oslen Alvarez Carmona.

En la actualidad resulta vital para los desarrolladores de software contar con herramientas que sean capaces de diseñar y generar formularios web, pues sin este tipo de tecnologías, la implementación resultaría muy compleja, pudiendo afectar el tiempo final de entrega de un proyecto. En el presente trabajo se pretende desarrollar una herramienta para diseñar y generar de formularios web para el sistema Ventanilla Única Aduanera (VUA), centro vinculado a la Facultad 3 de la Universidad de las Ciencias Informáticas (UCI). El software de diseño y generación de formularios web surge como respuesta a la necesidad de los desarrolladores de disminuir el tiempo de implementación de los formularios. Como base de este proceso, previamente fue realizado un amplio estudio de los principales conceptos, lenguajes, herramientas y metodologías implicados en el negocio. Después de aplicar patrones, métricas y distintas pruebas de validación y aceptación del software se obtuvo como resultado una herramienta que contribuye con la aceleración y automatización del diseño y la generación de formularios web, disminuyendo así el tiempo de implementación de los mismos por los desarrolladores.

Palabras claves:

Diseñador, generador, formularios web, VUA.

ABSTRACT

Currently it is vital for software developers to have tools that are capable of designing and generating web forms, because without this type of technology, the implementation would be very complex, and may affect the final delivery time of a project. In the present work we intend to develop a Web Form Designer and Generator for the Single Window System, a center linked to the Faculty 3 of the University of Informatics Sciences. The design and generation of web forms software arises in response to the need of developers to reduce the time of implementation of the forms. As a basis for this process, a broad study of the main concepts, languages, tools and methodologies involved in the business was previously carried out. After applying patterns, metrics and different software validation and acceptance tests, a tool was obtained that contributes to the acceleration and automation of the design and generation of web forms, thus decreasing the implementation time of the forms by the developers.

Keywords:

Designer, generator, web forms.

ÍNDICE

ÍNDICE DE TABLAS

ÍNDICE DE ILUSTRACIONES

INTRODUCCIÓN

Las Tecnologías de la Información y las Comunicaciones (TIC) son un concepto muy asociado al de informática. Si se entiende esta última como el conjunto de recursos, procedimientos y técnicas usadas en el procesamiento, almacenamiento y transmisión de información (1). Las TIC son aplicadas en áreas como la informática y las telecomunicaciones, con una importancia cada vez mayor en los procesos que se desarrollan en la sociedad y en la economía. Estas tecnologías están presentes en áreas muy importantes a nivel empresarial, posibilitando que los procesos de negocio sean más eficientes y rápidos. Con las TIC surgen sistemas informáticos que facilitan la gestión de la información en las empresas.

Los sistemas informáticos son programas de desarrollo tecnológico que permiten a los clientes consumir servicios de una empresa y realizar consultas desde sus computadoras. Dichos sistemas son creados para mejorar las características productivas de una entidad y de esta manera poder gestionar y explotar la información de las instituciones para las cuales han sido creadas (2). También presentan dentro de sus funciones particulares un modelo de captura de datos eficiente.

La captura de datos puede tornarse en ocasiones muy compleja cuando no se dispone de herramientas eficaces que logren gestionar la tarea sin demoras, proporcionándole al usuario un producto final donde no se pueda apreciar lo que en realidad él necesita. Para realizar dicha tarea son utilizados los formularios, por lo que existen disímiles herramientas encargadas de realizar dentro de sus procesos la captura de datos utilizando los mismos. Para facilitar y agilizar el proceso de diseño y generación de los formularios web en la actualidad, se han desarrollado los generadores de formularios web. Estos dan la oportunidad al desarrollador de diseñar, y generar los formularios web automáticamente.

En la actualidad, resulta vital para los desarrolladores de las empresas de software, contar con componentes que sean capaces de generar formularios web. Sin este tipo de tecnologías, la implementación resultaría compleja y podría comprometer el tiempo de finalización de un proyecto. Después de consultar un conjunto de bibliografías (3), (4), (5), (6), (7), (8), se pudo constatar que en el mundo existen diversos generadores de formularios web como, por ejemplo: Form Builder, DesignExpert, FormLogix, Wufoo, Formspring; pero estos no responden a las necesidades de la VUA.

En el sistema VUA, los formularios como forma de captura de información son muy utilizados, presentando la cara del sistema e interactuando de forma fácil con usuarios no identificados rigurosamente con la informática. Los desarrolladores de VUA necesitan implementar formularios web para la gestión de la información recibida a través de los usuarios. Estos formularios pueden llegar a ser más de veinte y pueden tener hasta más de diez componentes cada

uno, por lo que se hace engorroso su diseño e implementación, pudiendo llevar consigo una demora en el tiempo de finalización del sistema.

Luego de analizada la problemática existente se define como **problema a resolver**:

¿Cómo generar formularios web disminuyendo el tiempo estimado para la implementación de requisitos funcionales por un desarrollador en el sistema VUA?

Se plantea como **objeto de estudio** el proceso de desarrollo de las aplicaciones web, centrado en el **campo acción:** Proceso de desarrollo de formularios web.

Para darle solución al problema de investigación se plantea como **objetivo general:**

Desarrollar una herramienta para el diseño y generación de formularios web que disminuya el tiempo estimado de implementación de requisitos funcionales por un desarrollador en el sistema VUA; y este se desglosa en los siguientes **objetivos específicos:**

1 Construir el marco teórico de la investigación relacionado con las herramientas para diseñar y generar formularios web, así como modelos, metodologías, herramientas y lenguajes para el desarrollo de software.

2 Desarrollar los artefactos ingenieriles para la obtención de la herramienta informática.

3 Implementar una herramienta web que posibilite el diseño y la generación de formularios web.

4 Validar la herramienta web a partir de su utilización por los desarrolladores del sistema VUA.

Se propone como **idea a defender** de la investigación: Si se desarrolla una herramienta para el diseño y generación de formularios web, entonces se disminuye el tiempo estimado en la implementación de requisitos funcionales que requieran de su utilización por un desarrollador.

Con el objetivo de resolver el problema y lograr el objetivo planteado, se hizo necesaria la utilización de los siguientes **métodos de investigación**:

Métodos Teóricos:

- Se utiliza el método **Analítico-Sintético:** se realizó un estudio de los requerimientos de la aplicación permitiendo la extracción de los elementos más importantes que se relacionan con el objeto de estudio. Se analizaron cada uno de los conceptos, que de una forma u otra guardan relación con el objeto de estudio de la investigación y las relaciones existentes entre ellos. Con los resultados de este

análisis se realiza una síntesis para comprender las características del objeto de estudio y sus relaciones. Este método permitió definir el marco teórico de la investigación propuesto en el *Capítulo 1*.

- Se utiliza el método **Histórico-Lógico:** partiendo de la observación y análisis de los antecedentes del objeto de estudio se dio inicio a esta investigación y se realizará un estudio de la existencia de otros sistemas de similar funcionamiento, proyectando el análisis de la evolución histórica de los fenómenos, con la proyección lógica de su comportamiento futuro. Se empleó para analizar la trayectoria y evolución de los modelos y herramientas existentes para el diseño y generación de formularios web.

Métodos Empíricos:

- **Encuestas** y **Entrevistas:** se realizaron entrevistas a los desarrolladores del sistema, para generalizar los componentes que deben necesitarse en la aplicación.

- **Experimentación**: se realizó por su importancia decisiva en las pruebas para demostrar el funcionamiento del sistema.

El presente documento está conformado por tres capítulos estructurados de la siguiente manera:

Capítulo 1: Fundamentación teórica.

En este capítulo se realiza una revisión y análisis de los modelos, métodos, metodologías y herramientas existentes para el diseño y la generación de formularios web. Se enuncian los principales conceptos relacionados con el tema, así como un estudio de los lenguajes, las herramientas y los patrones a usar como apoyo para dar solución al problema planteado.

Capítulo 2: Descripción de la propuesta.

En este capítulo se describe la propuesta de la investigación, mostrando la especificación de los requisitos funcionales y no funcionales, definiéndose las historias de usuarios del sistema, se confecciona un plan de entrega en conjunto con un plan de iteraciones. Aborda los detalles relacionados con el diseño del generador de formularios web.

Capítulo 3: Implementación, pruebas y validación.

En este capítulo se abordan los elementos y estándares de codificación y la implementación, utilizando el lenguaje de programación y la metodología seleccionada. Son presentados los resultados de aplicar pruebas al sistema. Se valida la investigación realizada mediante un pre-experimento.

CAPÍTULO 1. FUNDAMENTACIÓN TEÓRICA

1.1 Introducción

En este capítulo se abordan los principales conceptos utilizados en la investigación. Se hace un estudio de diferentes herramientas existentes para el diseño y la generación de formularios web. Se realiza una explicación de la filosofía de trabajo que ofrecen las mismas y sus principales ventajas. Se describen las principales tecnologías, lenguajes de programación y herramientas definidas para el desarrollo de la solución, así como la metodología de desarrollo, los patrones a emplear, las métricas para la validación del diseño y las pruebas para garantizar la calidad del producto final.

1.2 Marco teórico

Las aplicaciones web son muy utilizadas por los usuarios debido a las ventajas que brindan. En la ingeniería de software se denomina aplicación web a aquellas herramientas que los usuarios pueden utilizar accediendo a un servidor web a través de internet o de una intranet mediante un navegador. En otras palabras, es un programa que se codifica en un lenguaje interpretable por los navegadores web en la que se confía la ejecución al navegador (5).

Otra definición que existe es la que brinda el grupo de (6): una aplicación web, es una herramienta informática distribuida cuya interfaz de usuario es accesible desde un cliente web, normalmente un navegador web. Esta definición será la adoptada, debido a que es una síntesis actualizada de la mencionada anteriormente.

Como parte del proceso de desarrollo de las aplicaciones web, en el caso de las aplicaciones que mantienen interacción directa con los usuarios, es imprescindible el uso de formularios. Por la importancia que representan y el uso que se les puede dar, es necesario un correcto diseño de los mismos. La mala concepción de esta función puede situarse como una barrera para la interacción e inducir al usuario a cometer errores potenciados por la frustración en el desempeño de sus tareas.

El diseño se ha descrito como un proceso multifase en el que se sintetizan representaciones de la estructura de datos, la estructura del programa, las características de la interfaz y los detalles procedimentales desde los requisitos de la información (7).

El concepto de diseño se aplica a diferentes acepciones en la informática, está el caso del diseño web que es el arte de planificar, diseñar e implementar sitios web (8).

En el diseño web, se aprecia el diseño de formularios y para hacer la tarea más rápida existen sistemas diseñadores de formularios. Según los autores (9) quienes definen, Diseñador de Formulario en su libro

Diccionario de Informática como “espacio de trabajo que reservan algunos entornos de desarrollo de los lenguajes de programación visual donde despliegan un formulario por defecto para la creación de la interfaz del usuario”. En este espacio denominado “diseñador del formulario”, es donde el programador inserta los controles que formarán la interfaz de usuario.

Los diseñadores y generadores de formularios web son herramientas que ayudan exponencialmente al desarrollo de formularios web. Estos garantizan que el usuario pueda crearlos sin poseer un nivel avanzado de lenguajes de programación, evitando así que se encuentre todo el tiempo implementando una herramienta de captura de datos a su medida. Entre las ventajas que presentan los diseñadores y generadores de formularios se pueden encontrar (3):

- Realizan la mayor parte de trabajo.
- Se pueden adaptar fácilmente a necesidades específicas.
- Se producen resultados más rápidos, sin efectos sobre la calidad del producto.

1.3 Estado del arte

Se realiza un análisis de herramientas para el diseño y la generación de formularios web. Como parte del estudio se explican sus principales características y diferencias.

1.3.1 Herramientas para el diseño y generación de formularios web en la actualidad

El estudio de los sistemas diseñadores y generadores de formularios web existentes facilita la toma de decisiones. Permite determinar cuál de las herramientas usar o cuáles de los sistemas se adecuan a las necesidades presentadas por los clientes. Cada una de las herramientas que se describen a continuación son utilizadas por las particularidades que implementan y los servicios que brindan. Existen varias herramientas dedicadas al diseño y generación de formularios web, ver Tabla 1.

Las herramientas seleccionadas para el análisis se explican a continuación:

Form Builder

Puede ser utilizado en cualquier navegador de Internet y permite construir fácilmente formularios web seguros. Es un software con licencia privada que entre sus características se incluyen: interfaz amigable y construcción con un solo clic. Es compatible con algunos navegadores como Firefox, Opera Internet Explorer y Chrome. Presenta la capacidad de desarrollar formularios básicos y complejos de forma fácil y rápida. Esta herramienta solo es compatible con el sistema operativo Windows (3), (4).

DesignExpert

El software de diseño de formularios DesignExpert permite a usuarios de formularios escaneables diseñar formularios para reproducción de lectura por marcas ópticas (OMR) o aplicaciones de proceso de formularios por imágenes. Dicho software logra crear y personalizar formularios que se ajusten a necesidades específicas e imprimir los formularios en el sitio del usuario o en el de la empresa productora del mismo. Esta herramienta no presenta una interfaz muy amigable y solo soporta los navegadores Firefox y Chrome. Todos los bloques de construcción básicos de páginas web, tablas y marcos están diseñados, junto con la navegación y el control (3), (6).

FormLogix

Aplicación que permite crear formularios online, sin necesidad de saber nada de programación. Permite crear los formularios usando una interfaz 2.0 de alta calidad. Presenta compatibilidad con algunos navegadores como Firefox, Internet Explorer, Opera, Safari y Chrome. A pesar de ser una herramienta muy potente en la web, debemos destacar que tiene limitaciones, pues no es totalmente gratis, la versión gratuita después de pasado algún tiempo anula los formularios enlazados y se eliminan las cajas de herramientas para administración de información. Puede ser utilizado en Windows y algunas versiones de Linux (3), (5).

Wufoo

Herramienta para confeccionar formularios online. Tiene la particularidad que al diseñar un formulario se crea automáticamente la base de datos, y las herramientas necesarias para almacenar, gestionar y entender la información recogida. Wufoo proporciona las herramientas para poder crear fácilmente formularios personalizados con una interfaz intuitiva de arrastrar y soltar. Cuenta con un plan gratuito en el que se puede crear hasta tres formularios. Si se desea tener un mayor número de formularios y características avanzadas se deberá acceder a uno de sus planes de pago. Esta herramienta solo trabaja en la plataforma Windows. Soporta casi todos los navegadores, a excepción de safari y opera (3), (7).

Formspring

Herramienta que permite generar formularios online, que se puede insertar en los sitios web. Cuenta con una serie de planes, siendo algunos de los mismos gratis. Con Formspring se pueden generar formularios con la utilización de herramientas que permiten realizar diversas tareas como insertar campos personalizados, editarlos y ordenarlos. Presenta compatibilidad con los navegadores Firefox, Internet Explorer, Opera, Safari y Chrome. Plataforma en la que trabaja, Windows (3), (8).

A continuación, se presentan las características de estas herramientas en una tabla calificadas en dos criterios (Si, No), de acuerdo a la necesidad en el sistema VUA. Significando el criterio “Si”, que la herramienta cumple con los requisitos para ser utilizada en sistema VUA. En el caso de ser el criterio “No”, ocurre todo lo contrario, o sea, la herramienta no cumple con los requisitos para ser utilizada en sistema.

Ejemplo: en el caso de la Licencia, cumple con los requisitos del sistema VUA si es gratuita, en caso de ser una licencia privada no cumple con los requisitos del sistema VUA, puesto que en el sistema se busca una solución que no incluya más gastos.

Tabla 1: Herramientas para el diseño y generación de formularios web.

Herramienta	Form Builder	DesignExpert	FormLogix	Wufoo	Formspring
Licencia	No	No	Si	No	Si
Conectividad a Internet	No	No	No	No	No
Integra con Symfony	No	No	No	No	No
Usabilidad	Si	No	Si	Si	No
Navegadores	Si	No	Si	No	No
Plataforma	No	No	Si	No	No

Estas herramientas son muy eficientes atendiendo a las particularidades que cada una implementa en sus funcionalidades, pero no cumplen con el principal requisito deseado (integración con el marco de trabajo symfony) para ser utilizadas por los desarrolladores del sistema VUA para la creación de los formularios web. Los desarrolladores del sistema VUA no pueden utilizar estas herramientas debido a que necesitan estar todo el tiempo conectados a internet para poder utilizarla. Algunas de estas herramientas poseen planes gratuitos pero las funcionalidades que necesita utilizar el sistema VUA son de pago, lo que implicaría tener un gasto adicional en el sistema. Aunque estas herramientas de forma general no satisfacen las necesidades de los desarrolladores del sistema, cabe destacar que se tuvieron en cuenta algunos aspectos positivos como son:

1 Interfaz amigable

2 Compatibilidad con varios navegadores web

3 Compatibilidad con varias plataformas

4 Diseño “drag and drop” o “arrastrar y soltar” en la conformación del formulario

1.4 Metodología de desarrollo de software

El desarrollo de software puede ser un reto para los equipos de proyecto, pues crear un software en el menor tiempo posible y con calidad puede ser complicado, si no se cuenta con algún proceso que ayude a agilizar el desarrollo de software.

Las metodologías de desarrollo imponen un proceso disciplinado sobre el desarrollo de software con el fin de hacerlo eficiente. Se define como metodología a emplear la misma que se utiliza en el sistema VUA, una variación de AUP (Agile Unified Process, por sus siglas en inglés) apoyándose en el modelo CMMI-DEV V1.3 (14), que se adapta a la actividad productiva de la Universidad de las Ciencias Informáticas.

Esta metodología describe tres fases, las cuales son: Inicio, Ejecución y Cierre. La investigación se centra en la fase de Ejecución en la cual se ejecutan las actividades requeridas para el desarrollo del software, se obtienen los requisitos, se elabora la arquitectura y el diseño, se implementa y se libera el producto. Las disciplinas que se emplearán son: Requisitos, Análisis y diseño, Implementación, Pruebas internas y Pruebas de aceptación. La metodología para desarrollar la solución propone 4 escenarios. Se estará haciendo uso del escenario 4, el cual utiliza Historias de Usuario (HU) como técnica de encapsulación de requisitos. Este escenario se aplica dadas las características del software a desarrollar. Estas características son:

El proyecto no es extenso, el equipo de desarrollo es de dos personas.

El cliente, que son los desarrolladores del sistema VUA está relacionado con el equipo de desarrollo para convenir los detalles de los requisitos y así poder implementarlos, probarlos y validarlos.

En cada iteración se repite el flujo de trabajo de las disciplinas, Requisitos, Análisis y diseño, Implementación y Pruebas internas. De manera creciente y escalonada se brindan los resultados del producto final. Para llegar a lograr esto, cada requisito debe tener un completo desarrollo en una única iteración.

1.5 Herramientas, tecnologías y lenguajes.

Se seleccionó como marco de trabajo Symfony 2.8.4 pues además de ser el framework utilizado en el sistema VUA, el equipo de trabajo cuenta con experiencia en el desarrollo de aplicaciones usando este marco de trabajo PHP. Como Entorno de Desarrollo Integrado se seleccionó JetBrains PhpStorm 9.0.2 ya que es un

IDE que tiene muy buena integración con el marco de trabajo Symfony 2.8.4, permitiendo la ejecución de comandos y auto- completamiento de código. A continuación, se describen estas herramientas, tecnologías y lenguajes seleccionados:

1.5.1 Herramienta CASE

Las herramientas CASE (Ingeniería de Software Asistida por Computadora) son aplicaciones informáticas dedicadas al aumento de la productividad en el desarrollo de software. Estas herramientas pueden ayudar en todos los aspectos del ciclo de vida de desarrollo del software en tareas como el proceso de realizar un diseño del proyecto, cálculo de costos, implementación de parte del código automáticamente con el diseño dado, compilación automática, documentación o detección de errores entre otras (11).

Visual Paradigm 8.0

Es una herramienta CASE, desarrollada por la compañía Visual Paradigm International, que utiliza UML como lenguaje de modelado. Presenta las siguientes características (12):

1 Soporta el ciclo de vida completo del software: análisis, diseño, implementación y despliegue.

2 Permite la captura de requisitos, el dibujo de diagramas UML, la realización de ingeniería inversa y generación de código PHP.

Se integra con las siguientes herramientas:

- Eclipse/IBM Web Sphere
- Builder
- Net Beans IDE
- Oracle JDeveloper
- BEA Weblogic

Está disponible en varias ediciones, cada una destinada a necesidades específicas: Enterprise, Professional, Community, Standard, Modeler y Personal.

Se decide utilizar como herramienta CASE el visual paradigm, debido a su alta capacidad de integración con lenguajes de programación. Es una herramienta multiplataforma, de código abierto, disponible en varios idiomas además de ser fácil de instalar y actualizar. Visual paradigm facilita la organización de los diagramas

generando la documentación del proyecto en varios formatos. Además de ser capaz de generar código, modelo de datos, entre otras funcionalidades útiles para el desarrollo de un producto.

1.5.2 Entorno de Desarrollo Integrado

Un Entorno de Desarrollo Integrado, por sus siglas en inglés (IDE) es un programa compuesto por una serie de herramientas utilizadas por programadores para desarrollar aplicaciones.

JetBrains PhpStorm 9.0.2

Es un imponente entorno de desarrollo integrado (IDE) que está especialmente desarrollado para los desarrolladores web para la edición de PHP, CSS[1], HTML, XML y archivos de JavaScript. Esta aplicación permitirá a los usuarios crear, así como cambiar el código fuente, independientemente del lenguaje de programación que se utiliza. Al igual que cualquier otro editor IDE[2], JetBrains PhpStorm también tiene todas las características básicas que incluye zoom, puntos de interrupción y marcadores (13).

Además de estas características, incluye análisis de código, navegación rápida y macros para hacer su trabajo fácil. JetBrains PhpStorm tiene una interfaz intuitiva y amigable con menús muy bien organizados y puede navegar rápidamente la clase específica y la región personalizada mediante el menú Navegar. También tiene el menú Ejecutar que te permitirá ejecutar secuencias de comandos en poco tiempo. Es una aplicación que permite trabajar con archivos PHP, HTML y CSS en un entorno muy amigable.

Características:

- Desarrollado para desarrolladores web para editar archivos PHP, CSS, HTML y JavaScript.
- Permite a los usuarios crear y cambiar el código fuente.
- Tiene todas las características básicas como zoom, marcadores y puntos de interrupción.
- Incluye análisis de código, macros y navegación rápida.
- Tiene una interfaz intuitiva y amigable.
- Puede navegar rápidamente el símbolo, la clase específica y la región personalizada mediante el menú Navegar.

[1] Cascading Stylesheets
[2] Integrated Development Environment

- Puede ejecutar scripts desde el menú Ejecutar en poco tiempo.

1.5.3 Marco de trabajo

Symfony 2.8.4

Es un marco de trabajo diseñado para optimizar el desarrollo de las aplicaciones web basado en el patrón Modelo Vista Controlador. Separa la lógica de negocio, la lógica de servidor y la presentación de la aplicación web. Proporciona varias herramientas y clases encaminadas a reducir el tiempo de desarrollo de una aplicación web compleja. Además, automatiza las tareas más comunes, permitiendo al desarrollador dedicarse por completo a los aspectos específicos de cada aplicación (14).

Symfony está desarrollado completamente en PHP[3]5.3. Ha sido probado en numerosos proyectos reales y se utiliza en sitios web de comercio electrónico de primer nivel. Se puede ejecutar tanto en plataformas *nix (Unix, Linux, etc.) como en plataformas Windows. Este marco de trabajo presenta las siguientes características (15):

Características

- Permite la internacionalización para la traducción del texto de la interfaz, los datos y el contenido de localización.
- La presentación usa templates y layouts que pueden ser construidos por diseñadores de HTML [4]que no posean conocimientos del framework.
- Los formularios soportan la validación automática, lo cual asegura mejor calidad de los datos en las bases de datos y una mejor experiencia para el usuario.
- El manejo de cache reduce el uso de banda ancha y la carga del servidor.
- La facilidad de soportar autenticación y credenciales facilita la creación de áreas restringidas y manejo de seguridad de los usuarios.
- El enrutamiento y las URLs [5]inteligentes hacen amigable las direcciones de las páginas de la aplicación.

[3] Hypertext Preprocessor
[4] HyperText Markup Language
[5] Uniform Resource Locator

- Las listas son más amigables, ya que permite la paginación, clasificación y filtraje automáticos.
- Los plugins proveen un alto nivel de extensibilidad.

Bootstrap 4.0.0

Desarrollado como un marco de trabajo (framework), para fomentar la consistencia entre las herramientas internas. Es un framework web de código abierto para diseño de sitios y aplicaciones web. Contiene plantillas de diseño con tipografía, formularios, botones, cuadros, menús de navegación y otros elementos de diseño basado en HTML y CSS, así como extensiones de JavaScript adicionales. A diferencia de muchos frameworks web, solo se ocupa del desarrollo front-end. Desde la versión 2.0 también soporta diseños web adaptables. Esto significa que el diseño gráfico de la página se ajusta dinámicamente, tomando en cuenta las características del dispositivo usado (Computadoras, tabletas, teléfonos móviles) (20) (21).

1.5.4 Servidor web

Generalmente, un usuario o navegador, solicita una página web cada vez que se conecta a Internet. El servidor se encarga de responder a esta “petición” o “demanda”, y manda el contenido solicitado por el usuario. Así pues, existen diferentes tipos de servidores HTTP, de entre ellos, algunos de los más popularmente conocidos y utilizados son: el servidor Apache, Microsoft IIS, NGinx, Wamp y Xampp (16).

Servidor web Xampp 3.2.2.

X (Para cualquier sistema operativo), A (Apache), M (MySql), P (Php), P (Perl), es un servidor independiente de plataforma libre que integra en una sola aplicación un servidor web Apache, intérpretes de lenguaje de scripts PHP, un servidor de base de datos MySQL. Permite instalar de forma sencilla Apache en el ordenador, sin importar el sistema operativo (Linux, Windows, MAC o Solaris).

A continuación se muestran algunas características de Xampp (21)**:**

- Para Windows existen dos versiones, una con instalador y otra portable para descomprimir y ejecutar.
- La licencia de esta aplicación es GNU (General Public License), está orientada a proteger la libre distribución, modificación y uso de software.
- Es multiplataforma, es decir existen versiones para diferentes sistemas operativos
- Xampp se actualiza regularmente para incorporar las últimas versiones de Apache/MySQL/PHP y Perl. También incluye otros módulos como Open SSL y PHPMyAdmin.

Se selecciona como servidor web Xampp en su versión 3.2.2 por ser multiplataforma, cuenta con abundante documentación y soporta varios lenguajes de programación, dentro de los cuales se encuentra el seleccionado para la solución.

1.5.5 Navegador web.

A continuación, se menciona el navegador web Mozilla Firefox, puesto que es el utilizado en el desarrollo del software para visualizar el resultado a medida que se implementan las funcionalidades. El software será compatible con otros navegadores, como Chrome, Internet Explorer, Opera, Safari.

Mozilla Firefox

Es un navegador web libre y de código abierto desarrollado para Linux, Android, IOS, OS X y Microsoft Windows coordinado por la Corporación Mozilla y la Fundación Mozilla. Usa el motor Gecko para renderizar páginas web, el cual implementa actuales y futuros estándares web. Está basado en la interfaz Starta. El enfoque consiste en un menú unificado (no disponible en OS X) con las pestañas de navegación hacia arriba, y la barra de direcciones con los botones de acción debajo. Es compatible con varios lenguajes web, incluyendo HTML, XML, XHTML, SVG 1.1 (parcial), CSS 1, 2 y 3, ECMAScript (JavaScript), DOM, MathML, DTD, XSLT, XPath, e imágenes PNG con transparencia alfa. También incorpora las normas propuestas por el WHATWG, y es compatible con el elemento HTML Canvas. Entre sus características incluyen (18):

- Navegación por pestañas
- Corrector ortográfico (que puede ser incluido vía Mozilla Addons - Complementos de Mozilla)
- Búsqueda progresiva
- Marcadores dinámicos
- Administrador de descargas
- Lector RSS,
- Navegación privada
- Navegación con georreferenciación
- Aceleración mediante GPU e integración del motor de búsqueda que desee el usuario

- Se puede instalar tanto sin conexión como también en línea desde la página web, este último es utilizado para descargar los componentes de segundo plano, ideal para equipos con conexiones mínimas.

1.5.6 Lenguajes de programación

Lenguaje Unificado de Modelado

El Lenguaje Unificado de Modelado (Unified Modeling Language, UML) es un lenguaje de modelado visual usado para especificar, visualizar, construir y documentar artefactos de un software. Capta la información sobre la estructura estática y el comportamiento dinámico de un sistema. Brinda apoyo a la mayoría de los procesos de desarrollo orientados a objetos (19).

De acuerdo a (19) UML es un lenguaje que permite modelar, construir y documentar los elementos que forman un sistema orientado a objetos. Dentro de los principales beneficios de utilizar UML se encuentran:

- Mejores tiempos totales de desarrollo (de 50% o más).
- Modelar sistemas (y no sólo de software) utilizando conceptos orientados a objetos.
- Es un lenguaje de modelado utilizado tanto por humanos como por máquinas.

Se emplea el lenguaje de modelado UML en su versión 2.0 para la confección del diagrama de componentes y del modelo de datos de la solución.

Lenguaje JavaScript

Se utiliza el lenguaje de programación JavaScript en su versión 2.5. Este lenguaje se implementó por primera vez en la versión beta de Netscape Navigator 2.0 en junio de 1995, por la empresa Netscape Communications Corporation. Es un lenguaje de programación interpretado utilizado en la realización de páginas Web. La verdadera fuerza de JavaScript es la compatibilidad con los distintos navegadores. Además de que es interpretado por todos los navegadores modernos (20).

Las principales características de JavaScript como lenguaje son (20):

- Lenguaje interpretado.
- Estructurado: provee todos los recursos de un lenguaje estructurado como C (funciones, ciclo por conteo y por condición, condicionales, condicionales múltiples, y demás estructuras).

- Tipado Dinámico: como la mayoría de los lenguajes scripts los tipos de las variables están asociados al valor que contiene en un momento dado.

- Orientado a Objetos: implementa una variante del orientado a objetos no basada en clases propiamente sino en objetos prototipos a través de los cuales se pueden crear otros objetos idénticos y extender sus funcionalidades.

JQuery es una biblioteca multiplataforma de JavaScript, que permite simplificar la manera de interactuar con los documentos HTML, manipular el árbol DOM6, manejar eventos, desarrollar animaciones y agregar interacción con la técnica AJAX a páginas web. jQuery es la biblioteca de JavaScript más utilizada. Es software libre y de código abierto, posee un doble licenciamiento bajo la Licencia MIT y la Licencia Pública General de GNUv2, permitiendo su uso en proyectos libres y privados. jQuery, al igual que otras bibliotecas, ofrece una serie de funcionalidades basadas en JavaScript que de otra manera requerirían de mucho más código, es decir, con las funciones propias de esta biblioteca se logran grandes resultados en menos tiempo y espacio (21).

A continuación, se muestran algunas características de jQuery (26):

- Manipulación de la hoja de estilos CSS.

- Soporta diferentes extensiones.

- Ofrece varias utilidades como obtener información del navegador, operar con objetos y vectores, funciones como trim () (elimina los espacios en blanco del principio y final de una cadena de caracteres).

- Compatibilidad con los navegadores Mozilla Firefox 2.0+, Internet Explorer 6+, Safari 3+, Opera 9+ y Google Chrome.

PHP

Hypertext Preprocessor (PHP) es un lenguaje de programación de propósito general de código del lado del servidor originalmente diseñado para el desarrollo web de contenido dinámico. Incrustado en HTML, significa que en un mismo archivo se puede combinar código PHP con código HTML. Puede ser utilizado en cualquier sistema operativo y servidor web. Posibilita la utilización de programación por procedimientos o programación orientada a objetos (POO), o una mezcla de ambas (22).

[6] Document Object Model

Entre las funciones más destacadas que presenta se pueden citar (23):

- Soporte para una gran cantidad de bases de datos: MySQL, PostgreSQL, Oracle, MSSQL Server, SybasemSQL e Informix.
- Integración con varias bibliotecas externas, permite generar documentos en PDF y analizar código XML.
- Perceptiblemente más fácil de mantener y poner al día que el código desarrollado en otros
- lenguajes.
- Soportado por una gran comunidad de desarrolladores, como producto de código abierto, PHP goza de la ayuda de un gran grupo de programadores, permitiendo que los fallos de funcionamiento se encuentren y reparen rápidamente.
- El código se pone al día continuamente con mejoras y extensiones de lenguaje para ampliar las capacidades de PHP.

Lenguaje de Marcado de Hipertexto (HTML)

Lenguaje utilizado para la publicación de hipertexto en la web, es decir, texto presentado de forma estructurada y agradable, con enlaces (hyperlinks) que conducen a otros documentos o fuentes de información relacionadas, y con inserciones multimedia (gráficos, sonido, etc.). La descripción se basa en especificar en el texto la estructura lógica del contenido (títulos, párrafos de texto normal, enumeraciones, definiciones y citas) así como los diferentes efectos que se quieren dar (especificar los lugares del documento donde se debe poner cursiva, negrita, o un gráfico determinado) y dejar que luego la presentación final de dicho hipertexto se realice por un programa especializado (24).

Es un lenguaje estático constituido por elementos que el navegador interpreta y los despliega en la pantalla de acuerdo con su objetivo. Se caracteriza por ser sencillo, presentar el texto de forma estructurada y agradable, no necesita de grandes conocimientos cuando se cuenta con un editor de páginas web, archivos pequeños, despliegue rápido, lenguaje de fácil aprendizaje y lo admiten todos los exploradores (24).

Hojas de Estilo en Cascadas (CSS)

Mecanismo que describe cómo se va a mostrar un documento en la pantalla, o cómo se va a imprimir, o incluso cómo va a ser pronunciada la información presente en el documento a través de un dispositivo de

lectura. Esta forma de descripción de estilos ofrece a los desarrolladores el control total sobre el estilo y formato de sus documentos.

Es utilizada para separar el contenido de la presentación y está basado en una serie de reglas, que rigen el estilo de los elementos en los documentos estructurados formando de esta forma la sintaxis de las hojas de estilo. Una de las principales características de CSS es su flexibilidad y las diferentes opciones que ofrece para realizar una misma tarea. Usar CSS tiene varias ventajas porque posibilita un mayor control de la presentación agilizando de esta manera su actualización. La hoja de estilo se almacena en caché después de la primera solicitud y se puede volver a usar para cada página del sitio, de esta forma, contribuye con menos costes de almacenamiento y más velocidad a la hora de cargar las páginas web (25).

1.6 Requisitos del software

La especificación de requisitos del software es una descripción completa del comportamiento del sistema a desarrollar. Incluye la descripción de todas las interacciones que se prevén que los usuarios tendrán con el software. También contiene requisitos no funcionales, que imponen restricciones de diseño o funcionamiento del sistema software (7).

1.6.1 Técnicas para la captura de requisitos

La captura de requisitos es la actividad donde el analista del equipo de desarrollo trabaja junto al cliente para describir el problema que el sistema debe resolver, los diferentes servicios que el sistema debe proporcionar, las restricciones que se pueden presentar y otros. Esta actividad debe ser muy efectiva ya que de su éxito dependerá que se satisfagan las necesidades del cliente (26).

Existen varias técnicas para la captura de requisitos, de ellas solo se seleccionaron para ser aplicadas las siguientes:

- **Entrevista**: es de gran utilidad para obtener información cualitativa como opiniones, o descripciones subjetivas de actividades. Es una técnica muy utilizada, requiere una mayor preparación y experiencia por parte del analista. La entrevista se puede definir como un “intento sistemático de recoger información de otra persona” a través de una comunicación interpersonal que se lleva a cabo por medio de una conversación estructurada. Debe quedar claro que no basta con hacer preguntas para obtener toda la información necesaria. Es muy importante la forma en que se plantea la conversación y la relación que se establece en la entrevista (27).

- **Tormenta de ideas**: consiste en reuniones con cuatro a diez personas donde como primer paso sugieren toda clase de ideas sin juzgar su validez (por muy disparatadas que parezcan), y después

de recopilar todas las ideas se realiza un análisis detallado de cada propuesta. Esta técnica se puede utilizar para identificar un primer conjunto de requisitos en aquellos casos donde no están muy claras las necesidades que hay que cubrir, o cuando se está creando un sistema que habilitará un servicio nuevo para la organización (27).

1.6.2 Requisitos funcionales.

Los requisitos funcionales son declaraciones de servicios que el sistema debería proporcionar, cómo el sistema debería reaccionar a entradas particulares, y cómo debería comportarse el sistema en situaciones particulares. En algunos casos, los requisitos funcionales también pueden indicar explícitamente lo que el sistema no debería hacer. Describen las transformaciones que el sistema realiza sobre las entradas para producir las salidas (28).

1.6.3 Historias de usuario.

Las historias de usuario (HU) son una representación de un requisito escrito en una o dos frases utilizando el lenguaje común del usuario. Son utilizadas en las metodologías de desarrollo ágiles para la especificación de requisitos (acompañadas de las discusiones con los usuarios y las pruebas de validación). Las HU son una forma rápida de administrar los requisitos de los usuarios sin tener que elaborar gran cantidad de documentos formales y sin requerir de mucho tiempo para administrarlos. Permiten responder rápidamente a los requisitos cambiantes (29). Para la solución que se propone en la investigación se definen cuatro HU, las cuales son explicadas en el acápite *Plan de iteraciones*. Estas HU son definidas a partir de los cuatro requisitos funcionales mencionados al inicio del capítulo.

1.6.4 Requisitos no funcionales.

Los requisitos no funcionales son restricciones que debe cumplir la aplicación. Consisten en restricciones impuestas por el entorno y las tecnologías. Pueden ser especificaciones sobre tiempo de respuesta o volumen de información tratado por una unidad de tiempo, requisitos en cuanto a interfaces, extensibilidad y facilidad de mantenimiento (28).

1.6.5 Técnicas para validación de requisitos.

Existen varias técnicas para validar los requisitos, las cuales se aplican con el objetivo de examinar las especificaciones para asegurar que todos los requisitos han sido establecidos sin ambigüedad, sin inconsistencias, sin omisiones y que los errores detectados hayan sido corregidos (7). Las técnicas a emplear para validar los requisitos de la solución son:

- **Prototipo de interfaz de usuario**: el prototipo de interfaz de usuario es una técnica de representación aproximada de la interfaz de usuario de un sistema software que permite a clientes y usuarios entender más fácilmente la propuesta de los ingenieros de requisitos para resolver sus problemas de negocio (28). Los dos tipos principales de prototipos de interfaz de usuario son:
    - **Desechables**: se utilizan sólo para la validación de los requisitos y posteriormente se desechan. Pueden ser prototipos en papel o en software (28).
    - **Evolutivos**: una vez utilizados para la validación de los requisitos, se mejora su calidad y se convierten progresivamente en el producto final (28).
- **Caso de prueba**: es un conjunto de condiciones o variables bajo las cuales un analista determinará si una aplicación, un sistema software, o una característica de éstos es parcial o completamente satisfactoria (28).

1.7 Patrones para el desarrollo de software.

Según (30) cada patrón describe un problema que ocurre una y otra vez en nuestro entorno, para describir después el núcleo de la solución a ese problema, de tal manera que esa solución pueda ser usada más de un millón de veces sin hacerlo siquiera dos veces de la misma forma. Cada uno es una regla de tres partes, que expresa una relación entre un contexto, un problema y una solución.

1.7.1 Patrones de arquitectura.

Para el desarrollo de la solución se selecciona como patrón arquitectónico; Modelo-Vista-Controlador (conocido por sus siglas en inglés MVC, Model View Controller), permite realizar la programación multicapa, separando en tres componentes distintos los datos de una aplicación, la interfaz del usuario y la lógica de control. Este patrón se ve usualmente en aplicaciones web, donde el controlador recibe todas las peticiones de la aplicación y entonces trabaja con el modelo para preparar los datos que necesita la vista, la vista es la página HTML y el código que provee de datos dinámicos a la página, esta utiliza los datos preparados por el controlador para generar una respuesta final. Por tanto, los tres niveles de MVC serían (28):

- **Modelo**: representa la información con la que trabaja la aplicación, o sea, su lógica de negocio.
- **Vista**: convierte el modelo en una página web que facilita al usuario interactuar con ella.
- **Controlador**: es el encargado de procesar las interacciones del usuario y ejecuta los cambios adecuados en el modelo o en la vista.

1.7.2 Patrones de diseño.

Un patrón de diseño es una descripción de clases y objetos comunicándose entre sí adaptada para resolver un problema de diseño general en un contexto particular. Además de permitir describir el sistema con el suficiente detalle para ser implementado (31). Los patrones de diseño tienen las siguientes ventajas:

- Proponen una forma de reutilizar la experiencia de los desarrolladores, para ello clasifican y describen formas de solucionar problemas que ocurren de forma frecuente en el desarrollo (31).

- Además, están basados en la recopilación del conocimiento de los expertos en desarrollo de software por lo que es una experiencia real, probada, que funciona y ayuda a no cometer los mismos errores.

- Mediante el uso de los patrones de diseño es más probable no consumar los mismos errores.

Patrones GRASP[7]

Los patrones GRASP constituyen un apoyo para la enseñanza que ayudan a entender el diseño de objetos esencial, y aplica el razonamiento para el diseño de una forma sistemática, racional y explicable. Este enfoque para la compresión y utilización de los principios de diseño se basa en los patrones de asignación de responsabilidades (32). A continuación, se mencionan los patrones utilizados en el diseño de la solución.:

- **Experto**: es un patrón que se usa más que cualquier otro al asignar responsabilidades; es un principio básico que suele ser útil en el diseño orientado a objetos. El cumplimiento de una responsabilidad requiere a menudo información distribuida en varias clases de objetos. El patrón Experto asigna responsabilidades a las clases que tienen la información necesaria para cumplir con la responsabilidad.

- **Creador**: guía la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos. El propósito fundamental de este patrón es encontrar un creador que debemos conectar con el objeto producido en cualquier evento. Al escogerlo como creador, se da soporte al bajo acoplamiento. Lo que define este patrón es que una instancia de un objeto la tiene que crear el objeto que tiene la información para ello.

- **Alta cohesión**: mantiene la complejidad dentro de los límites manejables, es decir asigna una responsabilidad de modo que la cohesión siga siendo alta. La cohesión es una medida de cuan

[7] General Responsibility and Assignment Software Patterns.

relacionadas y enfocadas están las responsabilidades de una clase. Una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme.

- **Bajo Acoplamiento**: es un patrón evaluativo que el diseñador aplica al juzgar sus decisiones de diseño. Este patrón estimula asignar una responsabilidad de modo que su colocación no incremente el acoplamiento tanto que produzca los resultados negativos propios de un alto acoplamiento. Soporta el diseño de clases más independientes, que reducen el impacto de los cambios, y también más reutilizables, que acrecienten la oportunidad de una mayor productividad

- **Controlador**: es un patrón que sirve como intermediario entre una determinada interfaz y el algoritmo que la implementa, de tal forma que es la que recibe los datos del usuario y la que los envía a las distintas clases según el método llamado. Asigna las responsabilidades de capturar los eventos del sistema a las clases.

Patrones GoF[8]

Los patrones GoF pertenecen al campo del Diseño Orientado a Objetos. Están conformados por patrones que se clasifican según su propósito en creacionales, estructurales y de comportamiento (33).

- **Patrones de creación:**

    - **Constructor**: separa la construcción de un objeto complejo de su representación, de forma que el mismo proceso de construcción pueda crear diferentes representaciones.

    - **Método de la fábrica**: este patrón define una interfaz para crear un objeto, pero deja que sean las subclases quienes decidan qué clase instanciar. Además, permite que una clase delegue en sus subclases la creación de objetos.

    - **Prototipo**: especifica los tipos de objetos a crear por medio de una instancia de un prototipo, y crear nuevos objetos copiando este mismo prototipo.

- **Patrones estructurales:**

    - **Adaptador**: convierte la interfaz de una clase en otra distinta que es la que esperan los clientes. Además, permiten que cooperen clases que de otra manera no podrían hacerlo, pues tendrían interfaces incompatibles.

[8] Gand of Four.

- **Puente**: desvincula una abstracción de su implementación, de manera que ambas puedan variar de forma independiente.

- **Compuesto**: combina objetos en estructuras de árbol para representar jerarquías de parte todo. Permite que los clientes traten de manera uniforme a los objetos individuales y a los compuestos.

- **Decorador**: añade dinámicamente nuevas responsabilidades a un objeto, proporcionando una alternativa flexible a la herencia para extender funcionalidades.

- **Fachada**: proporciona una interfaz unificada para un conjunto de interfaces de un subsistema. Define una interfaz de alto nivel que hace que sea más fácil de utilizar el subsistema.

- **Patrones de comportamiento**

  - **Cadena de Responsabilidad**: evita acoplar el emisor de una petición a su receptor, al dar más de un objeto la posibilidad de responder a la petición. Crea una cadena con los objetos receptores y pasa la petición a través de la cadena hasta que esta sea tratada para un objeto.

  - **Iterador**: proporciona un modo de acceder secuencialmente a los elementos de un objeto agregado sin exponer su representación interna.

  - **Mediador**: define un objeto que encapsula cómo interactúan un conjunto de objetos. Promueve un bajo acoplamiento al evitar que los objetos se refieran unos a otros explícitamente y permite variar la interacción entre ellos de forma independiente.

  - **Observador**: define una dependencia de uno a muchos entre objetos, de forma que cuando un objeto cambia de estado se notifica y actualizan todos los objetos automáticamente.

  - **Estado**: permite que un objeto modifique su comportamiento cada vez que cambia su estado interno. Tal parece que cambia la clase del objeto.

1.8 Métricas

Una métrica es una medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo determinado. (34).

1.8.1 Métricas de validación del diseño

Las métricas cuantifican los atributos del diseño de manera que permiten evaluar su calidad. Dentro de las métricas para el diseño se encuentran las especializadas en diseño orientado a objetos, las cuales miden características de clases. Permiten averiguar cuan bien están definidas las clases y el sistema (34). Miden de forma cuantitativa la calidad de los atributos internos del producto como son la responsabilidad de las clases, la reutilización, la complejidad de implementación y el mantenimiento. Además, permiten evaluar el bajo acoplamiento y la cantidad de pruebas de unidad que deben realizarse.

El diseño de la solución propuesta se valida utilizando las métricas Tamaño Operacional de Clases (TOC por sus siglas) y Relaciones entre clases (RC por sus siglas). Según (35) estas son métricas de validación del diseño a nivel de componentes y evalúan gran parte de los atributos de software que se han mencionado, siendo esta la razón por la cual han sido seleccionadas.

Tamaño Operacional de Clase:

El tamaño operacional de una clase está dado por el número de métodos asignados a una clase. Para ello mide los siguientes atributos de calidad (36):

- **Responsabilidad**: responsabilidad asignada a una clase. Un aumento del TOC implica un aumento de esta responsabilidad, teniendo así demasiada responsabilidad, lo cual reduce la posibilidad de reutilización de la clase, lo que hace complicada la implementación y la prueba.
- **Complejidad de implementación:** grado de dificultad que tiene implementar un diseño de clases determinado. Un aumento del TOC implica un aumento de la complejidad de implementación de una determinada clase.
- **Reutilización**: grado de reutilización presente en una clase o estructura de clase. Un aumento del TOC implica una disminución del grado de reutilización de una determinada clase.

Relaciones entre Clases:

Esta métrica está dada por el número de relaciones de uso de una clase. Para ello mide los siguientes atributos de calidad (36):

- **Acoplamiento**: grado de dependencia de una clase o estructura de clase, con otras. Un aumento del RC provoca aumento del acoplamiento de la clase.

- **Complejidad de mantenimiento**: grado de esfuerzo necesario para desarrollar un arreglo, rectificación de algún error o mejora de un diseño. Un aumento del RC provoca aumento de la complejidad del mantenimiento de la clase.

- **Reutilización**: grado de reutilización presente en una clase o estructura de clase. Un aumento del RC provoca disminución en el grado de reutilización de la clase.

- Cantidad de pruebas: grado de esfuerzo necesario para realizar pruebas de unidad al sistema diseñado. Un aumento del RC provoca aumento de la cantidad de pruebas de unidad necesarias para probar una clase.

1.9 Evaluación de software.

Las pruebas presentan una interesante anomalía para el ingeniero del software. El ingeniero crea una serie de casos de prueba que intentan demoler el software construido. La prueba demuestra hasta qué punto las funciones del software parecen funcionar de acuerdo con las especificaciones y parecen alcanzarse los requisitos de rendimiento. Además, los datos que se van recogiendo a medida que se llevan a cabo las pruebas proporcionan una buena indicación de la fiabilidad del software y, de alguna manera, indican la calidad del software como un todo (7). Para la evaluación del software se realizan pruebas de caja blanca, pruebas de caja negra y pruebas de aceptación.

1.9.1 Pruebas de Caja Blanca.

Las pruebas de caja blanca (también conocidas como pruebas de caja de cristal o pruebas estructurales) se centran en los detalles procedimentales del software, por lo que su diseño está fuertemente ligado al código fuente. Habitualmente se aplican a las unidades de software. Su objetivo es comprobar los flujos de ejecución dentro de cada unidad, pero también pueden probar los flujos entre unidades durante la integración, e incluso entre subsistemas, durante las pruebas de sistema. La prueba de caja blanca es un método de diseño de casos de prueba que usa la estructura de control del diseño procedimental para obtener los casos de prueba (7).

Una técnica de las pruebas unitarias es la prueba del camino básico. Esta técnica permite al diseñador de casos de prueba obtener la complejidad lógica de un procedimiento o algoritmo y usar esta como guía para la definición de un conjunto básico de caminos de ejecución. Los casos de prueba obtenidos del conjunto básico garantizan que durante la prueba se ejecuta por lo menos una vez cada sentencia del programa (7).

La complejidad ciclomática es una métrica del software que proporciona una medición cuantitativa de la complejidad lógica de un programa. Cuando se usa en el contexto del método de prueba del camino básico,

el valor calculado como complejidad ciclomática define el número de caminos independientes del conjunto básico de un programa y proporciona un límite superior para el número de pruebas que se deben realizar para asegurar que se ejecuta cada sentencia al menos una vez (7). Se selecciona para la realización de las pruebas unitarias o pruebas de caja blanca el método de prueba del camino básico, junto a la métrica complejidad ciclomática. Para llevar a cabo esta técnica se deben realizar los siguientes pasos:

- Representar el programa en un grafo de flujo.
- Calcular la complejidad ciclomática.
- Determinar el conjunto básico de caminos independientes.
- Derivar los casos de prueba que fuerzan a la ejecución de cada camino.

1.9.2 Pruebas de Caja Negra.

Las pruebas de caja negra, también denominada prueba de comportamiento, se centran en los requisitos funcionales del software. Estas permiten al ingeniero del software obtener conjuntos de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa (7). Hacen referencia a pruebas que se llevan a cabo sobre la interfaz del software sin tener en cuenta el código de la aplicación. Los casos de prueba pretenden demostrar que las funciones del software son operativas, que la entrada se acepta de forma adecuada y que se produce una salida correcta. Según (7) estas pruebas tienen como propósito detectar:

- Funciones incorrectas o ausentes.
- Errores de interfaz.
- Errores en estructuras de datos o en accesos a bases de datos externas.
- Errores de rendimiento.
- Errores de inicialización y de terminación.

1.10 Conclusiones parciales

Al finalizar el presente capítulo se arribó a las siguientes conclusiones:

- Tras el análisis realizado teniendo en cuenta la fundamentación teórica de la investigación en curso se ha determinado que actualmente los desarrolladores del sistema VUA necesitan una herramienta que disminuya el tiempo en la implementación de formularios.

- Se construyó el marco teórico de la investigación relacionado con las herramientas para diseñar y generar formularios web, así como modelos, metodologías, herramientas y lenguajes para el desarrollo de software.

- El estudio de las herramientas para el diseño y la generación de formularios web permitió conocer que ninguna de las existentes soluciona el problema planteado, aunque brindó conocimiento esencial para la implementación del software propuesto.

CAPÍTULO 2. DESCRIPCIÓN DE LA PROPUESTA

2.1 Introducción

A continuación, se describe la propuesta de solución. Se definen y validan los requisitos funcionales y no funcionales, utilizando un conjunto de técnicas. Luego se describen los mismos a partir de las historias de usuario que propone el escenario número cuatro de la metodología AUP-UCI, estimando el tiempo y el esfuerzo que se necesita para implementar cada una. Se presenta el plan de iteraciones para el desarrollo del sistema. Resultado del diseño se describe la arquitectura y el diseño del software mediante los patrones de diseño aplicados, y se valida el diseño mediante el uso de métricas.

2.2 Diseño de la propuesta de solución.

La solución que se propone como resultado de este trabajo es el desarrollo de una herramienta para la generación de formularios web. Esta herramienta implicaría una reducción en cuanto al tiempo empleado en el proceso de implementación de formularios web que realizan los desarrolladores en el sistema VUA.

La propuesta se describe conceptualmente de la siguiente manera. El sistema propuesto permitirá al programador diseñar y generar formularios web sin necesidad de programar. Mediante el mismo el desarrollador podrá mover los elementos que necesite de un lado a otro hasta conformar el formulario. Una vez incorporados todos los componentes deseados dentro de una entidad que muestra el sistema, se le asigna el nombre para poder ser generada. En caso de existir errores, el sistema muestra un mensaje informando su ubicación y la solución del mismo. Luego de ser creada la entidad en el sistema, muestra la opción para generar el formulario con el código de todos los componentes previamente añadidos. Por último, el sistema crea el código del formulario de manera tal que pueda ser reutilizable. De la descripción anterior se puede deducir el proceso fundamental, generador formulario y los subprocesos generar entidad y generar formulario.

1 **Generar entidad:** en esta fase se crea la entidad que contiene todos los componentes del formulario. Para poder crear la entidad, debe contar con al menos un componente y tener un nombre asignado. El sistema muestra la ruta donde se encuentra la nueva entidad.

2 **Generar formulario:** este proceso posibilita crear el formulario a partir de la entidad creada. Una vez creado se genera el código de todos los componentes que se encuentran en la entidad. El sistema muestra una opción para visualizar el formulario y un mensaje con la ruta donde se encuentra el nuevo formulario.

2.2.1 Diagrama de procesos.

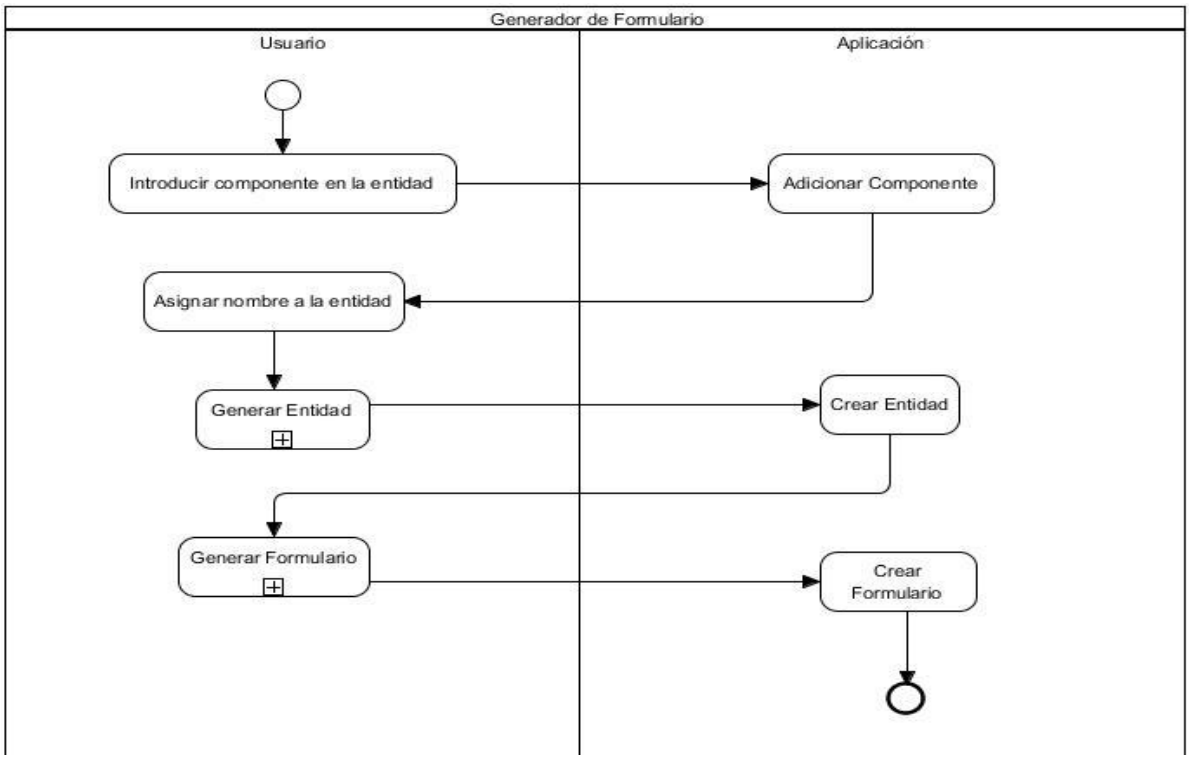


Ilustración 1: Diagrama del proceso de generación de formularios web.

En la *Ilustración 1* se describe el funcionamiento de la aplicación a través de los diferentes procesos que la componen. El flujo comienza con la tarea Introducir componentes en la entidad y culmina con la tarea Crear Formulario. Son representados además las tareas Adicionar Componente, Asignar nombre a la entidad y Crear Entidad; junto a los subprocesos Generar Entidad y Generar Formulario. Estos subprocesos son agrupados el proceso principal Generador de formularios. El diagrama además especifica los actores que intervienen en cada proceso.

2.3 Requisitos.

2.3.1 Técnicas para la captura de requisitos.

Estas técnicas se aplicaron con el objetivo de tener un mejor entendimiento del problema y de las necesidades del cliente. A partir de estas se definen los requisitos funcionales que debe cumplir la propuesta de solución. A continuación, se describen las técnicas utilizadas en la captura de requisitos de la presente solución:

Entrevista: la entrevista se realizó a partir de la selección de un conjunto de preguntas que responden a lo anterior y fueron realizadas al Ing. Leansi Vega Rouco, uno de los desarrolladores del sistema VUA, que a su vez es uno de los clientes y tiene conocimientos sobre el objeto de estudio y campo de acción de la investigación (Consultar *Anexo 1*).

Tormenta de ideas: en varias reuniones se desarrollaron tormentas de ideas, o debates con la participación del equipo de desarrollo y el cliente. Como resultado se logró generar opiniones sobre los requisitos a satisfacer por el software. En un principio se definieron siete requisitos funcionales, resumiéndolos en cuatro en un encuentro final, estos son mostrados en la *Tabla 2*.

Las técnicas realizadas brindaron información fundamental para el desarrollo de la propuesta de solución. A partir de ella se definieron los requisitos funcionales de la solución, los cuales constituyen la base para lograr una correcta implementación y así cumplir con las necesidades y expectativas del cliente.

2.3.2 Definición de los requisitos funcionales.

Luego de aplicadas las técnicas de entrevista y tormenta de ideas se identificaron un total de cuatro requisitos funcionales. La prioridad de cada requisito fue definida por el cliente en función de la importancia del negocio. La complejidad se obtuvo a partir de la utilización del artefacto *Evaluación de requisitos* propuesto por el área de proceso *Administración de requisitos*, ver *Anexo 4*. La *Tabla 2* muestra los requisitos funcionales de la propuesta de solución:

Tabla 2: Requisitos funcionales.

No.	Requisito	Prioridad	Complejidad
1	Adicionar Componente	Alta	Media
2	Eliminar Componente	Alta	Media
3	Generar Entidad	Alta	Alta
4	Generar Formulario	Alta	Alta

2.4 Historias de usuario.

2.4.1 Estimación de esfuerzo por Historias de Usuario.

Según la prioridad asignada por el cliente a cada HU y teniendo en cuenta la complejidad determinada por el programador, se realiza la estimación de cada una de las HU identificadas. Los resultados de la estimación

se muestran en la *Tabla 3*. La unidad de estimación es el punto. Un punto equivale a una semana ideal de programación.

Tabla 3: Estimación por HU.

Historias de Usuario	Puntos de estimación
Adicionar Componente	2
Eliminar Componente	1
Generar Entidad	3
Generar Formulario	3

Como se puede apreciar en la *Tabla 3*, los requisitos Adicionar Componente y Eliminar Componente requieren menor tiempo de implementación, puesto que poseen menor complejidad. Los requisitos Generar Entidad y Generar Formulario, al contar con mayor complejidad, requieren mayor tiempo de implementación.

2.4.2 Plan de Iteraciones.

Una vez descritas las HU y su estimación por el cliente para su posterior implementación por los desarrolladores involucrados, se procede a la planificación de la entrega del sistema, agrupando todas las HU que serán desarrolladas por cada iteración. A partir de lo planteado se decide desarrollar el sistema en dos iteraciones, en un total de nueve semanas, o sesenta y tres días, los cuales se especifican a continuación.

Tabla 4: Plan de Iteraciones.

Iteración	Historias de Usuario	Puntos de estimación	Estimación total
1	Adicionar Componente	2	6
	Eliminar Componente	1	

	Generar Entidad	3	
2	Generar Formulario	3	3

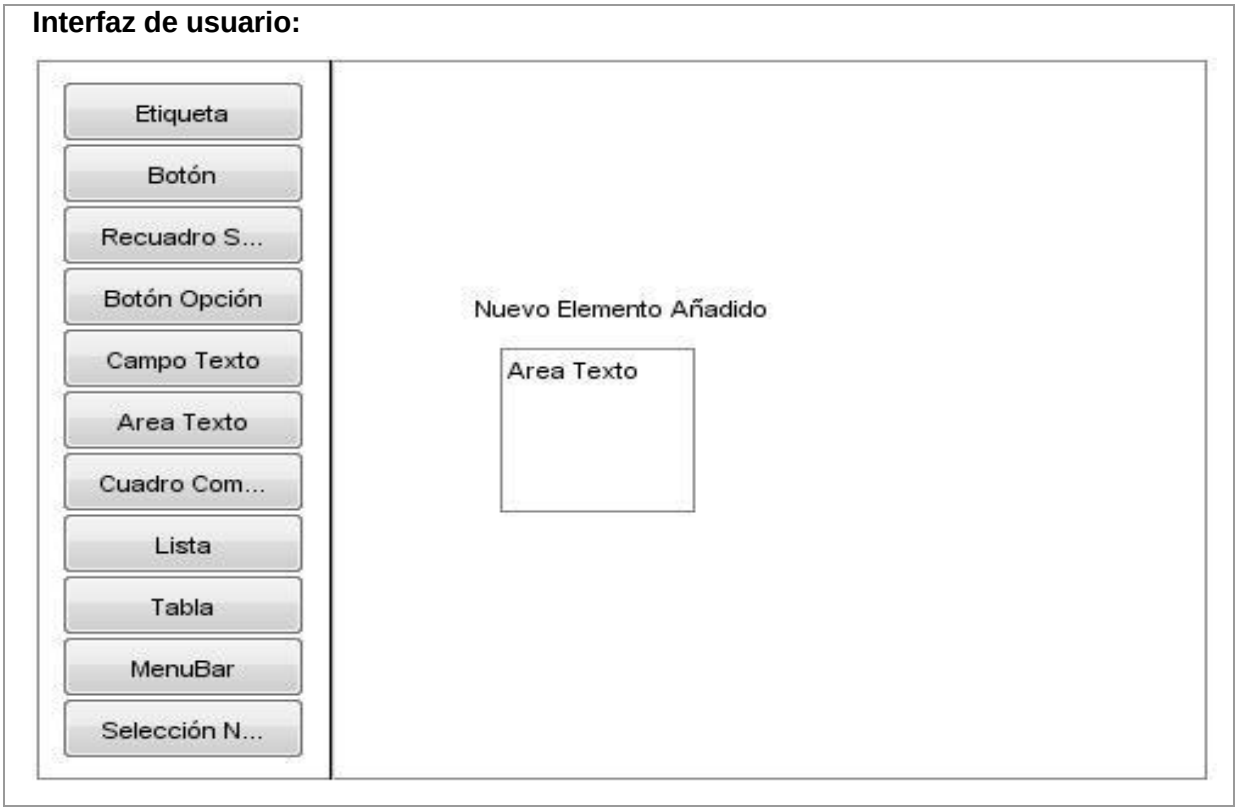
Iteración 1

En esta iteración se implementan las HU *Adicionar Componente*, *Eliminar Componente* y *Generar Entidad*. La primera permite al desarrollador seleccionar y arrastrar un componente del menú y soltarlo dentro de la entidad. La segunda HU permite eliminar un componente que haya sido agregado anteriormente a la entidad. La última HU de esta iteración permite al desarrollador generar una entidad con todos los componentes agregados. A continuación, en la *Tabla 5* se muestra descrita la HU Adicionar Componente, para observar las demás, remítase al *Anexo 6*.

Tabla 5: Historia de usuario de Adicionar Componente.

HISTORIA DE USUARIO	
Número: HU-1	**Nombre historia de usuario:** Adicionar Componente.
Usuario: Osniel Ramos	**Iteración asignada:** 1
Prioridad en negocio: Alta	**Puntos estimados:** 2
Riesgo en desarrollo: Medio	**Puntos reales:** 1
Descripción: Permite al desarrollador adicionar un nuevo componente a la entidad.	
Observaciones: Los elementos son; jTextField, RadioButton, CheckBox, Select.	

Interfaz de usuario:



Iteración 2

En la segunda y última iteración se implementa la HU Generar Formulario. Con esta HU se pretende generar el formulario a partir de la entidad previamente generada. Para consultar esta HU remítase al Anexo 6.

2.5 Definición de los requisitos no funcionales

Para la solución se proponen los siguientes requisitos no funcionales:

Tabla 6: Requisitos no funcionales.

Descripción		
RNF		
Usabilidad		La aplicación debe presentar una vista sencilla, descriptiva y fácil de usar, con el objetivo de proporcionar a los usuarios un mejor entendimiento con la aplicación.

Portabilidad	La aplicación debe poder instalarse en los sistemas operativos Windows y Linux/Unix.	
Disponibilidad	La aplicación deberá estar disponible siempre, asegurando el acceso desde cualquier lugar de red a los usuarios.	
	El sistema permanecerá fuera de servicio sólo en caso que se estén realizando tareas de mantenimiento o de configuración.	
	No deben ocurrir errores críticos, los cuales son: pérdida total de la información, o inhabilitación para el uso de ciertas partes del funcionamiento del sistema.	
Interfaz		La aplicación debe ofrecer una interfaz amigable y un diseño sencillo que le permita al usuario interactuar fácilmente.
		La aplicación, al ocurrir un error con los datos de entrada muestra mensajes al usuario informando este.
Servidor	Debe tener 2 GB de memoria RAM o superior, un disco duro de 320 GB de almacenamiento o superior, un procesador Intel Core 2 Duo 2.0 GHz o superior.	El servidor de aplicación debe tener instalado el servidor web Apache y el marco de trabajo Symfony 2.8.
Cliente	Las PC clientes deben tener 512 MB, como mínimo de memoria RAM, un procesador Pentium IV 1.7 GHz o superior.	Las PC clientes deben tener instalado un navegador web; se recomienda para estas Mozilla Firefox en su versión 24.0 o superior.

2.6 Validación de requisitos.

A continuación, se describen las técnicas utilizadas en la validación de requisitos:

- **Prototipo de interfaz de usuario:** se aplican prototipos de interfaces de usuario evolutivos. Se les muestra a los clientes y usuarios finales aproximaciones de la interfaz de usuario a medida que se

avanza en el desarrollo con el objetivo de corregir errores. Se realizan reuniones con los ingenieros Leansi Vega Rouco y Leduan Flores Riera que son usuarios finales del sistema para recoger sus inquietudes y mejorar la calidad de las interfaces, convirtiéndolas en el producto final.

- **Caso de prueba:** se diseñan un conjunto de casos de prueba para comprobar si los requisitos identificados se corresponden con las necesidades del cliente. La ejecución de las pruebas permite encontrar no conformidades que existan en la implementación de los requisitos. Los casos de pruebas creados, su utilización y las no conformidades obtenidas son explicados en el *Capítulo 3*, en el acápite correspondiente a las pruebas de caja negra.

El principal resultado de aplicar las técnicas de manera sistemática consiste en la constante actualización y refinamiento de los requisitos, lo cual tributa al desarrollo de una solución eficaz.

2.7 Patrones para el desarrollo de software.

2.7.1 Patrón de arquitectura.

Es el encargado de separar los datos de la aplicación, la interfaz de usuario y la lógica de control en tres componentes distintos. La arquitectura del sistema está basada en el patrón MVC, puesto que se desarrolla en el marco de trabajo Symfony, el cual facilita la utilización de este patrón. Véase en la *Ilustraciön 10.*

Modelo: entidades y el repositorio.

Vista: vistas definidas en formato HTML.

Controlador: clase controladora del sistema.

2.7.2 Patrones GRASP

A continuación, se describen los patrones GRASP empleados en el desarrollo del sistema.

- **Experto**: el uso de este patrón resulta de utilidad en las clases del modelo, las cuales contienen toda la información relacionada con los objetos que representan. La *Ilustración 3* muestra la declaración del atributo *name*, perteneciente a la clase *Model*.

```
private $name;

public function setName($name)
{
    $this->name = $name;

    return $this;
}

public function getName()
{
    return $this->name;
}
```

Ilustración 2: Patrón experto en la clase Model.

- **Creador:** este patrón se pone de manifiesto en las clases controladoras del sistema. Brinda soporte a un bajo acoplamiento y mejora la reutilización. La *Ilustración 4* muestra el uso de este patrón mediante la creación de la función *createEntityByJson()* en la clase controladora *DefaultController*, el cual comienza el proceso de crear las entidades en el *Bundle*. .

```
public function createEntityByJson(Request $request, Util $util): Response
{
    $json = json_decode($request->get('form_json'));
    $entity = ucfirst(strtolower($request->get('entity')));
    $util->executeCommand(
        $util->getApplication($this->get('kernel')),
        $util->getInputOutput($util->getArrayInputEntity($json, $entity))
    );
```

Ilustración 3: Patrón creador en la clase DefaultController.

- **Alta cohesión:** en el diseño del sistema las clases tienen responsabilidades estrechamente relacionadas y no realizan un trabajo excesivo, lo que permite simplificar el mantenimiento y aumentar la capacidad de reutilización de estas. Symfony permite asignar responsabilidades con una alta cohesión. Este se evidencia en las clases *Repository*, que tienen como única responsabilidad realizar operaciones de acceso a datos solo con la entidad que representa. En la *Ilustración 5* se puede observar la utilización de este patrón en la clase *ModelRepository*.

```php
<?php

namespace AppBundle\Repository\Model;

use Doctrine\ORM\EntityRepository;

/**
 * Clase Repository de nuestra clase Entity/Model. No borrar!!!!
 *
 * ModelRepository
 *
 * This class was generated by the Doctrine ORM. Add your own custom
 * repository methods below.
 */
class ModelRepository extends EntityRepository
{
}
```

Ilustración 4: Patrón alta cohesión en la clase ModelRepository

- **Bajo Acoplamiento:** en el diseño del sistema cada clase depende lo menos posible de otra, de tal manera que una de ellas solo recurre a otra en caso de que exista referencia dentro de sus atributos, lo que permite que el sistema sea mucho más robusto y de fácil mantenimiento. Este se evidencia en el marco de trabajo, ya que dentro de la capa modelo la clase de abstracción de datos es la más reutilizable y no tiene asociaciones con las clases de la capa vista ni con el controlador.

- **Controlador:** este patrón se evidencia en la clase controladora, responsable de implementar las funcionalidades pertenecientes a una interfaz determinada. La *Ilustración 6* muestra el uso de este patrón en la clase controladora *DefaultController*, mostrándose específicamente la declaración de los métodos *createEntityByJson()* y *createFormTypeByEntity()*, en los cuales se genera la entidad y el formulario respectivamente .

```
<?php

namespace AppBundle\Controller;

use ...

/** Clase encargada de controlar el flujo de trabajo en los diferentes ...*/
class DefaultController extends Controller
{


   /** Ruta que comienza el proceso de crear las Entidades en ...*/
   public function createEntityByJson(Request $request, Util $util): Response{...}

   /** Ruta para la creación y visualización del formulario ...*/

   public function createFormTypeByEntity(Request $request, Util $util, $entity):
Response{...}
}
```

Ilustración 5: Patrón controlador en la clase DefaultController.

Estos patrones facilitan la localización de los diferentes objetos del sistema, así como la comunicación y el entendimiento entre desarrolladores y diseñadores.

2.7.3 Patrones GoF

A continuación, se describen los patrones GoF utilizados en el desarrollo del sistema.

Estructurales

- **Decorador**: aplicado a la generación de vistas, la solución que ofrece este patrón es la de añadir funcionalidad adicional a las plantillas. Ejemplo, añadir el menú y el pie de página a las plantillas que lo requieran; se trata de decorar las plantillas con elementos adicionales reutilizables. El sistema de plantillas Twig está provisto de un mecanismo de herencia gracias al cual la decoración de plantillas resulta de una flexibilidad y versatilidad total. En la *Ilustración 7* se muestra la aplicación de este patrón en la vista *base.html.twig,* en este fragmento de código se organizan los componentes del formulario en un menú de navegación.

```
<!DOCTYPE html>
<html>
   <head…>
   <body class="fixed-sidebar fixed-nav fixed-nav-basic">

      <div id="wrapper" class="tooltip-demo">

         <nav class="navbar-default navbar-static-side" role="navigation">
            <div class="sidebar-collapse">
               <div class="nav metismenu" id="side-menu">
                  <div class="dd" id="nestable">
                     <div class="spiner-example" id="spiner-example">
                        <div class="sk-spinner sk-spinner-three-bounce">
                           <div class="sk-bounce1"></div>
                           <div class="sk-bounce2"></div>
                           <div class="sk-bounce3"></div>
                        </div>
                     </div>
                     <ol class="dd-list animated fadeInDown" id="init-content"
style="display: none">

                        {{ include('element/index.html.twig', {'index': 1}) }}

                     </ol>
                  </div>
                  <textarea id="nestable-output" class="form-control" title=""
readonly rows="10" style="display: none;"></textarea>
               </div>
            </div>
         </nav>
```

Ilustración 6: Patrón decorador en la vista base.html.twig.

Comportamiento

- **Mediador**: la clase controladora, o sea la clase *DefaultController* es mediadora entre las capas de modelo y vista, brindando la posibilidad de reutilizar el código fuente. Para esto encapsula la comunicación entre los objetos, evitándose con ello la necesidad de que lo hagan directamente entre sí. En la *Ilustración 8* se puede apreciar la utilización de este patrón en el método createFormTypeByEntity () de la clase DefaulController, en el que se ejecutan los comandos que se encuentran en la clase Util para generar un formulario, dando respuesta a una llamada desde la vista.

```
public function createFormTypeByEntity(Request $request, Util $util, $entity):
Response
{
   if ($request->isMethod('GET')) {
      return $this->render('response/entity.html.twig', ['entity' =>
strtolower($entity)]);
   }
   $entity = ucfirst(strtolower($entity));
   $kernel = $this->get('kernel');
   $valid = $util->fileExist('Form/'.$entity.'Type.php');
   $response = 'The new '.$entity.'Type.php class file has been created';
   if (!$valid) {
      $response = $util->executeCommand(
         $util->getApplication($kernel),
         $util->getInputOutput($util->getArrayInputFormType($entity))
      );
      $valid = $util->fileExist('Form/'.$entity.'Type.php');
   }
   $this->addFlash(
      $valid ? 'success' : 'danger',
      $valid ? 'Action success' : 'Action code Error!!!. '.$entity.'Type no valid.'
   );
   $form = $valid ? $this->createForm('AppBundle\\Form\\'.$entity.'Type', null,
['data_class' => null]) : null;
   return $this->render(
      'response/form.html.twig',
      [
         'response' => $response,
         'valid' => $valid,
         'entity' => $entity,
         'form' => $form ? $form->createView() : null,
      ]
   );
}
```

Ilustración 7: Patrón mediador en la clase DefaultController.

2.8 Aplicación de las métricas de validación del diseño

Para la validación del diseño se aplicaron las métricas Tamaño Operacional de Clases (TOC) y Relaciones entre Clases (RC).

Tamaño Operacional de Clases.

Para el caso del software desarrollado se analizaron un conjunto de 5 clases, obteniendo el valor 6 como promedio en cantidad de procedimientos. Para asignar las categorías de *Baja*, *Media*, o *Alta* a los atributos *Responsabilidad*, *Complejidad de implementación* y *Reutilización* se tuvieron en cuenta los criterios mostrados en la *Tabla 7*, estos criterios son en relación con la cantidad de operaciones o métodos asignados a una clase. La variable *Prom*, hace referencia al promedio de procedimientos por clases existentes en el sistema.

Tabla 7:Criterios para asignar las categorías de la métrica TOC.

	Categoría	**Criterio**
Responsabilidad	Baja	< =Prom.
	Media	Entre Prom. y 2* Pom.
	Alta	> 2* Prom.
Complejidad implementación	Baja	< =Prom.
	Media	Entre Prom. y 2* Pom.
	Alta	> 2* Prom.
Reutilización	Baja	> 2*Prom.
	Media	Entre Prom. y 2* Pom.
	Alta	<= Prom.

Luego de aplicar los criterios de medición correspondientes a la métrica se obtienen los siguientes resultados mostrados en la *Ilustración 9.*

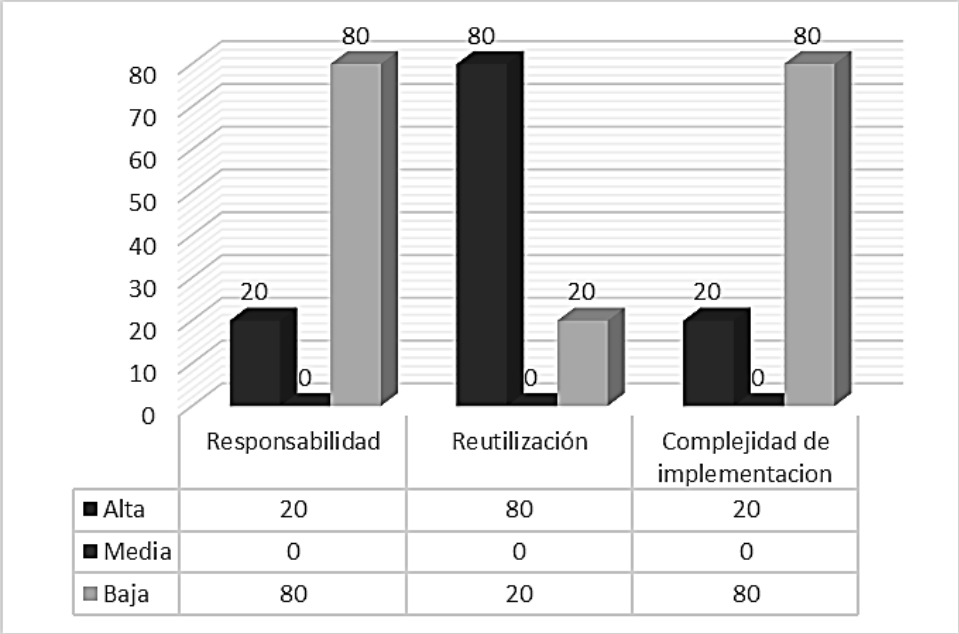


Ilustración 8: Resultados de aplicar métrica TOC.

Relaciones entre Clases.

Para el caso del software desarrollado se analizaron un conjunto de 5 clases, obteniendo el valor 0.8 como promedio de relaciones de uso entre las mismas. Para asignar las categorías de *Ninguno*, *Bajo*, *Medio*, o *Alto* a los atributos *Acoplamiento*, *Complejidad de mantenimiento*, *Reutilización* y *Cantidad de pruebas* se tuvieron en cuenta los criterios mostrados en la *Tabla 8*, estos criterios están dados por el número de relaciones de uso de una clase con otra. La variable *Prom*, hace referencia al promedio de asociaciones de uso por clases existentes en el sistema.

Tabla 8:Criterios para asignar las categorías de la métrica RC.

	Categoría	**Criterio**
Acoplamiento	Ninguno	0
	Bajo	1
	Medio	2
	Alto	>2
	Baja	<= Prom.

Complejidad Mantenimiento.	Media	Entre Prom. y 2*Prom.
	Alta	> 2*Prom.
Reutilización	Baja	>2* Prom.
	Media	Entre Prom. y 2*Prom.
	Alta	<= Prom.
Cantidad de Pruebas	Baja	<= Prom.
	Media	Entre Prom. y 2*Prom.
	Alta	> 2*Prom.

Luego de aplicar los criterios de medición correspondientes a la métrica se obtienen los siguientes resultados mostrados en la *Ilustración 8*:

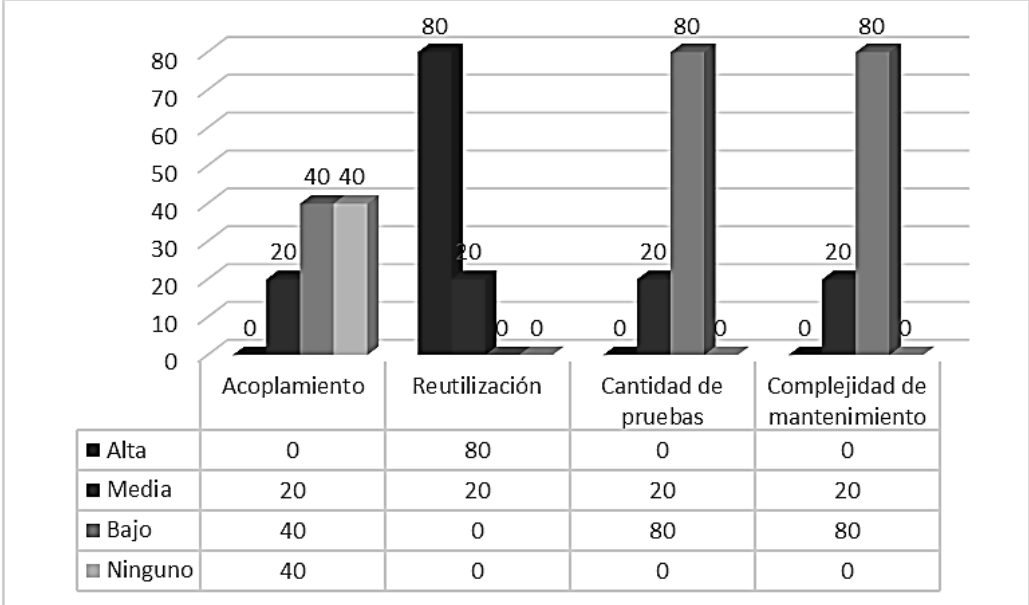


Ilustración 9: Resultados de aplicar métrica RC.

Analizando los resultados obtenidos luego de aplicadas las métricas se puede observar que el diseño propuesto provee un nivel bajo de responsabilidad. Esto trae consigo un porciento bajo de complejidad de implementación, lo cual significa un menor grado de dificultad en el desarrollo de la herramienta. El 80% de

las clases presentan un nivel bajo de mantenimiento, lo que reduce el grado de esfuerzo necesario para desarrollar un arreglo o rectificar algún error. Teniendo en cuenta que el 80% de las clases tienen una baja necesidad de realizar pruebas unitarias al sistema diseñado, se reduce el esfuerzo necesario para la realización de estas. Evidenciando el uso del patrón bajo acoplamiento, se puede observar el bajo acoplamiento de elementos que favorece a la reutilización, en la cual se alcanzaron los resultados deseados, pues se obtuvieron valores altos de reutilización en ambas métricas, 80% en TOC y en RC, ya que se busca el poder reutilizar las clases en caso de ser necesario. Teniendo en cuenta los resultados obtenidos y lo que representan para la solución, es posible afirmar que el diseño propuesto tributa al desarrollo de una aplicación con calidad.

2.9 Conclusiones Parciales.

Al finalizar el presente capítulo se arribó a las siguientes conclusiones:

- El equipo de desarrollo, en reuniones con el cliente, obtuvo los requisitos del software mediante las entrevistas y tormentas de ideas realizadas.
- La confección de los artefactos ingenieriles, *Historias de usuario* y *Plan de iteraciones*, permitieron organizar el proceso de desarrollo del sistema cumpliendo con lo establecido en el escenario cuatro de la metodología AUP-UCI.
- El uso de los patrones de diseño y de las métricas TOC y RC para la validación de este contribuyeron a la obtención de un diseño con calidad.

CAPÍTULO 3. IMPLEMENTACIÓN, PRUEBAS Y VALIDACIÓN.

3.1 Introducción

En el presente capítulo se muestra como fue implementada la solución propuesta a partir del proceso de análisis y diseño anterior. Se abordan los estándares de codificación empleados durante la implementación del sistema para garantizar un buen entendimiento y legibilidad del código. Se describen las pruebas efectuadas al software que tienen como objetivo: detectar y corregir errores en el sistema, antes de su entrega al cliente.

3.2 Modelo de implementación.

El modelo de implementación describe como los elementos del modelo de diseño se implementan en términos de componentes. Describe además como se organizan los componentes de acuerdo con los mecanismos de estructuración disponibles en el entorno de implementación y en los lenguajes de programación utilizados; y como se relacionan entre ellos.

3.2.1 Diagrama de componente.

En el diagrama de componentes mostrado en la *Ilustración 10* se puede apreciar cómo se relacionan las diferentes vistas del sistema con las clases y repositorios, pudiendo evidenciar además su organización haciendo uso del patrón MVC.

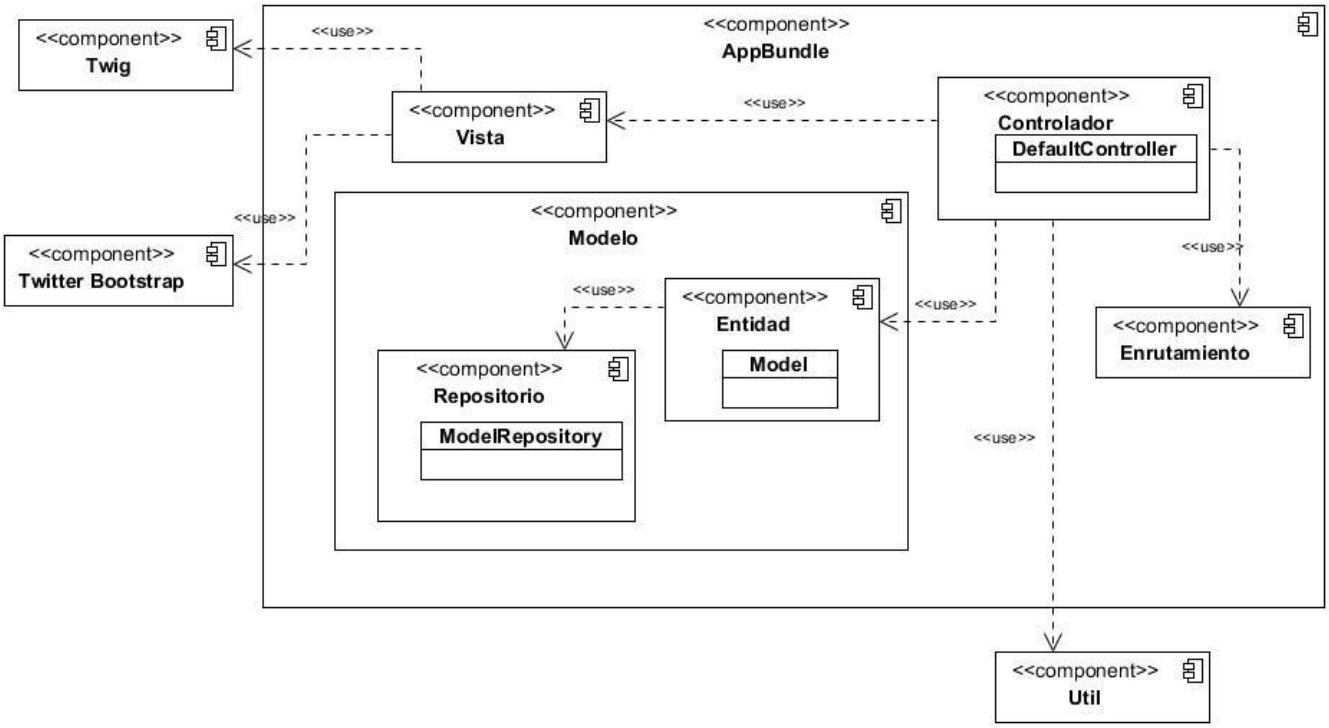


Ilustración 10: Arquitectura de la herramienta mediante el diagrama de componentes.

La *Ilustración 10* muestra la arquitectura de la herramienta a través del diagrama de componentes. Este está compuesto por un componente principal llamado AppBundle, el cual está asociado a otros componentes. Se puede apreciar el componente Modelo, donde se encuentran las clases Repositorios, que hacen uso de las clases Entidades. La Vista hace uso de los componentes Twig y TwitterBootstrap. Esta a su vez es usada por el Controlador, el cual utiliza los componentes Enrutamiento, Entidades y Util.

3.2.2 Algoritmos utilizados.

La creación o generación de las entidades y los formularios constituyen dos de las funcionalidades claves en la solución. A continuación, se describen con mayor nivel de detalle.

Cuando se desea generar un formulario, lo primero que se debe hacer es tener pensado los componentes que se desea tenga el formulario. Una vez se conozcan los componentes y el orden que se desea en el formulario, se seleccionan del menú de componentes (situado en el panel de la izquierda en la *Ilustración 11*), y se arrastran en la entidad mostrada en el centro de la imagen.

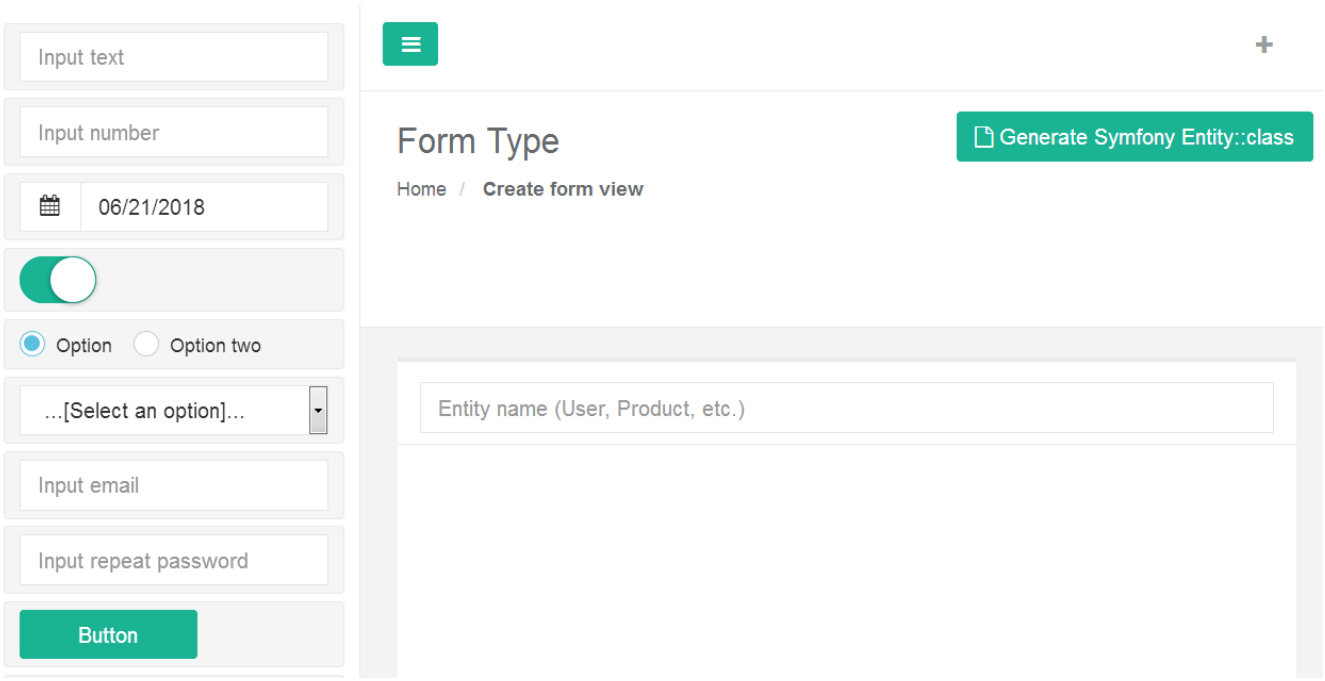


Ilustración 11: Interfaz de usuario para adicionar componentes.

1 Generar la entidad.

Una vez se encuentren los componentes dentro de la entidad se selecciona el botón *Generate Symfony Entity::class*, mostrado en la esquina superior derecha, el cual mediante el comando de Symfony *doctrine:generate:entity*, genera una nueva entidad y la almacena en el maco de trabajo, mostrando la ruta donde se encuentra. A continuación, en la *Ilustración 12* se muestra la interfaz con los componentes agregados en la entidad, y en la *Ilustración 13* el código mediante el cual se genera la entidad.

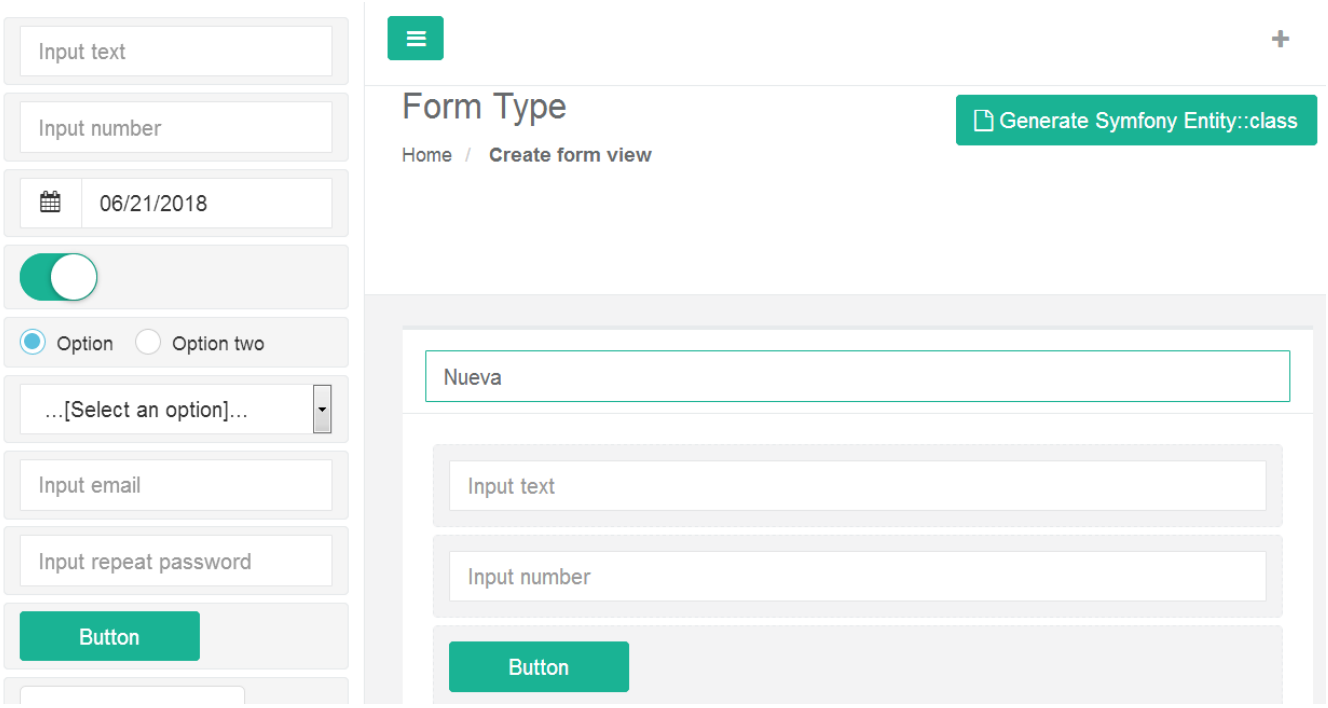


Ilustración 12: Interfaz de usuario para generar una nueva entidad.

```
public function createEntityByJson(Request $request, Util $util): Response
{
    $json = json_decode($request->get('form_json'));
    $entity = ucfirst(strtolower($request->get('entity')));
    $util->executeCommand(
        $util->getApplication($this->get('kernel')),
        $util->getInputOutput($util->getArrayInputEntity($json, $entity))
    );

    if (!$util->fileExist('Entity/'.$entity.'.php')) {
        $this->addFlash('danger', 'Action code Error!!!.
AppBundle/Entity/'.$entity.'.php no valid.');

        return $this->redirectToRoute('app_homepage', []);
    }

    $this->addFlash('success', 'Action success');
    return $this->redirectToRoute('app_create_form_by_entity', ['entity' =>
strtolower($entity)]);
}
```

Ilustración 13: Código ejecutado para generar una nueva entidad.

2 Generar el formulario.

Una vez se genere la entidad se procede a generar el formulario correspondiente (con todos los componentes agregados en la entidad), para esto se selecciona el botón Generate XType::class (X: nombre de la entidad), mostrado en la esquina superior derecha, en la *Ilustración 13.* El sistema, ejecuta el comando doctrine:generate:form, el cual genera el nuevo formulario y lo almacena en el marco de trabajo, mostrando la ruta donde se encuentra. A continuación, se muestra la interfaz de usuario para generar un nuevo formulario y el código correspondiente en la *Ilustración 14* e *Ilustración 15* respectivamente.

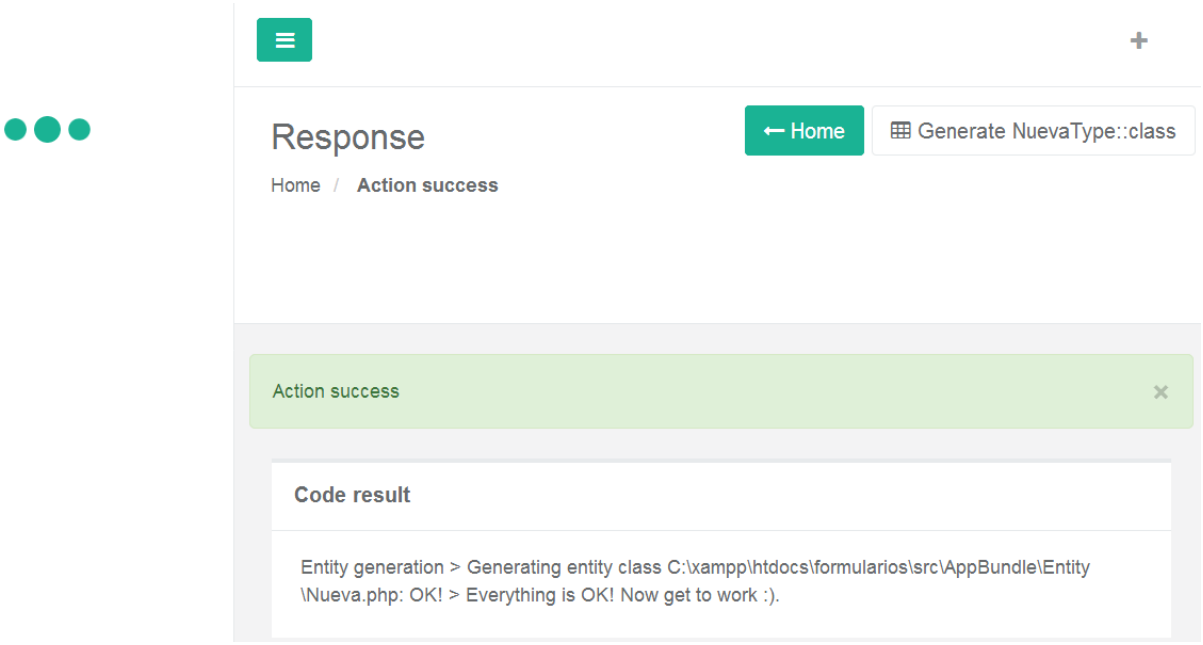


Ilustración 14: Interfaz de usuario para generar un nuevo formulario.

```
public function createFormTypeByEntity(Request $request, Util $util, $entity):
Response
{
    if ($request->isMethod('GET')) {
        return $this->render('response/entity.html.twig', ['entity' =>
strtolower($entity)]);
    }
    $entity = ucfirst(strtolower($entity));
    $kernel = $this->get('kernel');
    $valid = $util->fileExist('Form/'.$entity.'Type.php');
    $response = 'The new '.$entity.'Type.php class file has been created';
    if (!$valid) {
        $response = $util->executeCommand(
            $util->getApplication($kernel),
            $util->getInputOutput($util->getArrayInputFormType($entity))
        );
        $valid = $util->fileExist('Form/'.$entity.'Type.php');
    }
    $this->addFlash(
        $valid ? 'success' : 'danger',
        $valid ? 'Action success' : 'Action code Error!!!. '.$entity.'Type no valid.'
    );
    $form = $valid ? $this->createForm('AppBundle\\Form\\'.$entity.'Type', null,
['data_class' => null]) : null;
    return $this->render(
        'response/form.html.twig',
        [
            'response' => $response,
            'valid' => $valid,
            'entity' => $entity,
            'form' => $form ? $form->createView() : null,
        ]
    );
}
```

Ilustración 15: Código ejecutado para generar un nuevo formulario.

3.3 Estándares de codificación.

Para un proyecto de desarrollo de software se definen estándares de codificación, debido a que un estilo de programación homogéneo permite que todos sus participantes puedan entenderlo en menos tiempo. Define la escritura y organización del código fuente, cómo deben ser declaradas las variables, las clases y los comentarios (37).

- El marco de trabajo Symfony establece el uso de los estándares de codificación PSR-0, PSR-1, PSR-2 y PSR-4. Algunos de los elementos generales contemplados en estos estándares son: Los archivos deben utilizar solamente las etiquetas <?php y <?=.

- Los archivos deben emplear solamente la codificación UTF-8 para el código PHP.

- Las constantes de las clases deben declararse en mayúsculas con guiones bajos como separadores.

- El código debe usar cuatro espacios como indentación, no tabuladores.

- La visibilidad debe estar declarada en todas las propiedades y métodos; abstracto y final deben estar declaradas antes de la visibilidad; estático debe estar declarada después de la visibilidad.

3.3.1 Estándar para clases.

Las clases se encuentran en archivos independientes que solo contendrán el código de esta. Se utiliza el estilo de codificación “UpperCamelCase”, el cual establece que los nombres inician con letra mayúscula y si poseen más de una palabra, la primera letra de estas deberá ser mayúsculas también.

Ejemplo: ConstruirFormulario.

3.3.2 Estándar para funcionalidades y atributos.

Los nombres de las funciones y los atributos deben ser descriptivos y concisos. No se usan abreviaciones. Se utiliza el estilo de codificación “lowerCamelCase”, el cual establece que los nombres inician con letra minúscula y cada nueva palabra debe iniciar con mayúscula.

Ejemplo de función: *getName().*

Ejemplo de atributo: *name.*

3.3.3 Estándar para comentarios.

Se utilizan comentarios para brindar información adicional al desarrollador acerca de la implementación de cada clase, método, propiedad o constante creada. Esto permite un mayor entendimiento del código para otros programadores y posibilita el mantenimiento y la evolución a lo largo del tiempo.

```
if (!$valid) {
    //Se ejecuta el comando de Symfony (doctrine:generate:form) para la
generación del FormType dado una Entidad (Entity).
    //Devuelve las variables ($response, $valid):
    //$response: respuesta del comando
    //$valid: verificar si fue una ejecución válida
    $response = $util->executeCommand(
        $util->getApplication($kernel),
        $util->getInputOutput($util->getArrayInputFormType($entity))
    );
    $valid = $util->fileExist('Form/'.$entity.'Type.php');
}
```

Ilustración 16: Código del método createFormTypeByEntity() de la clase DefaultController.

3.4 Pruebas de software.

El proceso de prueba se realiza de forma continua para asegurar durante todo el proceso de desarrollo, el éxito del producto. Esto permite la detección de errores en una etapa temprana.

3.4.1 Pruebas internas

Pruebas de Caja Blanca.

Para la aplicación de las pruebas de caja blanca se hizo uso de la técnica camino básico. El método del camino básico permite obtener una medida de la complejidad de un diseño procedimental, y utilizar esta medida como guía para la definición de una serie de caminos básicos de ejecución, diseñando casos de prueba que garanticen que cada camino se ejecuta al menos una vez.
Se toma de ejemplo el método *createFormTypeByEntity()*, perteneciente a la clase *DefaultController* como base para realizar la técnica del camino básico.

El método *createFormTypeByEntity ()*, consiste en generar un formulario. Para realizar esta operación el método recibe datos como el nombre y los componentes de una entidad que se crea anteriormente. Luego se pregunta si la petición fue realizada por el método POST o GET. Se genera el formulario, dando la opción al usuario de poder visualizar como quedarían los componentes en el formulario.

A continuación, se muestra el método *createFormTypeByEntity ()*:

```
public function createFormTypeByEntity(Request $request, Util $util, $entity):
Response
{
   if ($request->isMethod('GET')) { //1
      return $this->render('response/entity.html.twig', ['entity' =>
strtolower($entity)]);  //2
   }

   $entity = ucfirst(strtolower($entity)); //3
   $kernel = $this->get('kernel'); //3
   $valid = $util->fileExist('Form/'.$entity.'Type.php'); //3
   $response = 'The new '.$entity.'Type.php class file has been created'; //3

   if (!$valid) { //4
      $response = $util->executeCommand( //5
         $util->getApplication($kernel), //5
         $util->getInputOutput($util->getArrayInputFormType($entity))); //5
      $valid = $util->fileExist('Form/'.$entity.'Type.php'); //5
   }

   $this->addFlash( //6
      $valid ? 'success' : 'danger', //6
      $valid ? 'Action success' : 'Action code Error!!!. '.$entity.'Type no valid.'); //6

   $form = $valid ? $this->createForm('AppBundle\\Form\\'.$entity.'Type', null,
['data_class' => null]) : null; //7

   return $this->render('response/form.html.twig',['response' => $response,'valid'
=> $valid,'entity' => $entity,
         'form' => $form ? $form->createView() : null,]); //8
} //9
```

Ilustración 17: Código del método createFormTypeByEntity() de la clase DefaultController.

A continuación, se muestra el grafo de flujo correspondiente al código del método seleccionado:

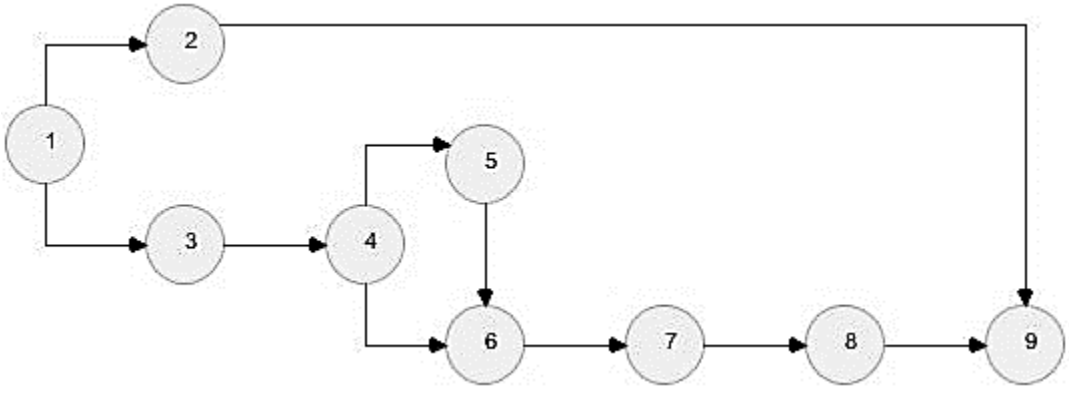


Ilustración 18: Grafo de flujo.

Complejidad ciclomática

Existen varias formas de calcular la complejidad ciclomática de un programa a partir de un grafo de flujo:

1. V(G) = A – N + 2
   A: número de aristas, N: número de nodos.
   V(G)= (10 - 9) + 2 = 3
2. V(G) = NP + 1
   NP: número de nodos predicados (nodos de los cuales parten dos o más aristas).
   V(G)= 2 + 1 = 3
3. V(G)= R
   R: número de regiones del grafo.
   V(G)= 3

Luego de calculada la complejidad ciclomática mediante las tres fórmulas se evidencia que la complejidad es 3, lo cual indica que existen 3 caminos independientes, representando este valor el mínimo número de casos de pruebas.

Los caminos independientes resultantes son: **Camino 1:** 1-2-9

Camino 2: 1-3-4-5-6-7-8-9

Camino 3: 1-3-4-6-7-8-9

Casos de prueba

A continuación, se muestra el caso de prueba de aceptación para el camino 1 en la *Tabla 9*, los demás casos de pruebas se pueden observar en el *Anexo 7*.

Tabla 9: Caso de Prueba para el camino 1.

Escenario	Descripción	Respuesta del sistema	Flujo central
EC 1.1 Verificar método.	Se verifica por cual método se envían los datos.	Si la petición fue realizada por el método GET, se envía de vuelta a la vista anterior y se termina el método.	1. Se comprueba por cual método se reciben los datos.

Análisis de los resultados

Una vez ejecutadas las pruebas de caja blanca a través de la técnica del camino básico al método createFormTypeByEntity(), en una primera iteración se detectaron un total de 2 errores en la lógica del código. Estos errores detectados fueron solucionados y se procedió a aplicar una segunda iteración donde no se detectó error alguno. De esta manera se comprobó que el flujo de trabajo de las funciones implicadas en el método es correcto y cumple con las condiciones planteadas anteriormente.

3.4.2 Pruebas de liberación.

Pruebas de caja negra.

Las pruebas de caja negra comprenden un conjunto de diferentes técnicas como, por ejemplo: Partición de Equivalencia, Análisis de Valores Límites y Grafos de Causa-Efecto. De estas técnicas se selecciona para ser utilizada la Partición de Equivalencia, la cual divide el campo de entrada en clases de datos que tienden a ejercitar determinadas funciones del software. Para la aplicación de la misma se realizaron dos diseños de casos de prueba (DCP), los cuales se basan en una evaluación de las clases de equivalencia para una condición de entrada. Una clase de equivalencia representa un conjunto de estados válidos o inválidos para condiciones de entrada (7). En los dos DCP se evalúan los cuatro requisitos funcionales. El DCP Generar Entidad describe las funcionalidades Adicionar Componente, Eliminar Componente y Generar Entidad. Mientras que el segundo DCP describe la funcionalidad Generar Formulario.

Se entregan los DCP y el software para ser evaluados y certificados por el Centro de Informatización de Entidades. Las pruebas son realizadas por el grupo de calidad del centro CEIGE[9] junto con el equipo de desarrollo. Se realizaron dos iteraciones. En la primera iteración se detectaron 3 no conformidades, siendo las 3 de aplicación, desglosadas en 1 de funcionalidad y 2 de interfaz. En una segunda iteración estas no conformidades fueron solucionadas obteniéndose resultados satisfactorios. Véase *Ilustración 19*.

[9] Centro de Informatización de Entidades.

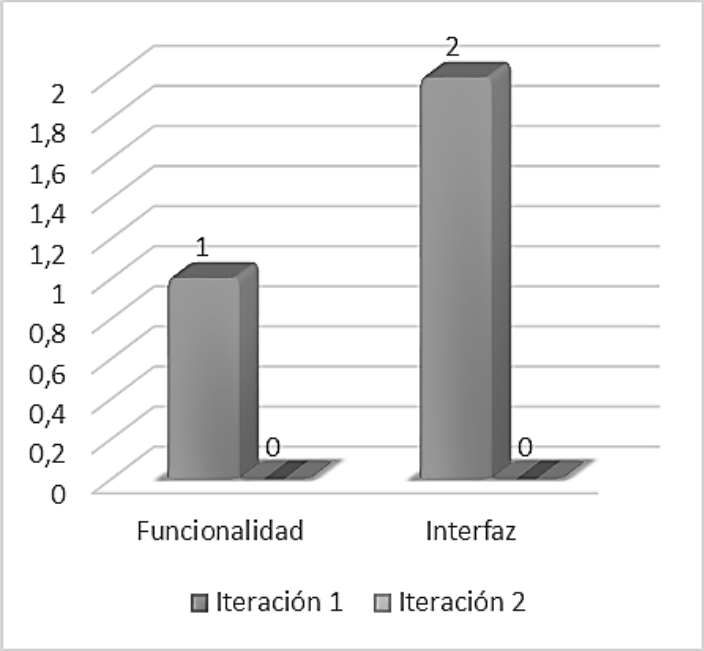


Ilustración 19: No conformidades detectadas.

3.4.3 Pruebas de aceptación.

Para realizar las pruebas de aceptación se utilizó la técnica de validación con el cliente. El cliente es quien valida si la aplicación está lista para ser utilizada por los usuarios finales. Para la validación del software se utilizaron los casos de pruebas definidos para las pruebas de liberación. En la primera iteración en la ejecución de las pruebas se mostró un resultado exitoso debido a que no se encontraron no conformidades. Terminadas estas pruebas el cliente avaló la solución con la entrega del Certificado de Aceptación del Producto, el cual se encuentra en el *Anexo 3*.

3.5 Caso de estudio.

Con el objetivo de validar la solución al problema de investigación se diseña un caso de estudio. Se desea conocer el tiempo estimado de implementación de los desarrolladores del sistema VUA en formularios web. Para esto se recoge la información registrada en los documentos de Descripción de requisitos por proceso, elaborados por la Ing. Osmaily Durán Pérez, analista del sistema VUA. En estos documentos se recoge toda la información referente a los requisitos Adicionar perfil, Registrar DM Tránsito y Registrar DM Transferencia. Para seleccionar los requisitos se tuvo en cuenta la alta complejidad de estos, medida por la cantidad de componentes que contienen. Además estos requisitos fueron desarrollados por el Ing. Yoandry Freire Pérez, quien es el miembro con menos experiencia del equipo de desarrollo.

Para el caso de estudio se cuenta con un equipo de cómputo con las siguientes prestaciones:

- Tipo de CPU: Intel Core 2 Duo CPU 2.00 GHz
- Memoria del sistema: 2 Gb de RAM

3.6 Diseño experimental.

Se utiliza en la investigación un pre-experimento para validar la propuesta de solución. Para el pre-experimento se precisa del valor del tiempo consumido por un desarrollador en la implementación de los formularios mencionados anteriormente, sin utilizar la herramienta propuesta. Este resultado será comparado con los valores obtenidos luego de que el mismo desarrollador genere los tres formularios utilizando el software propuesto. A continuación, se muestran los tiempos recogidos:

Tabla 10: Tiempos estimados desarrollador vs software.

Requisitos	**Tiempo del desarrollador sin utilizar el software (min)**	**Tiempo del desarrollador utilizando el software (min)**
Adicionar perfil	1 440	49
Registrar DM Tránsito	5 760	56
Registrar DM Transferencia	5 760	53
Total (min)	12 960	158
Total (horas)	216	2.63

Análisis de los resultados

Una vez consultado el tiempo que demora el desarrollador en implementar los formularios se obtiene un tiempo total de 12 960 minutos aproximadamente. En el proceso de generación de formularios utilizando el software, se obtiene un tiempo total estimado de 158 minutos.

La aplicación desarrollada reduce el tiempo en aproximadamente 12 802 minutos o lo que es lo mismo, 213 horas aproximadamente siendo una solución factible para ser utilizada por los desarrolladores del sistema VUA para la generación de formularios web. En el análisis de este resultado se debe tener en cuenta la

disponibilidad de recursos de hardware donde es puesta en funcionamiento la herramienta, ya que, aunque este proceso no requiere de un alto procesamiento, se debe tener al menos un equipo de cómputo con las características presentadas en el caso de uso para asegurar el correcto funcionamiento del sistema.

3.7 Conclusiones Parciales

Al finalizar el presente capítulo se arribó a las siguientes conclusiones:

- Se implementó una herramienta para el diseño y generación de formularios web, para los desarrolladores del sistema VUA
- El uso de estándares de codificación permitió proyectar una calidad en el código generado.
- Mediante la aplicación de las pruebas de software se pudo constatar que la implementación del componente funciona de manera correcta.
- En el diseño experimental se demostró que la herramienta para la generación de formularios web disminuye el tiempo que demoran los programadores del sistema VUA en implementar los formularios web.

CONCLUSIONES GENERALES

Finalizada la presente investigación se puede concluir que se desarrollaron todas las tareas a fin de cumplir los objetivos propuestos, resaltando que:

- La elaboración del marco teórico referencial de la investigación permitió el estudio del estado del arte, así como las herramientas, lenguajes y la metodología de desarrollo a emplear en la implementación.
- Durante el análisis y diseño de la solución se obtuvieron los artefactos propuestos por la metodología de desarrollo seleccionada, lo que permitió facilitar la implementación de las funcionalidades definidas.
- Se identificaron los requerimientos funcionales y no funcionales, logrando una primera visión del sistema.
- La realización de las historias de usuarios facilitó una descripción detallada de las funcionalidades del sistema.
- A partir de esta implementación se obtuvo un sistema capaz de satisfacer las necesidades del cliente.
- Las pruebas realizadas permitieron garantizar un correcto funcionamiento de la herramienta y cumplimiento de las necesidades y requisitos del cliente, avalado mediante el certificado de aceptación.
- Con la realización de las pruebas se demostró que la “Herramienta para el diseño y generación de formularios web para la Ventanilla Única Aduanera” se encuentra en correcto funcionamiento.
- La propuesta de solución disminuye el tiempo que consume un programador en implementar formularios web. Es una solución factible que se puede integrar al sistema VUA.

RECOMENDACIONES

Debido a los resultados de la investigación efectuada, a la experiencia adquirida durante la realización de este trabajo, y con el propósito de asegurar la posterior ampliación, modificación y mejora de la herramienta, se exponen a continuación algunas recomendaciones:

- Agregar componentes al generador para aumentar las opciones de diseño.
- Adicionar otras funcionalidades al sistema desarrollado para continuar perfeccionando el diseñador y generador de formularios web en próximas versiones.
- Se implemente una funcionalidad parra importar formularios que han sido generados

BILIBLIOGRAFÍA

1. **Carrera, M.** *TECNOLOGÍAS DE INFORMACIÓN Y COMUNICACIÓN.* 2015.

2. **Huidobro, José.** *Tecnologías de Información y Comunicación.* 2007.

3. **Terry, Risell y Nidia Dayana.** *Diseñador y generador de formularios web.* 2012.

4. **Chappell, Kevin.** formBuilder. [En línea] 2016. https://formbuilder.online/.

5. **Tsabba, Benny.** FormLogix. [En línea] 2005. http://www.formlogix.com/lang/esp/Create-Web-Forms.aspx.

6. **Scantron Corporation.** Scantron. [En línea] 2008. www.scantron.com.

7. **Kubessi Pérez, Malak.** *Análisis de competencias transversales referido al modelo educativo de ingeniería areonaútica en la universidad politécnica de Valencia.* 2015.

8. **Weblogs SL.** Genbeta. [En línea] 2006.

9. **Luján Mora, Sergio.** *Programación en Internet: Clientes Web.* 2001.

10. **Universidad Sevilla.** *Aplicaciones Web.* 2011.

11. **Pressman, Roger S.** *Ingeniería del Software. Un Enfoque Práctico.* 2010.

12. **Brooks, Frederick P.** *The Design of Design: Essays from a computer scientist.* s.l. : Addison-Wesley Professional, 2010. 978-0-201-36298-5.

13. **Núñez Camallea, Noel Luis y Coutin Abalo, Ronald.** *Diccionario de Informática.* s.l. : Científico-Técnica, 2006. 978-959-05-0391-7.

14. **Rodríguez Sánchez , Tamara .** *Metodología de desarrollo para la Actividad productiva UCI.* 2015.

15. **Karacostas, V. y Loucopulos, V.** *System Requirements engineerinuality which will perform effectively..* 1995. ISBN: 0-07-707843-8.

16. **Visual Paradigm.** Visual Paradigm. [En línea] 2018.

17. **Chaudhary, Mukund y Kumar, Ankur .** *PhpStorm Cookbook.* s.l. : Packt Publishing, 2014. 978-1-78217-387-8.

18. **Eguiluz, Javier.** *Desarrollo Web Ágil con Symfony 2.* 2013.

19. —. Symfony 2. *symfony.es.* [En línea] 2015. http://symfony.es.

20. **Efron, Bradley y Tibshirani, Robert J.** *An introduction to the Bootstrap.* s.l. : Chapman & Hall/CRC, 1998. ISBN: 0-412-04231-2.

21. **Efron , Bradley .** *Bootstrap Methods: Another Look at the Jackknife.* s.l. : Springer-Verlag, 1977. ISBN: 978-0-387-94039-7.

22. **Mateu, Carles.** *Desarrollo de aplicaciones web.* 2004.

23. **Vogelgesang, Kai y Seidler , Kay.** *Das XAMPP-Handbuch: Der offizielle Leitfaden zu Einsatz und Programmierung.* s.l. : Addison-Wesley, 1970. ISBN: 978-84-9732-813-5.

24. **Aggarwal, Gaurav, y otros.** *An analysis of private browsing modes in modern browsers.* 2010. ISBN: 888-7-6666-5555-4.

25. **Rumbaugh, James, Jacobson, Ivar y Booch , Grady.** *El lenguaje Unificado de Modelado. Manual de Referencia.* 2000. ISBN: 84-7829-037-0.

26. **Eguíluz Pérez, Javier.** *Introducción a JavaScript.* 2009.

27. **Alvarez, Miguel Angel.** *Manual de jQuery.* 2010. ISBN: 978-1499321418.

28. **Chaffer, Jonathan.** *Learning JQuery 1.3: Better Interaction and Web Development with Simple JavaScript Techniques.* s.l. : Packt Publishing Ltd, 2009. 978-1-84719-671-2.

29. **Eslava Muñoz, Vicente Javier.** *El nuevo PHP.Conceptos avanzados.* 2013. 978-84-686-4434--9.

30. **Rodas Hinostroza, Raul .** *Características de PHP.* 2014.

31. **Hogan, Brian P.** *HTML5 & CCS3.* 2010. ISBN-10: 1-934356-68-9.

32. **Briggs, Owen.** *Cascading Style Sheets.* 2003.

33. **Chave, M. A.** *La ingeniería de requerimientos y su importancia en el desarrollo de proyectos de software.* 2007. ISSN: 1409-4746.

34. **Escalona, María José y Koch, Nora.** *Ingeniería de Requisitos en Aplicaciones para la Web. Un estudio comparativo.* 2002.

35. **Sommerville, Ian.** *Software Engineering.* 2011. 0-13-703515-2.

36. **Cohn, Mike.** *User Stories Applied.* s.l. : Addison-Wesley Professional, 2004. ISBN: 0-321-20568-5.

37. **Christopher Alexander, Ishikawa, Sara y Silverstein, Murray.** *A Pattern Language.* 1977.

38. **Almeira, Adriana y Pérez, Sandra V.** *Arquitectura de Software: Estilos y Patrones.* 2007.

39. **Larman, Craig.** *ML y Patrones. Introducción al análisis y diseño orientado a objetos.* Mexico : Pablo Eduardo Roig Vázquez, 1999. ISBN: 970-17-0261-1.

40. **Christiansson, Benneth.** *GoF Design Patterns-with examples using Java and UML2.* 2008.

41. **Pressman, Roger S.** *Ingeniería del Software. Un Enfoque Práctico.* 2010.

42. **Desarrollo Web.** Métricas de Software. [En línea] 2015. http://www.desarrolloweb.com/articulos-copyleft/articulo-metricas-de-software.html..

43. **Kidd, Jeff y Lorenz , Mark .** *Object-oriented software metrics: a practical guide. .* 1994. 0-13-179292-X .

44. **Eguiluz, Javier.** *Manual de Symfony 2: Estándares de codificación.* 2014.

46. **Porebski, Bartosz, Przystalski, Karol y Nowak, Leszek.** *Building PHP Applications with Symfony, CakePHP, and Zend Framework.* 2011.

47. *Estándares de Diseño Web.* **Hernández Claro, Rosendo L.** 2010.

48. **Guerra, César Arturo.** *Obtención de Requerimientos. Técnicas y Estrategia.* 2018.

49. **McCall, J. A., Richards, P. K. y Walters, G. F.** *Factors in Software Quality.* 1977.

50. *Modelo de calidad para evaluar el software desarrollado en el centro de investigación aplicada y desarrollo en tecnologías de información CIADTI.* **Estévez, Y. S. y Esteban, L. A.** 2014.

51. *MODELOS Y MÉTRICAS PARA EVALUAR CALIDAD DE SOFTWARE.* **Estayno, Marcelo, Dapozo, Gladys y Cuenca Pletch , Liliana .** 2013.

52. **Rodríguez Sánchez, Tamara.** *Metodología de desarrollo para la Actividad productiva de la UCI.* 2014.

53. **Konsynski, Benn R., y otros.** *An Integrated Development Environment for Information Systems.* 1984.

54. *Arquitectura y Diseño de Sistemas Web Modernos.* **Castejon Garrido, J. S.** 2004.

55. **Cárdenas Luque, Lola.** *Curso de JavaScript.* 2001.

56. **Ayose Castillo, Alberto.** *Curso de Programación Web.* 2015.

57. **Steinberg, Daniel H. y Palmer, Daniel W.** *Extreme Software Engineering.* s.l. : Pearson Education, Inc., 2003. 0-13-047381-2.

58. **Leal Faunde, Edgar.** *Guía para asegurar la portabilidad en los sistemas informáticos de Gobierno Electrónico.* 2012.

59. **Addy Osmany.** *Learning JavaScript Design Patterns.* 2012.

60. **Fowler, Martin y Beck, Kent.** *Refactoring: Improving the Design of Existing Code.* s.l. : Addison-Wesley Professional, 1999. 978-0-201-48567-7.

61. *Tecnologías de información y comunicación.* **Alamo, Oscar Nicolás.** 2009.

62. *TECNOLOGÍAS DE INFORMACIÓN Y COMUNICACIÓN.* **Carrera, M.** 2015.

63. *Tecnologías de Información y comunicación.* **Ley General de Telecomunicaciones.** 2008.

64. **Potencier, Fabien y Zaninotto, François.** *The Definitive Guide to symfony.* 2010. 1-59059-786-9.

65. **Arturo Guerra, César.** CG. *Obtención de Requerimientos. Técnicas y Estrategia.* [En línea] 2018.

66. **Herrera, José Adolfo.** Administración de La Empresa Constructora. [aut. libro] José Adolfo Herrera. *Las fases de un proyecto.* 2012.

67. **Herramienta para modelado UML.** Paraiso Linux. *Herramienta para modelado UML.* [En línea] 2016.

68. **FireOS.** Servidor Web Apache: ¿Qué es, cuáles son sus caracteristicas y para qué se utiliza? [En línea] Ibrugor, 8 de marzo de 2017. http://www.ibrugor.com/blog/apache-http-server- que-es-como-funciona-y-para-que-sirve/..

69. **Sánchez Morales, Martín.** *Manual de desarrollo web basado en ejercicios y supuestos prácticos.* 2012. ISBN: 978-1-291-03777-7.

70. **Hidalgo, Saltos y Evangelina, Johanna .** *Desarrollo de una aplicación web que permita llevar el registro y control integral de la información del alumnado en el Centro de Educación Inicial "Globitos de colores".* 2017.

71. **Tsabba, Benny y Weinberg, Amy .** Formlogix. [En línea] 2018. http://www.formlogix.com/lang/esp/CreateWebForms.aspx?LangId=3.

72. **SurveyMonkey.** [En línea] 2018. https://www.wufoo.com/.

73. **Dvorski, Dalibor D.** *Installing, configuring, and developing with XAMPP.* 2007. ISBN: 978-987-3832-04-8.

ANEXOS

Anexo 1.

Entrevista realizada a uno de los desarrolladores del sistema VUA en la facultad 3, el Ing. Leansi Vega Rouco.

Entrevista

Objetivo: comprobar cómo se desarrollan los formularios web para el sistema VUA y cuáles son los componentes utilizados en su desarrollo.

Compañero.
Se necesita su colaboración para la realización de esta investigación.

Datos generales.

1. Nombre y Apellidos
2. Rol que desempeña

Aspectos a entrevistar

1. ¿Cuánto se tardan en implementar un formulario estándar en el sistema?
2. ¿Influye el tiempo de implementación de estos formularios en el tiempo de finalización del sistema?
3. ¿Existe alguna herramienta que pueda reducir este tiempo de implementación?
4. En caso de existir, ¿Se puede vincular al sistema VUA?, en caso de ser negativa la respuesta, ¿Por qué?
5. ¿Qué herramientas hay que tener en cuenta para desarrollar un generador de formulario que pueda vincularse al sistema?

Anexo 2.

Casos de prueba de aceptación.

Tabla 11: Caso de prueba de aceptación Generar Entidad.

Escenario	Descripción	Nombre de la entidad	Respuesta del sistema	Flujo central
EC 3.1 Generar entidad.	Se genera correctamente una entidad que no se ha introducido anteriormente en el sistema.	V Prueba	El sistema genera la entidad y muestra el mensaje “Action success”. Se muestra la ruta donde se encuentra la nueva entidad. Se muestra un botón para generar el formulario correspondiente.	1. Arrastrar al menos un componente del panel de la izquierda hacia el centro de la entidad. 2. Asignarle un nombre a la entidad de no menos de cuatro caracteres, todos alfabéticos. 3. Seleccionar el botón “Generate Symfony Entity::class”.
EC 3.2 Datos incompletos.	Se muestra un mensaje indicando que no se han introducido los datos obligatorios.	I Vacío	Muestra el mensaje “This field is required” debajo del campo “Entity name”.	1. Seleccionar el botón “Generate Symfony Entity::class”.
EC 3.3 Datos incorrectos.	Se muestra un mensaje de error indicando que se han introducido datos incorrectos.	I 12	Muestra el mensaje “Please create a valid form. Do not enter less than 1 component”. debajo del campo “Entity name” indicando porque la entrada es incorrecta. Los componentes se mantienen para que el usuario los modifique si lo desea.	1. Asignarle un nombre a la entidad de menos de cuatro caracteres, o valores numéricos o alfanuméricos. 2. Seleccionar el botón “Generate Symfony Entity::class”.

EC 3.4 Existe una entidad con el mismo nombre.	Se muestra un mensaje de error.	I Prueba	Muestra el mensaje “is already used ”, precedido de la ruta donde se encuentra la entidad.	1. Asignarle un nombre a la entidad que se encuentre en uso. 2. Seleccionar el botón “Generate Symfony Entity::class”.

Tabla 12: Caso de prueba de aceptación Generar Formulario.

Escenario	**Descripción**	**Nombre de la entidad**	**Respuesta del sistema**	**Flujo central**
EC 4.1 Generar formulario.	Se genera correctamente un formulario.	V Prueba	El sistema genera el formulario y muestra el mensaje “Action success”. Se muestra la ruta donde se encuentra el nuevo formulario. Se muestra un botón para ver una vista previa de los componentes del nuevo formulario.	1. Seleccionar el botón “Generate (nombre_entidad)Type::class”.

Anexo 3.

Certificado de Aceptación del Producto.

ACTA DE ACEPTACIÓN

En cumplimiento de los objetivos para la ejecución del trabajo de culminación de estudios que lleva por título: Herramienta para el diseño y generación de formularios web para el proyecto Ventanilla Única Aduanera, por parte de los estudiantes Osniel Ramos Avalos y Oslen Alvarez Carmona, se hace entrega a los clientes, Ing. Leduan Flores Riera e Ing. Leansi Vega Rouco, el producto:

- Herramienta para el diseño y generación de formularios web para el proyecto Ventanilla Única Aduanera

Luego de haber revisado y probado el producto entregado se determina que: cumple con los requisitos funcionales pactados entre las partes y se le da solución a la problemática planteda, por tanto es aceptado el software desarrollado, validado y liberado.

Comentarios

Entrega Desarrolladores	**Recibe** Clientes (Tutores)
Nombre y apellidos: Osniel Ramos Avalos	**Nombre y apellidos:** Ing. Leduan Flores Riera
Cargo: Estudiante	**Cargo:** Especialista "B" en Ciencias Informáticas
Nombre y apellidos: Oslen Alvarez Carmona	**Nombre y apellidos:** Ing. Leansi Vega Rouco
Cargo: Estudiante	**Cargo:** RGA

CEIGE

Fecha: 06/06/2018

Ilustración 20: Acta de aceptación.

Anexo 4.

Evaluación de requisitos

Tabla 13: Evaluación de requisitos.

Requisitos	Interfaces				Características					Complejidad
	H	**E**	**P**	**C**	**D.C**	**F.I**	**R.V**	**G.R**	**L.N**	
RF_Adicionar componente.	M	No	B	No	No	M	B	A	A	Media
RF_Eliminar componente.	M	No	B	No	No	M	M	A	A	Media
RF_Generar entidad.	A	A	A	No	A	A	A	A	A	Alta
RF_Generar formulario.	A	A	A	No	A	A	A	A	A	Alta

Leyenda:

H: Humanos.

E: Equipos.

P: Programación.

C: Comunicación.

D.C: Diferentes comportamientos.

F.I: Formas de inicialización.

R.V: restricciones de validación.

G.R: Grado de reutilización.

L.N: Lógica de negocio.

Anexo 5.

Interfaz de la aplicación.

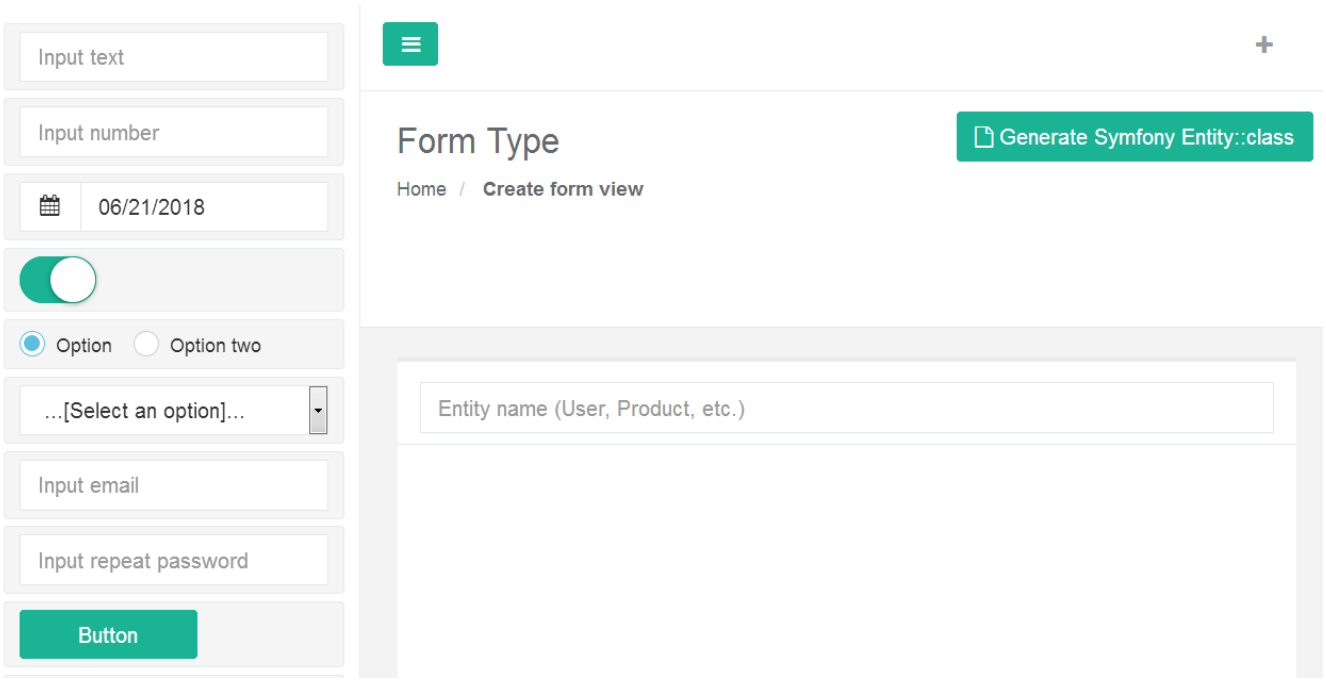


Ilustración 21: Interfaz de usuario.

Anexo 6.

Historias de usuario.

Tabla 14: Historia de usuario de Eliminar Componente.

HISTORIA DE USUARIO	
Número: HU-2	**Nombre historia de usuario:** Eliminar Componente.
Usuario: Osniel Ramos	**Iteración asignada:** 2
Prioridad en negocio: Alta	**Puntos estimados:** 1
Riesgo en desarrollo: Medio	**Puntos reales:** 1
Descripción: Permite al desarrollador eliminar un componente existente en la entidad.	
Observaciones: Los elementos son; jTextField, RadioButton, CheckBox, Select.	
Interfaz de usuario:	

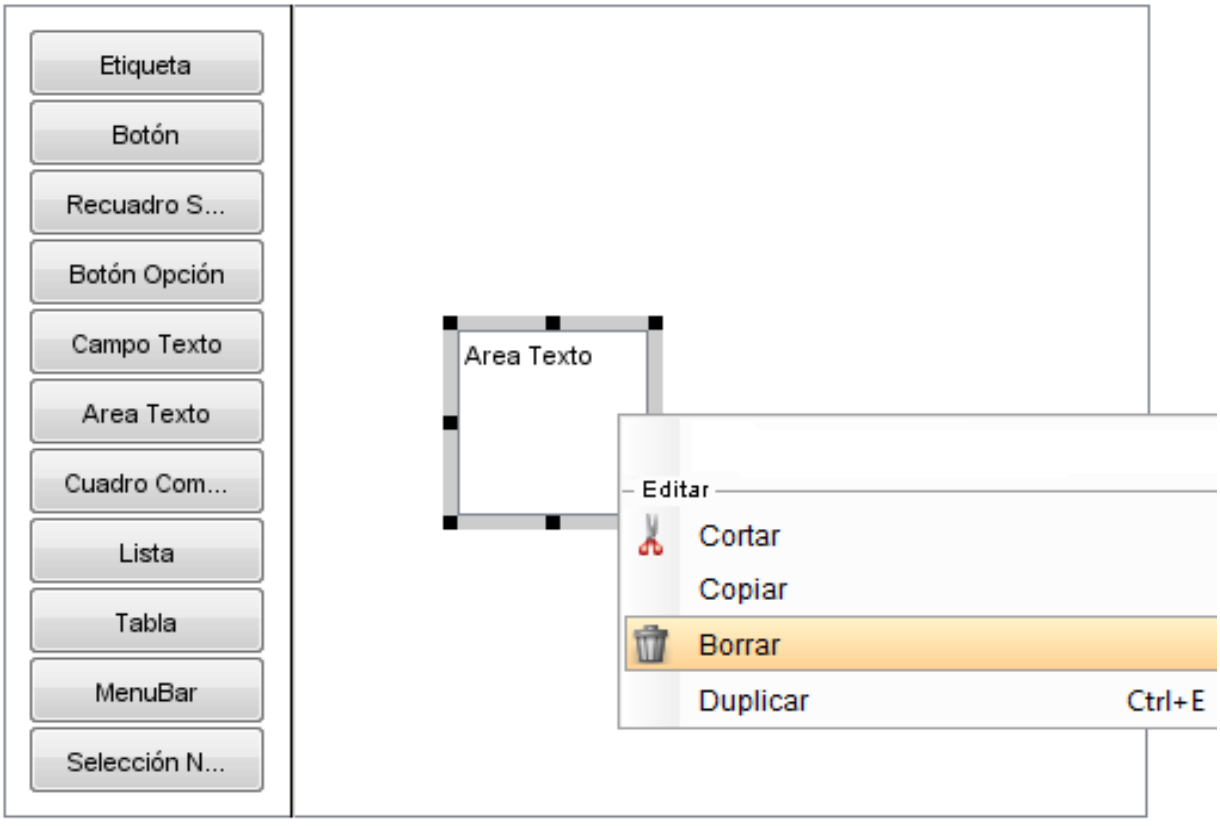


Tabla 15: Historia de usuario de Generar Entidad.

HISTORIA DE USUARIO	
Número: HU-3	**Nombre historia de usuario:** Generar Entidad.
Usuario: Osniel Ramos	**Iteración asignada:** 1
Prioridad en negocio: Alta	**Puntos estimados:** 3
Riesgo en desarrollo: Alto	**Puntos reales:** 3
Descripción: Permite al desarrollador generar una entidad previa a la creación del formulario.	
Observaciones: La entidad debe tener al menos un componente y un nombre asignado.	
Interfaz de usuario:	

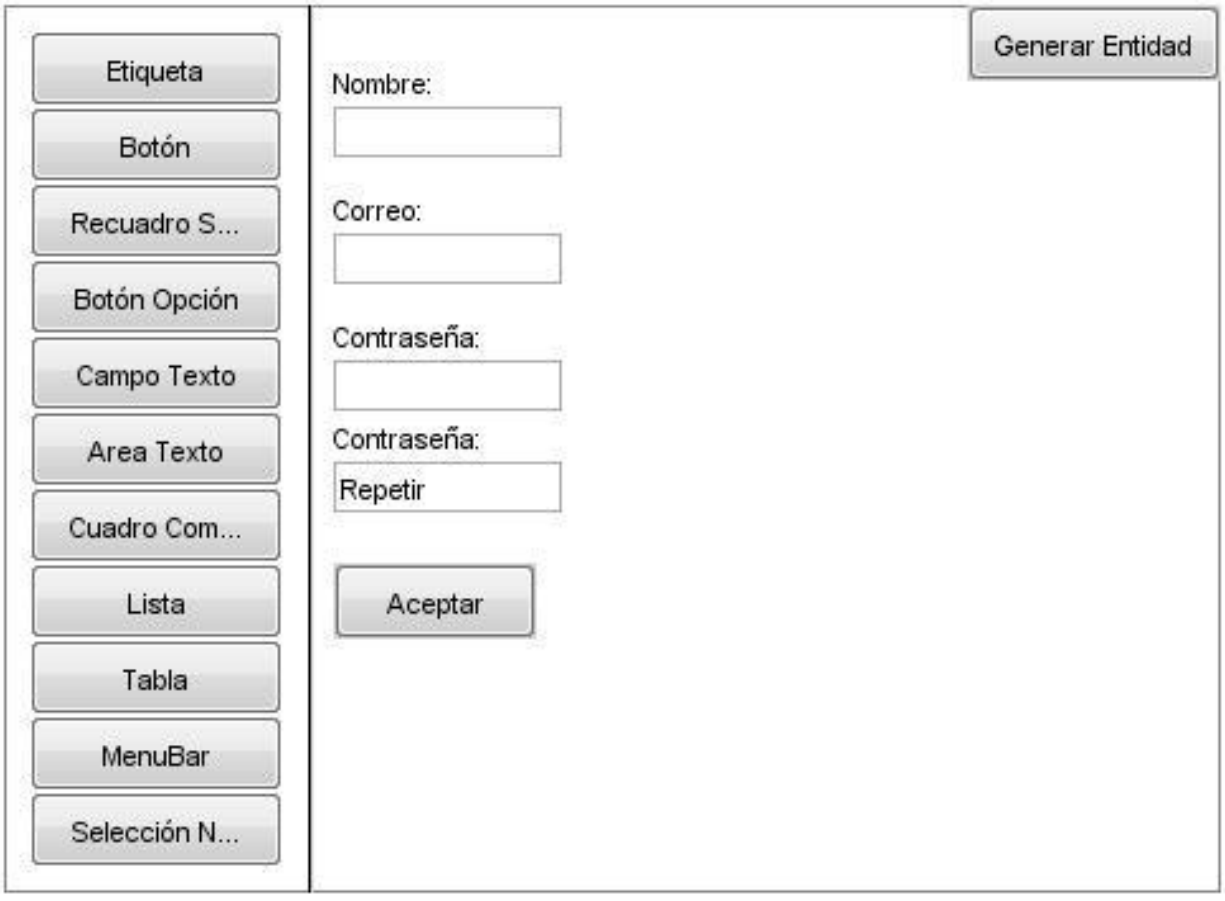
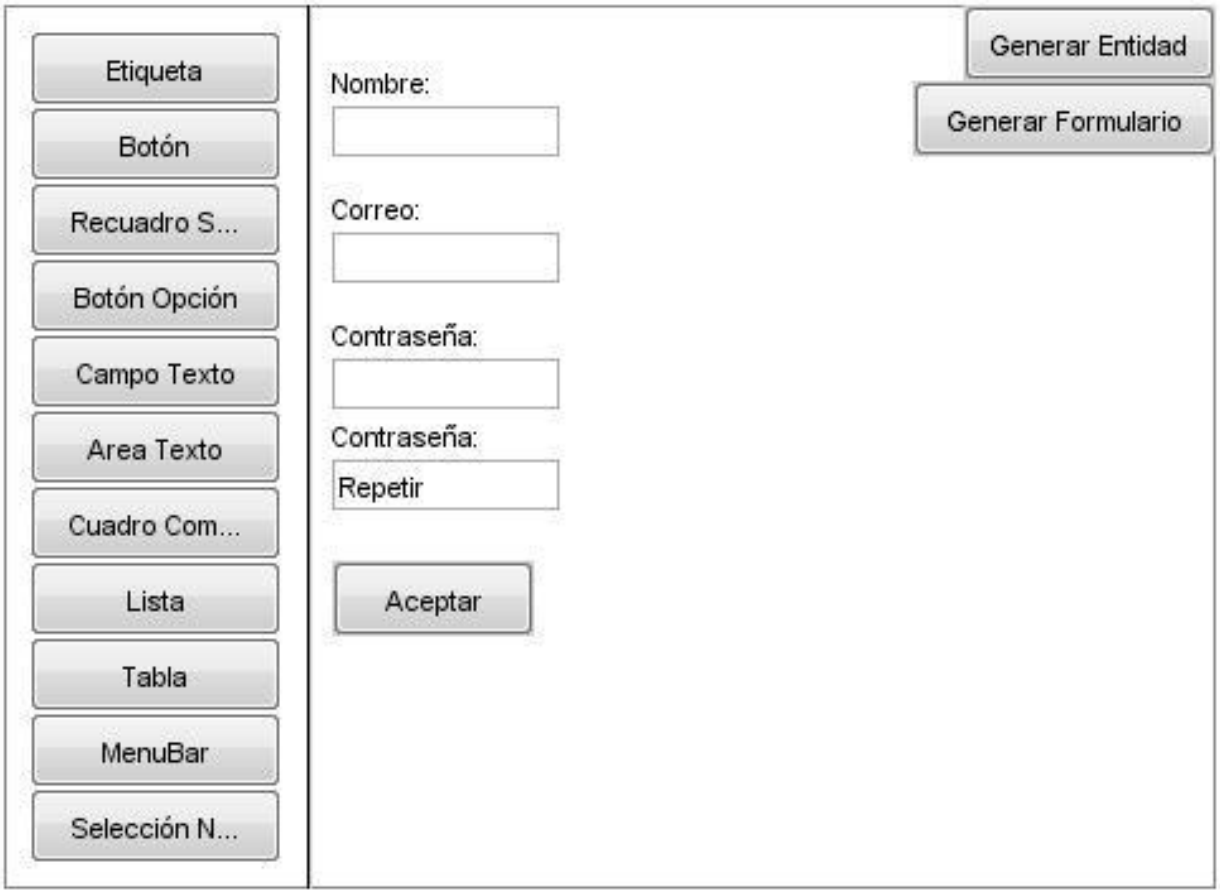


Tabla 16: Historia de usuario de Generar Formulario.

HISTORIA DE USUARIO	
Número: HU-4	**Nombre historia de usuario:** Generar Formulario.
Usuario: Osniel Ramos	**Iteración asignada:** 2
Prioridad en negocio: Alta	**Puntos estimados:** 3
Riesgo en desarrollo: Alto	**Puntos reales:**
Descripción: Permite al desarrollador generar un formulario a partir de la entidad creada.	
Observaciones: Debe haberse creado una entidad anteriormente.	
Interfaz de usuario:	



Anexo 7.

Casos de prueba de aceptación para los caminos de las pruebas de caja blanca.

Tabla 17: Caso de Prueba para el camino 2.

Escenario	Descripción	Respuesta del sistema	Flujo central
EC 1.1 Generación de la entidad.	Se verifica por cual método se envían los datos y si el formulario es válido se genera.	Si la petición fue realizada por el método POST, se genera el formulario a partir de la entidad recibida.	1. Verificar si los datos se envían por el método GET o por el método POST. 2. Verificar si el formularios es válido.

Tabla 18: Caso de Prueba para el camino 3.

Escenario	Descripción	Respuesta del sistema	Flujo central
EC 1.1 Formulario invalido.	Verificar la validez del formulario	Si la petición fue realizada por el método POST, no se genera el formulario y se muestra un mensaje informando el error.	1. Verificar si los datos se envían por el método GET o por el método POST. 2. Verificar si el formularios es válido.