UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

FACULTAD 3

Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas

Generador de interfaces en Angular JS para Symfony 3

Autora:

Claudia Rojas Maxam

Tutores:

Ing. Julio Cesar Ocaña Bermúdez

Ing. Dannel Jiménez Torres

Co-tutor

MSc. Ana Marys García Rodríguez

La Habana, junio de 2017

"Año 59 de la Revolución"

DECLARACIÓN DE AUTORÍA

Declaro ser el autor de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

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

_____	_____
Claudia Rojas Maxam Autora	Ing. Julio Cesar Ocaña Bermúdez Tutor
_____	_____
Ing. Dannel Jiménez Torres Tutor	MSc. Ana Marys García Rodríguez Co-Tutor

"Siga adelante sin dar un solo paso atrás ni para coger impulso"

Agradecimientos

Aquí termina una jornada de muchos esfuerzos, desvelos, fiestas y hasta lágrimas que no puedo pasar por alto sin agradecerle a tantas personas que junto a mi disfrutan este momento.

Le agradezco a la Revolución y a Fidel Castro por esta maravillosa universidad que hace cinco años me abrió sus puertas y hoy me ve crecer como toda una profesional.

A mi madre por todos los momentos de desvelo, sus consejos y la confianza que siempre depuso en mí. A mi padre que a pesar de no estar siempre en todos los momentos de mi vida me ha inculcado magníficos valores.

Agradezco a mis abuelas las que me han dado todo su amor y me han defendido de la furia de mamá.

Agradezco a toda mi familia, un grupo de personas maravillosas con las que se puede contar en todo momento.

Le agradezco a otra parte de mi familia: Ofelia, Laura, Eloy y Elio que me abrieron las puertas de su casa haciéndome sentir como en la mía propia.

En el proceso de mi formación como profesional, tengo que agradecerles también a todos los profesores que han pasado por cada una de las etapas de mi vida, que más que profesores son también familia.

En mi paso por la universidad le agradezco a cada uno de los profesores de nuestra Facultad Regional de Ciego de Ávila, a los colegas que por miedo no se sumaron a este grupo de locos que hoy formo parte pero ya como ingeniera.

Le agradezco a la UCI por permitirme conocer maravillosas personas que integran el colectivo de profesores y estudiantes de la Facultad 3, en especial a mis tutores por la paciencia que me han dedicado y sus maravillosos consejos.

A mi familia FEU, con la que he compartido tantos buenos y locos momentos que solo son posibles gracias a ellos.

A Yoiler y Danay que abrieron las puertas de su casa y me han ayudado en todo momento.

A las nuevas y viejas amistades con las que me he encontrado en estos últimos años, a mi niña malcriada Arlene, y hoy me declaro culpable por muchas de sus malacrianzas, pero sobre todo sus consejos y hasta por la marca de su plancha de pelo. A Yanser por su amistad y el cariño que siempre me ha tenido.

Le agradezco a Rolando por considerarme como una amiga de lo que voy a estar siempre orgullosa. Agradecerle también a Daimary, Pedro Pablo, Rafael, Camilo, Alejandro, Julio, Félix, Dorita, Lisandra, Claudia Rafael, Ortelio, Arlene Padrón, Yoandry, Leyriel, Yania, Verónica y cada una de las personas que han dejado una huella inolvidable en mi vida.

¡A todos muchas gracias!

Dedicatoria

A mi madre por su dedicación, su esfuerzo y su infinito amor.

Este triunfo hoy es de las dos.

RESUMEN

La interfaz de usuario es uno de los componentes importantes de cualquier sistema computacional funcionando como vínculo entre el humano y la máquina. Diseñar interfaces que satisfagan correctamente las necesidades del usuario no es sencillo, pues no siempre los componentes toman la alineación requerida en todos los navegadores web. El objetivo del presente trabajo es desarrollar un generador de interfaces de usuario mediante el uso de entidades de Symfony 3 en el marco de trabajo Angular JS.

Para lograrlo se desarrolló una aplicación web utilizando como guía la metodología de desarrollo el Proceso Unificado Ágil con la versión establecida por la Universidad de las Ciencias Informáticas. Se realizó un estudio de las herramientas generadoras de código llegando a la conclusión que no cumplen con las necesidades del departamento de Desarrollo de Componente, por lo que fue necesario desarrollar una nueva solución, empleándose como lenguaje de programación PHP del lado del servidor y del lado del cliente HTML, JavaScript y CCS3.

El funcionamiento de la herramienta desarrollada fue verificado mediante la aplicación de pruebas de caja negra. Se realizó la validación de la investigación mediante la realización de pruebas de aceptación y la aplicación de la técnica de Iadov, donde se evidenció un alto nivel de satisfacción por parte del cliente.

Palabras claves: entidades, generador de interfaces, herramienta, interfaces gráficas de usuario

Índice

ÍNDICE

ÍNDICE

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

INTRODUCCIÓN

Las Tecnologías de la Información y las Comunicaciones (TIC) han transformado todas las actividades del hombre en la sociedad moderna. La industria cubana del software inmersa en el desarrollo del modelo socialista cubano, cuya función está asociada a informatizar la sociedad desde tres aristas importantes: el desarrollo de la industria de software nacional, las transformaciones de procesos en las entidades para asumir su informatización y el soporte necesario para su mantenimiento. Estas necesidades están en concordancia con el nivel alcanzado en la informatización de la sociedad, los objetivos que se propone el país, las tendencias internacionales y los problemas profesionales actuales y futuros (Universidad de las Ciencias Informáticas 2014).

La Universidad de las Ciencias Informáticas (UCI) surge con el objetivo de darle cumplimiento a las necesidades surgidas en cada una de estas aristas. Esta institución cuenta con una serie de centros productivos para el desarrollo de software, entre estos se encuentra el Centro de Informatización de Entidades (CEIGE) de la Facultad 3. En dicho centro, se desarrollan aplicaciones haciendo uso de tecnologías web, en las cuales la interfaz de usuario (IU) es uno de los componentes importantes, pues funciona como vínculo entre el humano y la máquina. Diseñar interfaces que satisfagan correctamente las necesidades del usuario no es sencillo y por otro lado, posee costos de investigación y aprendizaje de cada tecnología a utilizar (Santana y Coni 2011).

En la actualidad, existen diversas herramientas para generar código a partir de un modelo, pero en su gran mayoría no abarcan la generación de IU. Estas herramientas no ofrecen la posibilidad de definir en el modelo ninguna característica de presentación ni de comportamiento de las IU, por ejemplo, definir el color y posición de un botón, ni indicar que acción realizar al perder el control (Santana y Coni 2011).

En el caso del marco de trabajo Symfony que genera su propia IU, sólo crea la típica interfaz tipo CRUD [1] que contiene elementos comunes en las interfaces gráficas por ejemplo: los botones, cuadrículas y paneles.

Entre las herramientas que se emplean, para el desarrollo de estas interfaces visuales en las aplicaciones web, se encuentra Angular JS, el cual es un marco de trabajo estructural para el desarrollo de aplicaciones web dinámicas, que permite utilizar HTML[2] como lenguaje de plantillas y le permite extender la sintaxis de HTML para la creación de nuevos componentes visuales. Al hacer un producto de software utilizando este marco de trabajo se debe definir una estructura para construir cada una de las interfaces gráficas y modificar el código fuente de la aplicación, lo que implica un tiempo de desarrollo y conocimientos previos de la tecnología. Para la construcción de estas interfaces los programadores necesitan llevar el diseño realizado del papel al sistema web, proceso que se torna

[1] Crear-Buscar-Modificar-Eliminar (Create- Retrieve -Update-Delete)

[2] Lenguaje de Macado de Hipertextos (del inglés HyperText Markup Language)

tedioso y complicado, debido a que no siempre los componentes toman la alineación requerida en todos los navegadores web. Para evitar estos problemas los desarrolladores dedican un valioso tiempo de trabajo tratando de lograr una alineación perfecta de los componentes, así como la compatibilidad con la mayoría de los navegadores web (Téllez 2014).

Con el fin de dar al usuario el control sobre la aplicación y formar las bases para los principios del diseño, el programador traza líneas de trabajo para obtener un producto con calidad y toma como punto de partida las reglas de oro de la interfaz de usuario que son definidas por (Pressman 2010).

En CEIGE, el proceso de desarrollo de software tiene como objetivo principal concluir los proyectos en la fecha acordada con el cliente y con la calidad requerida. En repetidas ocasiones se incumple con dicho acuerdo, pues en la mayoría de los proyectos el mayor tiempo de desarrollo recae en los componentes que estos contienen, y actualmente no son reutilizados a pesar de la similitud que hay entre ellos. Para erradicar este problema se han tomado medidas como: perfeccionar los procesos y metodologías de software, capacitar a los recursos humanos y utilizar las tecnologías y marcos de trabajo que están en constante desarrollo y perfeccionamiento, pero aun así este problema persiste.

A partir de una encuesta aplicada a una muestra aleatoria de 30 desarrolladores que utilizan el marco de trabajo Angular JS para el desarrollo de interfaces gráficas (ver anexo 1), con el objetivo de obtener el tiempo promedio que demoran en el desarrollo de las interfaces de un CRUD; se obtuvo como resultado que sólo el 75% de los programadores del centro que utilizan este marco de trabajo han desarrollado interfaces de un CRUD y de ellos el 60% considera que la creación de estas interfaces es compleja. En la actualidad a pesar de que el 35% de los profesionales se demoran de 1 a 3 horas en el proceso de desarrollo de las interfaces de un CRUD, todavía existe un 60% de estos que necesitan entre 1 y 3 días. A raíz de la encuesta realizada se pudo corroborar que más del 50% del tiempo empleado por los desarrolladores con este marco de trabajo en el centro, es utilizado en la creación de estas interfaces; convirtiéndose en uno de los factores que provoca el aumento del tiempo planificado para el proyecto.

A partir de lo antes expuesto y considerando que se necesita obtener aplicaciones en el menor tiempo posible y con una alineación correcta de los componentes que tienen implícitos, se define como **problema de la investigación**: ¿Cómo desarrollar interfaces gráficas en el marco de trabajo Angular JS que permita disminuir el tiempo de desarrollo?

En el marco de la investigación se define como **objeto de estudio**: generación de interfaces gráficas de usuario, enmarcado en el **campo de acción**: generación de interfaces gráficas de usuario empleando entidades de Symfony 3.

Para dar solución al problema identificado, se traza como **objetivo general** de la investigación: desarrollar un generador de interfaces de usuario mediante el uso de entidades de Symfony 3 en el

marco de trabajo Angular JS, de manera que contribuya a disminuir el tiempo de desarrollo de las interfaces gráficas de usuario.

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

1. Construir el marco teórico referencial de la investigación asociado a la generación de interfaces gráficas de usuario.
2. Modelar el análisis y diseño del generador de interfaces gráficas de usuario para la generación de código en Angular JS mediante el uso de las entidades de Symfony 3.
3. Implementar un generador para la construcción de interfaces gráficas de usuarios en Angular JS mediante el uso de las entidades de Symfony 3.
4. Verificar el correcto funcionamiento del generador de interfaces gráficas de usuarios a través de pruebas unitarias.
5. Validar la solución desarrollada mediante la aplicación de pruebas de aceptación y la técnica de Iadov.

Para la realización de la investigación se emplearon los siguientes **métodos científicos**:

Teóricos:

- **Histórico-lógico**: se empleó para hacer un análisis de las herramientas existentes en la actualidad que generan interfaces de usuario con el objetivo de determinar si existen entre ellas características similares que puedan ser adaptadas a la solución que se propone.
- **Analítico-sintético**: este método se empleó para descomponer el problema e identificar lo que trae consigo que los desarrolladores no cuenten una herramienta para generar interfaces de usuario en aras de disminuir el tiempo de desarrollo.

Empíricos:

- **Análisis documental:** este método se empleó en el proceso de obtención de requisitos funcionales.
- **Encuesta:** permitió mediante un cuestionario pre-elaborado, obtener información sobre el tiempo real que demoran los desarrolladores en realizar una interfaz con Angular JS.

El presente trabajo está estructurado en **tres capítulos**:

En el **Capítulo I**: “Marco Teórico Referencial”, se hace referencia a los principales conceptos teóricos existentes dentro del objeto de estudio, se analizan las características de los generadores de código y su funcionamiento. Además, se hace un análisis de las herramientas y tecnologías utilizadas en el proceso de desarrollo y el impacto de la herramienta en el proceso de informatización de la sociedad.

En el **Capítulo II**: “Propuesta de solución”, se llevan a la práctica los conocimientos adquiridos durante el estudio teórico realizado. También se fundamenta la propuesta de solución, la que da cumplimiento a los objetivos trazados. Esta incluye el desarrollo de un generador de código que dé solución al objetivo de la investigación. Además, se brinda una descripción clara del funcionamiento de la herramienta, se especifican los requisitos y se ejemplifican cada uno de los patrones y estándares de codificación utilizados.

En el **Capítulo III**: para la “Validación de la solución”, se aplican las pruebas de caja blanca haciendo uso del marco de trabajo PHPUnit, al igual que las pruebas de caja negra y las de aceptación para validar el funcionamiento de la herramienta. En el proceso de validación de la investigación desarrollada se aplicó la técnica de Iadov para determinar el índice de satisfacción grupal de los usuarios.

CAPÍTULO 1. MARCO TEÓRICO REFERENCIAL

1.1 Introducción

En el presente capítulo se define el marco teórico referencial de la investigación. Se hace un estudio de los conceptos asociados al dominio del problema y se estudia el estado de arte de los generadores de código existentes en la actualidad. Se describen las características, ventajas y desventajas de dichos generadores, al igual que: sus tipologías, funcionamiento y las técnicas existentes en el proceso de generación de código. Se define la metodología, las tecnologías, herramientas y técnicas que serán empleadas en el proceso de desarrollo de software. Se plantea como estrategia de validación y verificación las métricas para validar diseño.

1.2 Conceptos asociados a la investigación

*"La **generación de código** proporciona sistemas de rectificación automática que en líneas general permiten ahorrar grandes cantidades de recursos: esfuerzo, tiempo y dinero"* (Moreno 2003b).

"*Un **generador de código** es una herramienta capaz de generar código de forma automática. Son programas utilizados para crear otros programas en un menor tiempo, puesto que generan código que está listo para ser compilado o interpretado. Estas herramientas simplifican el trabajo de los programadores y reducen considerablemente el tiempo de desarrollo*" (Chang y Carbonell 2013).

Con la **"*generación automática,*** *se pretende que un generador proporcione todo el código necesario en un lenguaje de tercera generación listo para compilar (o interpretar) y ejecutar sin retoques adicionales. De este modo, se pretende reducir al máximo el retoque manual del código por parte de programadores"* (Moreno 2003a).

Mediante el generador de código se obtiene el *"**código fuente** definido como un conjunto de acciones, llamadas instrucciones, que permitirá dar órdenes al ordenador para operar el programa. Dependiendo del código fuente, el ordenador realiza varias acciones, como abrir un menú, iniciar una aplicación y efectuar búsquedas"* (Moseley 2007).

*"Las **interfaces gráficas de usuario** (IGU) se definen como elementos gráficos que comunican al usuario con un sistema o estructura"* (Moreno y Elena 2012).

Descomponiendo un sistema en capas lógicas según su función, se puede ver a la interfaz de usuario como la capa lógica encargada de dar soporte al diálogo con el usuario. Sus funciones, son básicamente dos:

Entrada: adquirir las órdenes, información y comandos expresados por el usuario a través de diversos dispositivos de interacción.

Salida: presentar resultados, retroalimentación y cooperar para facilitarle al usuario la realización de las tareas que pretende resolver el sistema.

La autora en el marco de la investigación hace un acercamiento al significado de interfaz gráfica de usuario, partiendo del punto de que son el puente entre el usuario y el sistema. Es válido mencionar que términos como generación de código con sus características, ventajas y desventajas se hacen notar en todo el proceso de la investigación. Y de igual modo se hace un análisis de los mismos.

1.3 Generador de código

Los generadores de código toman como entrada un modelo, que pudiera ser un diagrama de clases, un modelo de base de datos, e incluso un lenguaje declarativo previamente definido. El código generado depende en gran medida del modelo inicial, además de los algoritmos y patrones que tenga implementado el generador (Chang y Carbonell 2013).

De forma general estos sistemas son usados para construir programas de manera automática y en un tiempo breve, creando código consistente y de alta calidad para contribuir al desarrollo de proyectos y aplicaciones informáticas. Pero además de lo antes expuesto los generadores de código también cuentan con características específicas, tipos y funcionamiento.

1.3.1 Características de los generadores de códigos

Los generadores de código se caracterizan por el lenguaje que generan, el cual puede ser estándar o propietario. Por la portabilidad del código generado, que es la capacidad para poder ejecutarlo en diferentes plataformas físicas y/o lógicas. Por la generación del esqueleto del programa o del programa completo; si únicamente genera el esqueleto será necesario completar el resto mediante programación. Por la posibilidad de modificación del código generado y por la generación del código asociado a las plantillas e informes de la aplicación, mediante el cual se obtendrá la interfaz de usuario de la aplicación (Donal y Zurbano 2013).

1.3.2 Tipos de generadores de código según:

- **Activos**: el código puede ser generado varias veces con solo hacer cambios en las entradas. Estos generadores definen espacios de código seguros donde el programador puede hacer los cambios que desee sin que estos se pierdan en las sucesivas generaciones de código (Marwedel y Goossens 2013).

- **Pasivos**: generan el código una vez y no vuelven a tener interacción con él. Tienen la desventaja de que si se corrige un error en los mecanismos de generación o se cambia el diseño y se vuelve a generar se pierde lo que se codificó manualmente (Marwedel y Goossens 2013).

Según la aproximación que usan para generar el código se clasifican en:

- **Estructural**: generan bloques de código, desde modelos estáticos y relaciones entre objetos. Las primitivas de trabajo en estos modelos son clases, atributos, tipos y asociaciones. Algunas herramientas usan un motor de traducción y plantillas preexistente para especificar correspondencias con un código fuente en particular. La generación de código estructural es incompleta, pero ahorra esfuerzo de codificación manual y proporciona un marco de trabajo inicial consistente con los modelos (Marwedel y Goossens 2013).
- **De Comportamiento**: generan código completo a partir de modelos de máquinas de estados y la especificación de acciones en un lenguaje de alto nivel. Algunos métodos que modelan comportamiento con máquinas de estados, añaden código (como C++ o un lenguaje propietario) para representar las acciones que ocurren durante la transición de estado. Junto con modelos de estructuras de objetos y mecanismos de comunicación, esta técnica permite a las herramientas generar código para el modelo completo de la aplicación. Un beneficio de esta técnica es la capacidad para simular y verificar el comportamiento del sistema basado en modelos antes de que el código sea generado (Marwedel y Goossens 2013).
- **Traductivos**: se basan en que los modelos de aplicación y de arquitectura son independientes uno del otro. Un modelo de aplicación completo, con estructura de objetos, comportamiento y comunicaciones es creado usando el método de Análisis Orientado a Objetos. Un modelo de arquitectura (un conjunto de patrones llamados plantillas o arquetipos) es desarrollado con una herramienta que soporte esta aproximación. Entonces, un motor de traducción genera el código para la aplicación de acuerdo con las reglas de correspondencia en la arquitectura. Las aproximaciones traductivas ofrecen una reutilización significativa debido a que la aplicación y el modelo de arquitectura son independientes (Marwedel y Goossens 2013).

1.3.3 Funcionamiento de los generadores de código

Los generadores implementan las siguientes fases:

- **Carga**: la fase de carga consiste en la lectura desde un repositorio (puede ser un fichero binario, XML[3], una base de datos, o un diagrama UML[4]) del modelo a traducir y crear una representación de éste en memoria. La representación de este modelo en memoria, no tiene que ser completa, ni

[3] Lenguaje de Marcado Extensible (del inglés *eXtensible Markup Language)*
[4] Lenguaje Unificado de Modelado

tampoco seguir la misma estructura del modelo. Al contrario, la carga y las estructuras pueden ser adaptadas para cargar sólo la información necesaria y disponerla del modo que sea más conveniente para la tarea de traducción a realizar (Moreno 2003b).

- **Inferencia**: para la fase de inferencia, se han definido una serie de mecanismos que completan la información del modelado. Las estructuras del modelo en memoria son completadas y extendidas. Es necesario llevar a cabo este proceso antes de proceder a la generación propiamente dicha. El proceso es responsable de realizar pre-cálculos útiles y de disponer adecuadamente la información para la fase posterior (Moreno 2003b).
- **Generación**: en función del código destino a producir, se recorren secuencialmente y de modo anidado los elementos de modelo en sucesivas pasadas. Por ejemplo: para cada clase, para cada servicio y para cada argumento. Como resultado de esta fase se obtiene el código generado (Moreno 2003b).

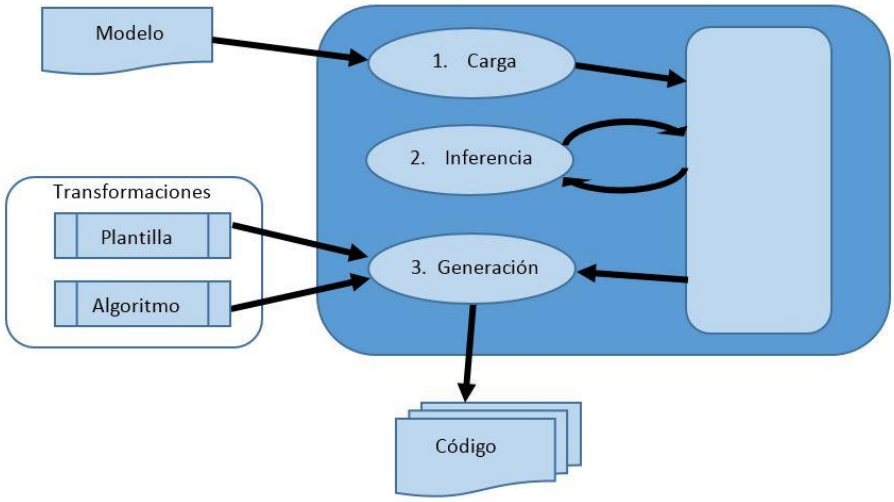

Figura 1 Fases del generador de código.
Fuente: (Moreno 2003b).

1.3.4 Técnicas de generación de código

Se reconocen cuatro técnicas generales utilizadas por los generadores de códigos: (Moreno 2003a):

- **Clonación**: si el código destino a producir contiene ficheros que permanecen constantes independientemente del modelo a traducir, la traducción puede llevarse a cabo por medio de clonación, o copia directa del fichero origen al directorio de generación. Las librerías de funciones genéricas o ficheros binarios representando imágenes o iconos constantes pueden ser tratados de este modo.

- **Concatenación de cadenas**: la concatenación de cadenas es un modo sencillo de ir construyendo el código destino desde un lenguaje de programación clásico. El nombre proviene del proceso de ir concatenando cadenas de texto que contienen el código a producir. Finalmente, la cadena es volcada a un fichero. Se dispone de la potencia del lenguaje de programación para determinar que código debe ser generado, permitiendo cálculos, sustituciones y procesados complejos.
- **Plantillas**: la generación mediante plantillas puede ser empleada cuando el código destino a producir tiene un patrón de repetición bien caracterizado, de modo que los ficheros generados son idénticos, salvo los datos procedentes directamente del modelo a generar. En este caso puede definirse una plantilla genérica a partir de ejemplares del código objetivo. En esta plantilla, las dependencias del modelo son sustituidas por marcadores con nombre único. Aparejado a la plantilla, se puede definir un proceso de generación que, dado un elemento del modelo, la plantilla sea instanciada a código mediante un proceso de sustitución de cadenas: los marcadores son sustituidos por los datos del modelo correspondiente.
- **Análisis de expresiones mediante gramáticas**: existen ocasiones donde las estrategias previas de generación se vuelven insuficientes. En particular, un caso muy claro es el del tratamiento de expresiones que siguen una gramática dada. Aquí las técnicas de compilación clásicas son las más adecuadas para producir el código necesario. La expresión puede ser convertida en una estructura con forma árbol: un analizador léxico, sintáctico y semántico realizan este trabajo. Después, un algoritmo puede recorrer dicho árbol que representa la expresión para analizarla y decidir el tipo de código a producir.

Las principales razones para usar técnicas de generación de código son:

1.3.4.1 Ventajas de la generación de código

La generación de código trae consigo muchas ventajas, pero muchos autores e investigadores del tema en las diferentes bibliografías que publican refieren que entre ellas se destacan cuatro muy relevantes, estas son: productividad, calidad, consistencia y abstracción (Chang y Carbonell 2013).

- **Calidad**: la calidad del código generado está en correspondencia con la calidad de las plantillas y del proceso que se usa para la generación. A medida que son detectados los errores y es mejorado el código de las plantillas, la calidad del código generado aumenta. Se pueden imponer reglas de estilo al código generado para aumentar su homogeneidad y legibilidad. Se puede generar la documentación del código así como comentarios que faciliten su mantenimiento (Chang y Carbonell 2013).
- **Consistencia**: el código generado es extremadamente consistente. El nombre de las variables, métodos y clases es creado de la misma forma a lo largo de todo el código. La relación entre el

modelo y el código generado permite que también haya consistencia entre la documentación y el código, la cual es muy difícil de mantener en un proceso de desarrollo tradicional (Chang y Carbonell 2013).

- **Productividad**: es fácil reconocer los beneficios de la generación de código respecto a la productividad. Se comienza con un diseño de entrada e instantáneamente se obtiene una implementación de salida que responde al diseño. Por otro lado estos beneficios son mucho más apreciados cuando se regenera el código debido a un cambio en el diseño y no se pierde el trabajo que ya está hecho. Además, libera a los programadores del trabajo tedioso y repetitivo, permitiéndoles enfocarse en los aspectos que sí requieren un profundo análisis (Chang y Carbonell 2013).
- **Abstracción**: muchos generadores construyen el código basados en modelos abstractos. Por ejemplo, se puede generar una capa de acceso a datos, a través de un XML que represente las tablas, los atributos y sus relaciones. Entre las ventajas que esto brinda está la portabilidad a distintas plataformas, ya que elevando el nivel de abstracción no se crean especificaciones propias de una plataforma, sino que permite centrarse en el modelado de los datos o del negocio. Incluso si surgiera una nueva tecnología sólo habría que cambiar el generador y volver a generar la aplicación para que la implemente (Chang y Carbonell 2013).

1.3.4.2 Desventajas de la generación de código

- **Mantenimiento**: cuando se usa un generador hay que darle mantenimiento constantemente. Si es desarrollado por terceros se tiene que estar al tanto de las versiones nuevas que salen y de los errores que se le van detectando. En todo caso se le dedican esfuerzos a resolver los errores que pueda traer el generador y agregarle las nuevas funcionalidades que van surgiendo en el mercado. Si es un generador poco usado se corre además el riesgo de que sus desarrolladores dejen de darle soporte y que por tanto en poco tiempo se vuelva obsoleto. Por otra parte el grado de sofisticación de estas herramientas dificulta mucho su mantenimiento (Chang y Carbonell 2013).

- **Dominios reducidos**: la generación de código tradicionalmente se ha aplicado a dominios muy específicos y bien conocidos donde es posible anticiparse a la variabilidad de problemas que pueden encontrarse en ese dominio. Los dominios o áreas de aplicación demasiado grandes o fuera de ámbito no se benefician de la aplicación de técnicas de generación de código (Chang y Carbonell 2013).
- **Resistencia por parte de los desarrolladores**: hay programadores convencionales que ven la generación de código como una amenaza para su trabajo, ante lo cual, toman una postura

defensiva o de rechazo. Otros afirman que el uso de generadores va contra las buenas prácticas de diseño y critican mucho la idea de copiar y pegar sobre plantillas (Chang y Carbonell 2013).

1.3.5 Resultado del análisis del estudio de los generadores de código

Una vez analizada la definición de generador de código y los principales aspectos vinculados a las herramientas de esta naturaleza, se determinó que era factible desarrollar una herramienta para darle cumplimiento a la problemática descrita, decidiendo utilizar la técnica de plantillas para desarrollar un generador de tipo pasivo. Esta técnica permite crear plantillas genéricas con el código que se desea generar, no se necesita tener interacción con el código una vez que se ha generado y en caso de cambiar la tecnología o el lenguaje de salida, poder modificarlas permitiéndole al componente seguir siendo reutilizable.

1.4 Análisis de soluciones existentes

En el presente epígrafe se hará un análisis de las herramientas que en la actualidad son utilizadas para la generación de interfaces, con el objetivo de identificar características que puedan servir de guía en el desarrollo de la solución.

1.4.1 Ext Designer

Ext Designer es un constructor de interfaz gráfica de usuario para las aplicaciones web con Ext JS[5]. Esta aplicación de escritorio facilita la creación de las IGU de manera muy rápida y sencilla, permitiendo dedicar la mayor parte del tiempo a la programación y no tanto al diseño. Ext Designer proporciona un fácil uso de su entorno Drag and Drop[6], con el que en pocos minutos y conocimientos mínimos se pueden realizar el prototipado rápido de componentes de interfaz de la aplicación, la conexión de componentes de interfaz de datos y exportar de forma correcta el código orientado a objetos para cada componente. El propio entorno Drag and Drop da la posibilidad de acomodar los componentes a gusto con el mouse. Además proporciona gran cantidad de información técnica en el sitio oficial del producto para evacuar las dudas e incluso videos y tutoriales aclarativos con buenos ejemplos (Candela 2011). Sus características más sobresalientes son:

- **Cambio entre la vista diseño/código:** no será necesario hacer todo el diseño mediante código con la existencia de la vista previa (Candela 2011).

[5] Es una biblioteca de JavaScript para el desarrollo de aplicaciones web interactivas (Orchard et al. 2009).
[6] Arrastrar y soltar

- **La lista de todos los componentes a la mano:** una paleta de componentes definidos que permite seleccionarlos o arrastrarlos hacia el área de trabajo que se esté creando (Candela 2011).
- **Duplicación de Componentes:** le proporciona la capacidad de duplicar conjuntos de componentes y luego modificar sus valores sin arrastrar y soltar los componentes para volver a crear configuraciones comunes una y otra vez (Candela 2011).
- **Transformación de Componentes:** el diseñador realizará automáticamente las transformaciones necesarias para convertir un componente en otro según se necesite (Candela 2011).
- **Deshacer / Rehacer:** Permite deshacer un error o rehacer el último cambio que se hizo a través de botones en la barra de herramientas (Candela 2011).

Su principal desventaja radica en que esta aplicación no es gratuita y para utilizarlo es necesario pagar una licencia de software, además es una aplicación de escritorio que está diseñada para el trabajo en EXTJS por tanto no puede ser utilizada por Angular JS para crear las interfaces.

1.4.2 QT Creator

Es una herramienta para el diseño y la creación de interfaces gráficas de usuario para los componentes de Qt[7]. Esta herramienta pertenece al entorno de desarrollo integrado Qt Creator, que proporciona apoyo a la creación y funcionamiento de aplicaciones de Qt para entornos de escritorio (Windows, Linux, FreeBSD y Mac OS) y dispositivos móviles (Garrido 2009).

Con Qt Designer es posible componer y personalizar los cuadros de diálogo en un entorno WYSIWYG[8], además se pueden probar estos componentes con diferentes estilos y resoluciones directamente en el editor.

Los reproductores, formas y componentes creados con Qt Designer se integran perfectamente con el código programado, utilizando las señales y el mecanismo de las franjas horarias de Qt. Cuenta además con el Diseñador Qt Quick[9], que es una herramienta para el desarrollo de animaciones utilizando un lenguaje de programación declarativa QML[10]. Todas las propiedades establecidas en Qt Designer se pueden cambiar de forma dinámica dentro del código. Además, características como la promoción widget[11] y plugins[12] personalizados permiten utilizar componentes propios en Qt Designer (Rischpater 2014).

---

[7] Qt es un marco de trabajo de desarrollo de aplicaciones multiplataforma (Garrido 2009).

[8] "lo que ves es lo que obtienes" (del inglés ***W**hat **Y**ou **S**ee **I**s **W**hat **Y**ou **G**et*).

[9] Es la biblioteca estándar para escribir aplicaciones QML.

[10] QML (del inglés, *Qt Meta Language*) es un lenguaje basado en JavaScript creado para diseñar aplicaciones enfocadas a la interfaz de usuario (Gauchat 2012).

[11] **Artilugio**: es una pequeña aplicación o programa, usualmente presentado en archivos o ficheros pequeños que son ejecutados por un motor de *widgets*

[12] **Complemento**: es una aplicación (o programa informático) que se relaciona con otra para agregarle una función nueva y generalmente muy específica.

El inconveniente de esta solución es su tecnología, enfocado a componentes del lenguaje Qt y apoyo en la creación de aplicaciones de escritorio. En cambio la investigación está encaminada a la búsqueda de una solución web para Symfony 3.

1.4.3 Codejay

Es una herramienta diseñada para ayudar a los desarrolladores en la creación de aplicaciones web. Permite el trabajo con varias plataformas e incorpora mecanismos de administración para la aplicación. Soporta varios sistemas gestores de base de datos (SQL Server, MS Access, MYSQL). Soporta varios lenguajes de programación (ASP, PHP). Permite la creación automática de reportes WEB. Inserta código JavaScript para la validación dentro de los campos del formulario. En cuanto a su interacción con el código se clasifica en activo y utiliza una aproximación estructural para generar código (Lang y Team 2012).

Su uso se ve limitado, ya que es propietario y no se tiene acceso a su código fuente.

1.4.4 Generador de código JavaScript

El generador fue creado en la UCI para el uso de los trabajadores de CEIGE de la facultad 3. Es una herramienta sencilla, práctica y adaptable a los estilos de programación de cualquier marco de trabajo que utilice Doctrine 2.0[13] para generar las entidades y PHP como lenguaje de programación; mediante la cual se pueda generar el código JavaScript para construir interfaces gráficas en Ext JS 4 de un CRUD. La herramienta se clasificará en pasiva según su interacción con el código generado mediante la técnica de generación de plantillas.

El inconveniente de este generador es que a pesar de encontrarse desarrollado con entidades de Symfony ya se encuentra una nueva versión estable de este marco de trabajo. Por otra parte las interfaces que con este se generan están creadas en Ext JS 4 y la presente investigación está enfocada en Angular JS.

1.4.5 Comparación de las herramientas

Para la comparación de las herramientas seleccionadas se utilizaron los indicadores: propiedad, interacción con el código, plataforma y características; los cuales se utilizan para verificar si es necesario adquirir licencias para el uso de las mismas, comprobar cuantas de estas están basadas en la misma clasificación seleccionada para el desarrollo del componente (pasivo), analizar cuantas de

[13] Es un mapeador de objetos-relacional (ORM) escrito en PHP que proporciona una capa de persistencia para objetos PHP. Es una capa de abstracción que se sitúa justo encima de un SGBD (sistema de gestión de bases de datos)

estas fueron desarrolladas con el uso de las tecnologías web y observar particularidades propias de cada herramienta respectivamente.

Nombre de la herramienta	**Propiedad**	**Interacción con el código**	**Plataforma**	**Características**
Ext Designer	Privada	Activo	Escritorio	Se pueden realizar el prototipado rápido de componentes de interfaz de la aplicación, la conexión de componentes de interfaz de datos y exportar de forma correcta el código orientado a objetos para cada componente
QT Creator	Libre	Pasivo	Escritorio	Es posible componer y personalizar los cuadros de diálogo en un entorno WYSIWYG, además se pueden probar estos componentes con diferentes estilos y resoluciones directamente en el editor.
Codejay	Privada	Activo	Escritorio	Permite el trabajo con varias plataformas. Soporta varios sistemas
Generador de código JavaScript	Libre	Pasivo	Web	Es una herramienta sencilla, práctica y adaptable a los estilos de programación de cualquier marco de trabajo que utilice Doctrine 2.0 para generar las entidades y utilice PHP como lenguaje de programación.

Tabla 1 Comparación de las herramientas.
Fuente: elaboración propia.

Luego del análisis y la comparación de cada una de las herramientas descritas anteriormente se determina que no es factible utilizar ninguna de ellas para la realización del trabajo, ya que en caso de las que son privadas, no es posible tener acceso a su código fuente. En general ninguna de las herramientas se adapta a las necesidades reales del proyecto para el que se desarrolla la solución. La mayoría de estas están creadas para generar código para Ext JS y no para Angular JS o son aplicaciones de escritorio. Debido a estas deficiencias y a las necesidades de la presente investigación, se decide desarrollar una nueva herramienta que permita la generación de código para disminuir el tiempo de desarrollo de interfaces gráficas en Angular JS para aplicaciones web de gestión.

1.5 Metodología de desarrollo

"Una metodología consiste en múltiples herramientas, modelos y métodos para asistir en el proceso de desarrollo de software. En ella se define con precisión los artefactos, roles y actividades involucrados, junto con prácticas y técnicas recomendadas, las guías de adaptación de la metodología al proyecto y las guías para uso de herramientas de apoyo" (Figueroa, Solís y Cabrera 2008).

Para la selección de una metodología se debe tener en consideración las características del proyecto de Desarrollo de Componente. Atendiendo a que el equipo de trabajo es pequeño, el tiempo de entrega del producto es relativamente corto y la comunicación con el cliente es activa, se decidió utilizar como metodología de desarrollo el Proceso Unificado Ágil con la versión establecida por la Universidad de las Ciencias Informáticas (AUP-UCI).

1.5.1 Metodología AUP-UCI

"Al no existir una metodología de software universal, ya que toda metodología debe ser adaptada a las características de cada proyecto (equipo de desarrollo y recursos) exigiéndose así que el proceso sea configurable. Se decide hacer una variación de la metodología AUP, de forma tal que se adapte al ciclo de vida definido para la actividad productiva de la UCI." (Rodríguez Sánchez 2015).

Al contar con un proyecto en el que luego de haberse evaluado el negocio a informatizar y encontrarse muy bien definido; el cliente se encuentra siempre acompañando al equipo de desarrollo para convenir los detalles de los requisitos y así poder implementarlos, probarlos y validarlos se ha seleccionado el escenario cuatro de la metodología. En dicho escenario se plantea que los proyectos que no modelen negocio solo pueden modelar el sistema con HU[14] y es recomendado en proyectos no muy extensos, dado que una HU no debe poseer demasiada información (Rodríguez Sánchez 2015).

1.6 Herramientas, tecnologías y lenguajes a utilizar

Por las necesidades del Departamento de Desarrollo de componente de poder desarrollar interfaces gráficas en un tiempo menor al que se realiza en la actualidad y que con ellas se disminuya el tiempo de desarrollo de los proyectos que contienen este componente se hace necesario que la implementación sea una solución web.

[14] Historia(s) de Usuario(s)

1.6.1 Lenguaje Unificado de Modelado (UML)

UML[15] es un lenguaje muy conocido y utilizado en el modelado de sistemas de software, permite visualizar, especificar, construir y documentar el sistema a desarrollar. Ofrece un estándar para detallar un plano del sistema (modelo), el cual incluye características conceptuales como procesos de negocio, funciones del sistema, aspectos concretos como expresiones de lenguaje de programación, esquemas de bases de datos, entre otros (Quintero et al. 2012).

Se selecciona UML 2.0 como lenguaje de modelado, pues presenta un conjunto de herramientas que permiten modelar (analizar y diseñar) sistemas orientados a objetos, con la modelación de los artefactos durante las primeras fases del ciclo de vida del software. Los desarrolladores en fases posteriores tendrán mayor dominio y comprensión sobre qué es lo que se debe implementar, permitiendo tanto al cliente como a los desarrolladores tener una representación real del alcance y factibilidad que puede llegar a tener el producto.

1.6.2 Herramienta de modelado Visual Paradigm-UML 8.0.

Herramienta de Ingeniería de Software Asistida por Ordenador (CASE) que da soporte al modelado visual con UML 2.0. Visual Paradigm permite crear tipos diferentes de diagramas en un ambiente totalmente visual. Es sencillo de usar, fácil de instalar y actualizar. Genera código para varios lenguajes. Es compatible con otras ediciones y posibilita un entorno de creación de diagramas para UML 2.x. (Paradigm 2013). Es multiplataforma y su licencia es gratuita y comercial. Por las características antes expuestas se seleccionó esta herramienta en su versión 8.0.

Utiliza UML como lenguaje de modelado, ofreciendo soluciones de software que permiten a las organizaciones desarrollar las aplicaciones con mayor calidad de forma rápida y satisfactoria. Es fácil de usar y presenta un entorno gráfico agradable para el usuario.

1.6.3 Lenguajes de programación

Entre los lenguajes de programación existentes de código abierto para desarrollar aplicaciones orientadas a la web se encuentran dos grupos fundamentales de acuerdo con la arquitectura Cliente/Servidor, la programación del lado del servidor y la programación del lado del cliente. La programación del lado del cliente incluye aquellos lenguajes que son únicamente interpretados por una aplicación cliente como el navegador web, entre estos lenguajes se encuentra HTML. Los lenguajes de programación del lado servidor son los reconocidos, ejecutados e interpretados por el propio servidor

[15] UML- Lenguaje de Modelamiento Unificado del inglés (Unified Modeling Language)

y que se envían al cliente en un formato comprensible para él. Para el desarrollo de la herramienta propuesta se emplean los siguientes lenguajes:

- **PHP 7.0**

PHP Pre-procesador de hipertextos (del inglés *Hypertext Pre-processor*) es un lenguaje de programación del lado del servidor gratuito e independiente de plataforma, rápido, con una gran librería de funciones y mucha documentación.

Un lenguaje del lado del servidor es aquel que se ejecuta en el servidor web, justo antes de que se envíe la página a través de Internet al cliente. Las páginas que se ejecutan en el servidor pueden realizar accesos a bases de datos, conexiones en red y otras tareas para crear la página final que verá el cliente. El cliente solamente recibe una página con el código HTML resultante de la ejecución del PHP. Como la página resultante contiene únicamente código HTML, es compatible con todos los navegadores («PHP» 2001) (Olsson 2016).

- **HTML 5**

HTML es el lenguaje que se utiliza para crear todas las páginas web de Internet. Es “un lenguaje reconocido universalmente y que permite publicar información de forma global”. (Web 2015, p. 3) “Define una estructura básica y un código HTML para la definición de contenido de una página web, como texto, imágenes, entre otros” y se basa en la referenciación por hipertextos o enlaces entre páginas (Gauchat 2012) (Hogan 2011).

- **CSS 3.0**

CSS Hojas de Estilo en Cascada (por sus siglas en inglés Cascading Style Sheets) es un lenguaje para controlar la presentación de los documentos electrónicos definidos con HTML y XHTML. CSS es la mejor forma de separar los contenidos y su presentación y es imprescindible para la creación de páginas web complejas. Mejora la accesibilidad del documento, reduce la complejidad de su mantenimiento y permite visualizar el mismo documento en infinidad de dispositivos diferentes (Gauchat 2012).

Se utiliza para definir el aspecto de todos los contenidos, como: el color, tamaño y tipo de letra de los párrafos de texto, la separación entre titulares y párrafos, la tabulación con la que se muestran los elementos de una lista (Gauchat 2012). Esta forma de descripción de estilos ofrece a los desarrolladores el control total sobre estilo y formato de sus documentos. Funciona a base de reglas para las declaraciones sobre el estilo de uno o más elementos (Web 2015, p. 3) (Gauchat 2012).

- **JavaScript 3.0**

JavaScript es un lenguaje de programación web en el lado del cliente que se utiliza principalmente para crear páginas web dinámicas. Es un lenguaje interpretado, por lo que no es necesario compilar los programas para ejecutarlos, es decir, los programas escritos con Javascript se pueden probar directamente en cualquier navegador sin necesidad de procesos intermedios (Gauchat 2012). Está diseñado para el desarrollo de aplicaciones cliente-servidor a través de Internet. Permite incorporar funciones para el control de la información. Evita cargas en las transferencias de datos entre el cliente y el servidor ya que el navegador del usuario es el responsable de asumir toda la carga de procesamiento (Gauchat 2012).

1.6.4 Marcos de trabajo

- **Symfony 3.0**

Es un marco de trabajo que simplifica el desarrollo de una aplicación mediante la automatización de algunos de los patrones utilizados para resolver las tareas comunes. Además, proporciona estructura al código fuente, forzando al desarrollador a crear código más legible y más fácil de mantener. Por último, facilita la programación de aplicaciones, ya que encapsula operaciones complejas en instrucciones sencillas (Vladimir 2012).

Symfony 3 está diseñado para optimizar, gracias a sus características, el desarrollo de las aplicaciones web. 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 (Vladimir 2012).

- **Angular JS 2.2**

Es un marco de trabajo de JavaScript de código abierto, mantenido por Google, que se utiliza para crear y mantener aplicaciones web de una sola página. Su objetivo es aumentar las aplicaciones basadas en navegador con capacidad de Modelo Vista Controlador (MVC), en un esfuerzo para hacer que el desarrollo y las pruebas sean más fáciles (Brat Tech LLC 2013).

1.6.5 Entorno de Desarrollo Integrado (IDE)

- **PhpStorm 2016.1**

JetBrains PhpStorm es un IDE comercial multiplataforma para PHP construido en la plataforma IntelliJ IDEA de JetBrains. Es perfecto para trabajar con Symfony, Drupal, WordPress, Zend Framework, Laravel, Magento, Joomla, CakePHP, Yii y otros marcos de trabajo.

PhpStorm proporciona un editor para PHP, HTML y JavaScript con análisis de código en la marcha, prevención de errores y refactorizaciones automatizadas para PHP y código JavaScript. Incluye un editor SQL completo con resultados de consulta editables (Chaudhary y Kumar 2014).

1.6.6 Servidor de Aplicaciones

- **Apache 2.4**

El servidor HTTP Apache es un servidor web HTTP de código abierto, para plataformas Unix (BSD y GNU/Linux), Microsoft Windows, Macintosh y otras, que implementa el protocolo HTTP/1.1 y la noción de sitio virtual. Apache es altamente configurable, bases de datos de autenticación y negociado de contenido, pero fue criticado por la falta de una interfaz gráfica que ayude en su configuración.
Se considera Apache en su versión 2.4 como servidor web para el desarrollo del presente trabajo, pues además de las ventajas de ser de código abierto, gratis y de fácil adquisición, Apache es el servidor web más empleado en los centros de desarrollo de la UCI (Kersken 2012).

1.6.7 Sistema de Control de Versiones

- **Git 2.7.4**

Cuenta con una característica que lo hace destacar de casi cualquier otro Sistema de Control de Versiones (SCV) y es su modelo de ramas. Permitirá tener múltiples ramas locales que son independientes entre sí. Git permite crear ramas de prueba, volver a un punto de bifurcación de una versión, aplicar un parche y regresar a la rama de prueba y fusionar ambas. La herramienta implementa un sistema que permite tener una rama que contiene todo el trabajo de producción, otra rama con el trabajo para hacer pruebas (testear) y ramas más pequeñas para otros trabajos secundarios. Este permite el trabajo sin necesidad de estar conectado al repositorio central, esta es una de las características con que cuentan la mayoría de los SCV distribuidos. Esta herramienta cuenta con una gama mucho mayor de comandos a utilizar sin necesidad de una sincronización con la rama principal del servidor remoto (Loeliger y McCullough 2012; Somasundaram 2013; Spinellis 2012).
Para lograr que la herramienta cuente con la calidad que el usuario espera, se hace necesario, además de seleccionar las herramientas y tecnologías adecuadas, aplicarle criterios de calidad estipulados para las diferentes soluciones informáticas.

1.7 Calidad de software

"*Concordancia con los requisitos funcionales y de rendimiento explícitamente establecidos con los estándares de desarrollo explícitamente documentados y con las características implícitas que se espera de todo software desarrollado profesionalmente"* (Pressman 2010).

La calidad del software es el grado en que un sistema, componente o proceso cumple los requerimientos especificados y las necesidades o expectativas del cliente o usuario. Es el conjunto de características de una entidad que le confieren su aptitud para satisfacer las necesidades expresadas y las implícitas (Geraci et al. 1991), sirven como soporte a los procesos de verificación y validación de software para garantizar la calidad de un producto.

1.7.1 Reglas de oro de la interfaz de usuario

Para el diseño de la interfaz, (Pressman 2010) propone las tres reglas de oro que forman la base para los principios del diseño de la interfaz de usuario.

Dar el control al usuario:

- Definir los modos de interacción de manera que no obligue a que el usuario realice acciones innecesarias y no deseadas.
- Tener en consideración una interacción flexible, dado que diferentes usuarios tienen preferencias de interacción diferentes.
- Permitir que la interacción del usuario se pueda interrumpir y deshacer.
- Aligerar la interacción a medida que avanza el nivel de conocimiento y permitir personalizar la interacción.
- Diseñar la interacción directa con los objetos que aparecen en la pantalla.

Reducir la carga de memoria del usuario:

- Reducir la demanda de memoria a corto plazo; la interfaz deberá ser diseñada para reducir los requisitos y recordar acciones y resultados anteriores.
- Establecer valores por defecto útiles.
- Definir las deficiencias que sean intuitivas.
- El formato visual de la interfaz se deberá basar en una metáfora del mundo real.
- Desglosar la información de forma progresiva y de forma jerárquica, para lograr un alto nivel de abstracción.

Construir una interfaz consistente:

- Permitir que el usuario realice una tarea en el contexto adecuado.
- Mantener la consistencia en toda la familia de aplicaciones.

El éxito en el desarrollo de una buena interfaz de usuario según (Pressman 2010) está en lograr que el producto tenga un alto poder de navegación y le brinde al usuario el control total sobre las acciones que desea desarrollar en ella, brindar acceso al contenido y tareas a usuarios de todos los niveles.

1.7.2 Métricas para la validación del diseño

"Una métrica es una medida efectuada sobre los programas, documentación, su desarrollo y mantenimiento que permite medir de forma cuantitativa la calidad de los atributos internos del software" (Pressman 2010).

(Lorenz y Kidd 1994) dividen las métricas basadas en clases en cuatro categorías: tamaño, herencia, valores internos y valores externos. Las métricas orientadas a tamaños para una clase se centran en cálculos de atributos y de operaciones para una clase individual y promedian los valores para el sistema en su totalidad. Las métricas basadas en herencia se centran en la forma en que se reutilizan las operaciones en la jerarquía de clases. Las métricas para valores internos de clase, examinan la cohesión y asuntos relacionados con el código y las métricas orientadas a valores externos examinan el acoplamiento y la reutilización (Pressman 2010).

Del conjunto de métricas planteadas por (Lorenz y Kidd 1994) las que se proponen para validar la propuesta de solución son Tamaño operacional de la clase (TOC) y Relaciones entre clases (RC). De ambas métricas se evalúan parámetros como la responsabilidad, la complejidad de implementación, la reutilización, el acoplamiento, la complejidad del mantenimiento y la cantidad de pruebas con el objetivo de determinar la calidad de la solución diseñada.

1.7.2.1 Métrica Tamaño Operacional de la Clase (TOC).

Está dado por el número de métodos asignados a una clase y evalúa los siguientes atributos de calidad:

- **Responsabilidad**: mide la responsabilidad que tiene una clase determinada. Un aumento del TOC implica un aumento de la responsabilidad asignada a la clase.
- **Complejidad de implementación**: evalúa la complejidad de implementación que tiene una clase del diseño. Un aumento del TOC implica un aumento de la complejidad de implementación de la clase.
- **Reutilización**: indica el grado de reutilización que tiene una clase. Un aumento del TOC implica una disminución del grado de reutilización de la clase.

Criterio de evaluación de la métrica

Atributo	Categoría	Criterio
Responsabilidad	Baja	<=Promedio
	Media	Entre Promedio y 2* Promedio
	Alta	> 2* Promedio
Complejidad de implementación	Baja	<=Promedio

	Media	Entre Promedio y 2* Promedio
	Alta	> 2* Promedio
Reutilización	Baja	> 2* Promedio
	Media	Entre Promedio y 2* Promedio
	Alta	<=Promedio

Tabla 2 Criterio de evaluación de la métrica TOC.
Fuente: (Lorenz y Kidd 1994).

1.7.2.2 Relaciones entre clases (RC)

Está dado por el número de relaciones de uso de una clase con otra y evalúa los siguientes atributos de calidad:

Acoplamiento: representa las conexiones físicas (colaboraciones) entre las clases del diseño. Un aumento del RC implica un aumento del Acoplamiento de la clase.

Cantidad de pruebas: evalúa la cantidad de pruebas necesarias para probar una clase. Un aumento del RC implica un aumento de la Cantidad de pruebas de unidad necesarias para probar la clase.

Complejidad de mantenimiento: mide la complejidad de mantenimiento que tiene una clase determinada. Un aumento del RC implica un aumento de la complejidad del mantenimiento de la clase.

Reutilización: proporciona una medida de la reutilización de una clase. Un aumento del RC implica una disminución en el grado de reutilización de la clase.

Criterio de evaluación de la métrica

Atributo	**Categoría**	**Criterio**
Acoplamiento	Ninguno	0
	Bajo	1
	Medio	2
	Alto	>2
Complejidad del mantenimiento	Baja	<=Promedio
	Media	Entre Promedio y 2* Promedio
	Alta	> 2* Promedio
Cantidad de pruebas	Baja	<=Promedio
	Media	Entre Promedio y 2* Promedio
	Alta	> 2* Promedio
Reutilización	Baja	> 2* Promedio
	Media	Entre Promedio y 2* Promedio

	Alta	<=Promedio

Tabla 3 Criterio de evaluación de la métrica RC.
Fuente: (Lorenz y Kidd 1994).

1.7.3 Matriz de inferencia de indicadores de calidad

La matriz inferencia de indicadores de calidad o matriz de cubrimiento, es una representación estructurada de los atributos de calidad y métricas utilizadas para evaluar la calidad del diseño propuesto. Esta matriz permite conocer si los resultados obtenidos de las relaciones atributo/métrica es positivo o no, llevando estos resultados a una escala numérica donde, si los resultados son positivos se le asigna el valor de 1, si son negativos toma valor 0 y si no existe relación es considerada como nula y es representada con un guion simple (-). Luego se puede obtener un resultado general para cada atributo promediando todas sus relaciones no nulas.

1.7.4 Pruebas unitarias

Es la prueba enfocada a los elementos testeables más pequeño del software. Es aplicable a componentes representados en el modelo de implementación para verificar que los flujos de control y de datos están cubiertos y que ellos funcionen como se espera. Antes de iniciar cualquier otra prueba es preciso probar el flujo de datos de la interfaz del componente. Si los datos no entran correctamente, todas las demás pruebas no tienen sentido. El diseño de casos de prueba de una unidad comienza una vez que se ha desarrollado, revisado y verificado en su sintaxis el código a nivel fuente (Pressman 2010).

1.7.4.1 Pruebas de caja blanca

Las pruebas de caja blanca garantizan que se ejerciten por lo menos una vez todos los caminos independientes del código, así como la ejecución de todos los bucles en sus límites operacionales y todas las decisiones lógicas en las vertientes verdaderas y falsas (Pressman 2010).
La base de este método es el de hacer pruebas en pequeños fragmentos del programa. Cada prueba deberá ser lo más independiente posible de las demás y encargarse de una tarea específica. En la programación procedural u orientada a objetos se puede afirmar que estas unidades son los métodos o las funciones que tenemos definidas (Patiño Camargo, Suárez Villegas y others 2014; Yana y Parodi 2014) (Oré B 2014).
La técnica que se aplicará a la herramienta que se propone es la de camino básico mediante el marco de trabajo PHPUnit. El objetivo de las pruebas unitarias es el aislamiento de partes del código y la demostración de que estas partes no contienen errores. El resultado arrojado por el marco de trabajo se podrá encontrar en el epígrafe 3.3.

1.7.5 Pruebas de caja negra

En las pruebas de caja negra o de funcionamiento, se decide no tener en cuenta el funcionamiento interno de un sistema y solo se analizan sus entradas y salidas. Se aplica tanto como estrategia de testeo, fijándose más en el exterior (usuario) o en la conexión entre diferentes sistemas (interfaz), que como necesidad cuando no es accesible o no es practico estudiar el funcionamiento interno del sistema en análisis. Se requiere menos habilidad técnica, menos tiempo y menos herramientas; por ende, menos costo. Pero solo permite detectar errores y fallos pero no brinda una aproximación a la solución de éstos (Pressman 2010). Mediante los casos de prueba de caja negra se puede comprobar que:

- Las funciones del software son operativas.
- La entrada se acepta de forma adecuada.
- Se produce una salida correcta.
- La integridad de la información externa se mantiene (Pressman 2010) (Espinosa, Lara y Cutiño 2008).

En este trabajo fue seleccionada la técnica de partición de equivalencia, diseñando casos de prueba para cada uno de los requisitos funcionales de la aplicación.

Partición de Equivalencia: divide el campo de entrada en clases de datos que tienden a ejercitar determinadas funciones del software (Pressman 2010).

1.7.6 Pruebas de liberación

"Pruebas diseñadas y ejecutadas por una entidad certificadora de la calidad externa, a todos los entregables de los proyectos antes de ser entregados al cliente para su aceptación" (Rodríguez Sánchez 2015).

1.7.7 Pruebas de aceptación

Estas son pruebas que permiten que el cliente valide todos los requisitos. Las pruebas las realiza el usuario final en lugar del responsable del desarrollo del sistema, una prueba de aceptación puede ir desde un informal caso de prueba hasta la ejecución sistemática de una serie de pruebas bien planificadas. De hecho, la prueba de aceptación puede tener lugar a lo largo de semanas o meses, descubriendo así errores acumulados que pueden ir degradando el sistema (Pressman 2010). La técnica que se aplicará en este nivel es la prueba alfa. En esta un usuario con la presencia del desarrollador como espectador hará las pruebas correspondientes a la herramienta en la que una vez terminada procede a documentar los resultados.

1.7.8 Resultado del análisis de calidad

A partir del análisis realizado se decide aplicar ambas métricas descritas, con el objetivo de validar el diseño de la herramienta que se propone. Con el desarrollo de las pruebas de caja blanca haciendo uso del PHPUnit, se podrá probar todos los posibles caminos sin necesidad de hacerlas mediante cálculos humanos y así evitar errores en la ejecución de la misma. La prueba de liberación será realizada por el grupo de calidad de CEIGE. Las pruebas de caja negra y aceptación se han concebido para que sea el propio usuario el que le realice. Los resultados de cada uno de ellas serán descritos en la fase de validación de la solución en la presente investigación.

1.8 Componente humanístico de la investigación

Con fin de buscar la eficiencia en la rama de la economía, el equipo de diseño y programación de la IU pudiera ser más reducido. A la vez los demás miembros del equipo de desarrollo se pudieran redistribuir dentro del proyecto, con el fin de que cumplan otros roles y así reforzar las tareas más complejas. Para lograr la eficacia dentro del proyecto se hace necesario una buena distribución de los recursos materiales en el trabajo. No se puede tener un proyecto en el que un programador no cuente con una computadora con las mejores prestaciones, sin embargo, otro trabajador con una tarea menor cuente con dicha tecnología.

La ética es un aspecto que no debe faltar dentro de una buena investigación. Los valores de cada investigador al respetar autores reconocidos; hacer debidamente las citas de las publicaciones consultadas referentes al tema a investigar o dar una valoración respetando siempre un buen trabajo investigativo realizado ya por otra persona, son aspectos mínimos a tener en cuenta.

Haciendo un análisis del resultado del aporte social de la presente investigación, se espera que tribute a futuras investigaciones y que marque un paso de avance en el proceso de informatización de la sociedad.

1.9 Conclusiones del capítulo

En el presente capítulo se abordaron los elementos fundamentales relacionados con los generadores de código. De esta forma se definieron las características con las que contará la solución que se propone realizar. A través del análisis de soluciones existentes, se determinó que ninguna podría ser utilizada para dar solución al problema planteado. Es por ello que se hace necesaria la implementación de una nueva aplicación que cumpla con las necesidades identificadas durante la investigación. Se definió como metodología a utilizar la AUP-UCI, identificando así, el cuarto escenario para realizar la especificación de los requisitos funcionales. También fueron definidas las herramientas y tecnologías a utilizar en el proceso de desarrollo del generador, teniendo en cuenta la evolución de las mismas. Se

definió, además, la estrategia de validación a desarrollar en capítulos posteriores, partiendo desde el diseño hasta llegar a las pruebas que se le realizarán a la herramienta.

CAPÍTULO 2. PROPUESTA DE SOLUCIÓN

2.1 Introducción

En este capítulo se presenta la propuesta de solución al problema de la investigación planteado, estableciendo así, las funcionalidades más importantes con las que contará el generador de interfaces en Angular JS. Se enumera los requisitos funcionales y no funcionales que debe tener la aplicación y se aplican métricas de validación de requisitos para corroborar el cumplimiento de los atributos de calidad. Se muestran además los artefactos generados por la metodología AUP-UCI y el cumplimiento de los patrones del diseño. También se establecen los estándares de codificación a tener presentes en la implementación del sistema y se establece su distribución física.

2.2 Requisitos

"Un requisito es simplemente una declaración abstracta de alto nivel de un servicio que debe proporcionar el sistema o una restricción de éste. En el otro extremo, es una definición detallada y formal de una función del sistema." (Sommerville 2016).

Estos juegan un buen papel dentro de la etapa de desarrollo de un proyecto. Una mala definición, especificación o administración de requisitos puede fácilmente llevar al fracaso total lo que podría ser una buena solución informática. Factores tales como requisitos incompletos o mal manejo de los cambios son los errores humanos que se pueden cometer con facilidad.

La gestión de requisitos es un conjunto de actividades que ayudan al equipo del proyecto a identificar, controlar y rastrear los requisitos y los cambios a estos en cualquier momento mientras se desarrolla el proyecto (Pressman 2010).

2.2.1 Técnicas para la captura de requisitos

La captura de requisitos es la actividad mediante la cual, el equipo de desarrollo de un sistema de software, extrae de cualquier fuente de información las necesidades que debe cubrir dicho sistema. La técnica utilizada por la autora fue:

- **Estudio de documentación**: En esta técnica se estudia documentación o estándares que puedan informar sobre las actividades de las tareas a realizar, puesto que en muchas ocasiones algunos procedimientos ya están sujetos a algún tipo de regulación que es preciso tener en cuenta. En el proceso de desarrollo del generador de código se analizó toda la documentación presentada por el cliente.
- **Entrevista:** se aplicó en el Departamento de Desarrollo de Componentes de CEIGE, pues son los usuarios para los que inicialmente se desarrolla la solución. Dentro de este departamento

los clientes seleccionados fueros los ingenieros Julio Cesar Ocaña Bermúdez y Dannel Jiménez Torres. En el proceso de selección de los encuestados se tuvo en cuenta el nivel de integración de los mismos con el proyecto según el puesto que estos ocupan.

Con el estudio de la documentación presentada no fue suficiente para realizar todo el levantamiento de los requisitos, por lo que se aplicó la técnica de las entrevistas para sustentar el proceso. El resultado de ambas arrojó las necesidades reales que se enfrentan dichos clientes y la importancia de desarrollar una aplicación sencilla, amigable y fácil de utilizar.

2.2.2 Definición de los requisitos funcionales

Los requisitos funcionales definen el funcionamiento del sistema y varían según el tipo de solución a implementar. Facilitan un entendimiento de los procesos a desarrollar, que se comprenda con profundidad el problema en cuestión y una mejor identificación de las funcionalidades que serán implementadas (Pressman 2010).

La siguiente tabla muestra el listado de los requisitos funcionales identificados:

No	Nombre
RF1	Generar CRUD
RF1.1	Seleccionar la entidad
RF1.2	Seleccionar los atributos
RF1.3	Generar clase controladora de php
RF1.4	Generar clase formulario de php
RF1.5	Generar vista listar de Angular
RF1.6	Generar vista eliminar de Angular
RF1.7	Generar vista crear de Angular
RF1.8	Generar vista actualizar de Angular
RF1.9	Generar clase controlador de Angular
RF1.10	Generar servicios de Angular
RF1.11	Generar archivo de configuración de Angular
RF2	Visualizar archivos generados

Tabla 4 Requisitos funcionales.
Fuente: elaboración propia.

2.2.3 Requisitos no funcionales

"Los requisitos no funcionales (RnF) son restricciones de los servicios o funciones ofrecidos por el sistema. Incluyen restricciones de tiempo sobre el proceso de desarrollo y estándares. A menudo son aplicados en su totalidad al sistema y normalmente apenas se aplican a características o servicios individuales del sistema" (Pressman 2010).

A continuación se muestran los requisitos no funcionales identificados:

Requisitos no funcionales	
RNF	**Usabilidad**
1	La interfaz de la aplicación a desarrollar debe ser sencilla para reducir el tiempo de capacitación de los usuarios.
2	El generador debe lograr un estado positivo en todos los actores que interactúen con las funcionalidades del sistema.
3	El generador debe mostrar facilidad para interactuar y entender las actividades que realiza el usuario.
Portabilidad	
4	Instalación de un navegador (Firefox, Internet Explorer, etc.) en una computadora del lado del cliente.
5	El sistema debe funcionar sobre las plataformas Windows y Linux.
Funcionabilidad	
6	La herramienta está enfocada en la generación de interfaces de usuarios en aplicaciones web de gestión.
7	La herramienta debe contar con campos obligatorios para garantizar un manejo adecuado de la información introducida por el usuario.
8	La herramienta no permite la entrada de datos incorrectos.
Mantenibilidad	
9	La herramienta debe ser fácil a la hora de analizar el código fuente pues el mismo está comentado y la documentación está redactada en un lenguaje fácil de entender.
Requerimientos de Software y Hardware	
10	Servidor web Apache 2.4. Procesador Pentium III a 1.0 GHz como mínimo 512 Mb de memoria RAM, además de una tarjeta de red.
11	Navegador Web, Internet Explorer 8 o Firefox v32 o superior.

Tabla 5 Requisitos no funcionales.

Fuente: elaboración propia.

El proceso de especificación de requisitos funcionales se realiza a través de las historias de usuarios, artefacto generado en el escenario cuatro de la metodología AUP-UCI.

2.2.4 Historias de usuario

"Las Historias de Usuario son un enfoque de requerimientos ágiles que se focaliza en establecer conversaciones acerca de las necesidades de los clientes. Son descripciones cortas y simples de las funcionalidades del sistema, narradas desde la perspectiva de la persona que desea dicha funcionalidad, usualmente un usuario" (Rodríguez Sánchez 2015).

Las mismas emplean terminologías del cliente sin lenguaje técnico, se realiza una por cada característica principal del sistema. Se emplean para hacer estimaciones de tiempo y para el plan de lanzamientos; reemplazan documentos de requisitos y presiden la creación de las pruebas de aceptación. Las HU difieren de los casos de uso porque son escritas por el cliente, no por los programadores, empleando terminologías del cliente. Las historias de usuario son más "amigables" que los casos de uso formales (Fowler 2003).

Las historias de usuarios deben tener un tiempo de desarrollo de programación estimado entre una y tres semanas, si esta estimación es mayor de tres semanas, esta historia debe ser dividida en dos o más, en el caso de que sea menos de una semana debe ser combinada con otra HU.

2.2.4.1 Descripción de las historias de usuarios

El proceso de especificación de requisitos se realizará mediante las HU, artefacto generado en el cuanto escenario de la metodología AUP-UCI.

- **Estimación de esfuerzo por historias de usuario**

Según la prioridad asignada por el cliente a cada HU y teniendo presente la complejidad y riesgo determinado por el programador, se plasma la estimación de cada una de las HU identificadas, los resultados de la estimación se muestran en la siguiente tabla. La unidad de estimación es el punto. Un punto equivale a 5 días de programación.

No	Historias de Usuario	Puntos de estimación
1	Generar CRUD	12
1.1	Seleccionar la entidad	1
1.2	Seleccionar los atributos	1
1.3	Generar clase controladora de php	1

1.4	Generar clase formulario de php	1
1.5	Generar vista listar de Angular	1
1.6	Generar vista eliminar de Angular	1
1.7	Generar vista crear de Angular	1
1.8	Generar vista actualizar de Angular	1
1.9	Generar clase controlador de Angular	1
1.10	Generar servicios de Angular	1
1.11	Generar archivo de configuración de Angular	2
2	Visualizar archivos generados	2

Tabla 6 Estimación de esfuerzo por historias de usuario.
Fuente: elaboración propia.

Luego de realizar la estimación se presenta las historias de usuarios, la correspondiente al requisito "Generar CRUD" engloba los 12 primeros requisitos pues son necesarios para realizar el proceso.

Número: 1	**Nombre del requisito:** Generar CRUD	
Programador: Claudia Rojas Maxan		**Iteración Asignada:** 1
Prioridad: Alta		**Tiempo Estimado:** 5 días
		Tiempo Real: 1 semanas
Descripción: El usuario selecciona la entidad de Symfony a la que el usuario desea generarle la interfaz. Automáticamente se genera la ruta mediante la cual se va a tener acceso a la interfaz generada. Se debe seleccionar al menos un atributo de ambas listas pues si esta condición no se cumple no será posible activar el botón "Generar". Luego de hacer clic en el botón "generar" se muestra una lista de todos los archivos generados que corresponden con los requisitos que engloba el "Generar CRUD".		

Prototipo de la interfaz:

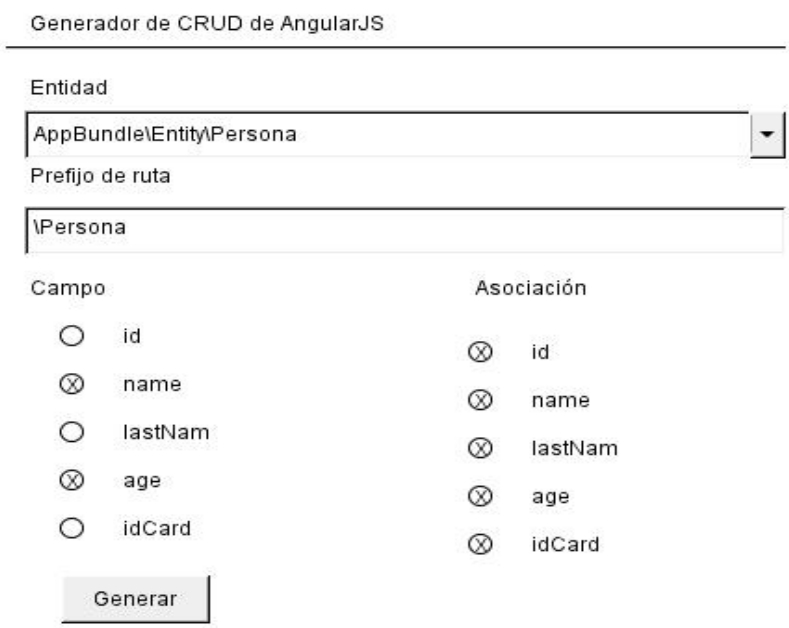

Tabla 7 Historia de usuario.

Fuente: elaboración propia.

A continuación se mostrará la Tarea de la Ingeniería que se corresponde con las HU descrita anteriormente.

2.2.5 Tareas de la ingeniería

Las Tareas de la Ingeniería (TI) se usan para describir las tareas que se realizan sobre el proyecto. Estas pueden ser de: desarrollo, corrección y mejora. Las TI tienen relación con una historia de usuario, aunque de una historia de usuario se puede derivar más de una tarea de la ingeniería. En estas se especifica la fecha de inicio y fin de la tarea, se nombra al programador responsable de cumplirla, también aparece el tiempo estimado que se demorará en realizar las tareas y se realiza una pequeña descripción de lo que se tratará de hacer en la tarea.

Tarea de Ingeniería	
Número Tarea: 1	**Número Historia de Usuario: 1**
Nombre Tarea: "Generar CRUD**".**	

Tipo de Tarea : Desarrollo	**Puntos Estimados: 12**
Fecha Inicio: 10-1-2017	**Fecha Fin: 15-1-2017**
Programador Responsable: Claudia Rojas Maxan	
Descripción: La herramienta permitirá seleccionar la entidad de Symfony a la que el usuario desea generarle la interfaz. Automáticamente se genera la ruta mediante la cual se va a tener acceso a la interfaz generada. Se debe seleccionar al menos un atributo de ambas listas pues si esta condición no se cumple no será posible activar el botón “Generar”. Luego de hacer clic en el botón “generar” se muestra una lista de todos los archivos generados que corresponden con los requisitos que engloba el “Generar CRUD”.	

Tabla 8 Tareas de ingeniería.
Fuente: elaboración propia.

2.2.6 Validación de requisitos

“La validación de requisitos demuestra que la definición de los mismos responde correctamente a las necesidades del cliente. Esta es una actividad fundamental, ya que un levantamiento de requisitos con errores, que no se detecten a tiempo conduce a resultados inesperados, provocan costos excesivos y gran pérdida de tiempo” (Pressman 2010).

2.2.6.1 Métricas para la validación de requisitos

La validación de los requisitos debe estar orientada a certificar que los requisitos definidos cumplen con los atributos de calidad (Davis et al. 1993):

- No ambiguo

Un requisito no es ambiguo, sí y solo sí, tiene una única posible interpretación. Múltiples interpretaciones de un requisito pueden causar desacuerdos entre clientes y desarrolladores. Este atributo se mide a partir del porcentaje de requisitos interpretados de una única manera:

$Q1 = \frac{nui}{nr}$ (1)

Donde:

Q1 = coeficiente de no ambigüedad

nui= número de requisitos con una única interpretación

nr = número total de requisitos identificados

Cuando el valor resultante se aproxima a 0, se interpreta como que los requisitos tienen múltiples interpretaciones; cuando el valor se aproxima a 1, se interpreta como que los requisitos tienen una única interpretación. Debido a que este atributo es muy crítico para el éxito de un proyecto se recomienda que $Q1 = 1$.

- Comprensible

Un requisito es comprensible cuanto todos lo leen (clientes, usuarios, desarrolladores) y comprenden su significado con un mínimo de explicación.

$Q1 = \frac{nur}{nr}$ (2)

Donde:

Q_2 = coeficiente del atributo de calidad comprensible

nur = número de requisitos identificados que todos los revisores entienden

nr = número total de requisitos identificados

El valor resultante oscila en el rango entre 0 (ningún requisito comprendido) y 1 (todos los requisitos comprendidos). Debido a que este atributo es crítico para el éxito de un proyecto se recomienda que $Q2 = 1$.

- Correcto

Un requisito es correcto, sí y solo sí, representa una necesidad de función del sistema a construir. Este atributo puede ser medido de la siguiente manera:

$Q3 = \frac{nc}{nr}$ (3)

Donde:

$Q3$ = coeficiente del atributo de calidad correcto

nc = número de requisitos correctos,

nr = número total de requisitos identificados.

- Posible de tracear

Un requisito es posible de tracear, sí y solo sí, es escrito de manera que facilite la referencia de cada requisito individualmente a los requisitos con los cuales se relaciona, así como a la fuente de origen en el modelo del negocio.

$Q4 = \frac{np}{nr}$ (4)

Donde:

$Q4$ = coeficiente del atributo de calidad posible de tracear

np = número de requisitos identificados posibles de tracear

nr = número de requisitos identificados

Para realizar la validación de los requisitos se tuvo en cuenta no solo el criterio del desarrollador, sino también el criterio de los usuarios. Los resultados obtenidos se muestran a continuación:

Atributos de calidad	Resultado	Interpretación
No Ambiguo	$Q1 = \frac{nui}{nr} = \frac{11}{11}$ =1	Todos los requisitos tienen una única interpretación (no ambiguos)
Comprensible	$Q1 = \frac{nur}{nr} = \frac{11}{11} = 1$	Todos los requisitos son comprendidos (comprensibles)
Correcto	$Q3 = \frac{nc}{nr} = \frac{11}{11} = 1$	Todos los requisitos representan una necesidad de función del sistema a construir (correctos)
Posible de tracear	$Q4 = \frac{np}{nr} = \frac{11}{11} = 1$	Todos los requisitos están escritos de manera que facilite la referencia de cada requisito individualmente a los requisitos con los cuales se relaciona (posibles de tracear)

Tabla 9 Métricas para la validación de requisitos.
Fuente: elaboración propia.

A partir de las métricas de validación de requisitos se puede constatar que luego de realizar dos iteraciones en cada uno de los criterios evaluados, ya en una tercera iteración se logró obtener un 100% de concordancia entre el programador y los usuarios que participaron en el desarrollo de los requisitos, respecto al cumplimiento de los atributos de calidad de los mismos, obteniéndose que en su totalidad son no ambiguos, correctos, comprensibles y posibles de tracear.

Luego de haber finalizado el proceso de especificación de requisitos se plantea la propuesta de solución.

2.3 Propuesta de solución

El usuario interactúa con una aplicación web que le facilitará generar interfaces. Para generar dichas interfaces debe primeramente seleccionar de una lista de entidades la que desea se le genere la interfaz, como respuesta a esta acción la herramienta genera automáticamente el prefijo de la ruta a la que se tendrá acceso. Como último paso se debe seleccionar los atributos que se van a mostrar en la vista listar y los campos del formulario. Luego de realizada esta acción se activa el botón generar que

al hacer clic en él se muestra un listado de los archivos generados y la ruta de acceso a la interfaz que se ha creado. Con la implementación de la nueva herramienta de generación de interfaces gráficas de usuarios para aplicaciones web de gestión, se disminuirá el tiempo de desarrollo de los proyectos, del mismo modo se esperaría un resultado favorable si se aplicara dicha herramienta en los proyectos productivos de CEIGE y lograr hacer más extensible el uso de la misma.

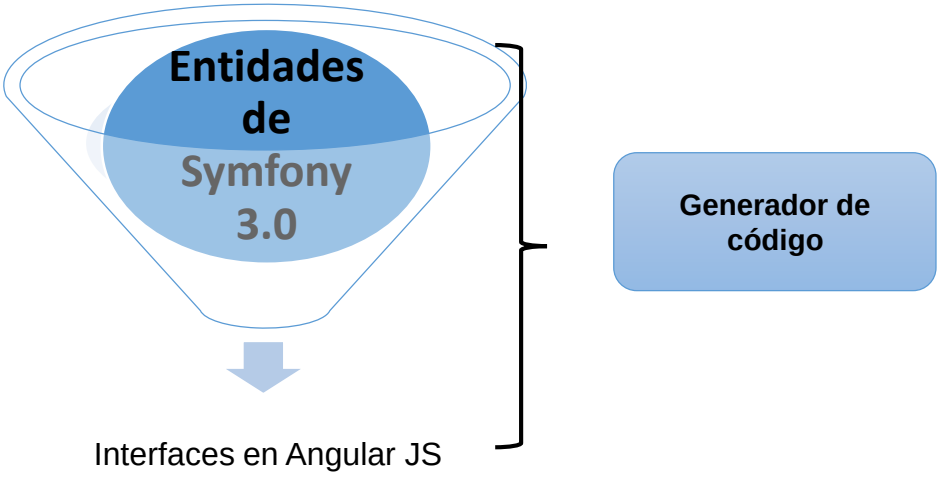

Figura 2 Propuesta de solución.
Fuente: elaboración propia.

2.4 Patrones arquitectónicos

Según (Alexander, Angel y Ishikawa 1980) cada patrón describe un problema que ocurre una y otra vez en un entorno, además describe el núcleo de la solución de ese problema de forma que se pueda utilizar esta solución, sin hacerlo nunca de la misma manera (Pressman 2010).
Los patrones arquitectónicos ofrecen soluciones a problemas de arquitectura en ingeniería de software. Dan una descripción de los elementos y el tipo de relación que tienen, junto a un grupo de restricciones sobre cómo pueden ser usados. Un patrón arquitectónico expresa un esquema de organización estructural esencial para un sistema de software, que consta de subsistemas, sus responsabilidades e interrelaciones (Larman 2014).

2.4.1 Modelo-Vista-Controlador (MVC)

Dentro de los patrones arquitectónicos existentes, se encuentra el patrón de arquitectura de las aplicaciones de software MVC, el cual resulta uno de los más empleados para el desarrollo de aplicaciones web. Este patrón permite dividir la aplicación en tres grandes capas (Jain, Bhansali y Mehta 2015).

- **Vista:** presenta el modelo en un formato adecuado para interactuar (usualmente la interfaz de usuario) por tanto requiere de dicho modelo la información que debe representar como salida. En la solución propuesta este patrón se evidencia con la existencia de una carpeta *view* donde se encuentran las vistas generadas para cada interfaz.

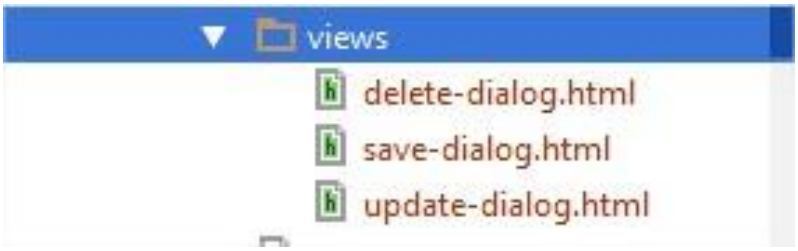

Figura 3 Ejemplo donde se evidencia el patrón vista.
Fuente: elaboración propia.

- **Modelo:** es la representación de la información con la cual el sistema opera, por lo tanto gestiona todos los accesos a dicha información, tanto consultas como actualizaciones, implementando también los privilegios de acceso que se hayan descrito en las especificaciones de la aplicación (lógica de negocio). Envía a la vista aquella parte de la información que en cada momento se le solicita para que sea mostrada. Las peticiones de acceso o manipulación de información llegan al modelo a través del controlador. La presencia del patrón se evidencia dentro de la carpeta *entity* donde se guardan las entidades creadas y son las encargadas de obtener toda la información.

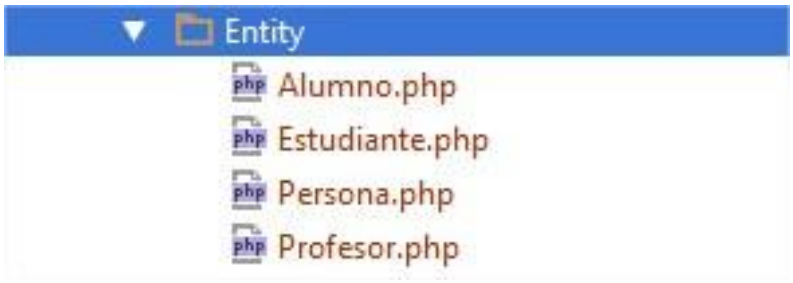

Figura 4 Ejemplo donde se evidencia el patrón modelo.
Fuente: elaboración propia.

- **Controlador:** responde a eventos e invoca peticiones al modelo cuando se hace alguna solicitud sobre la información. También puede enviar comandos a su vista asociada si se solicita un cambio en la forma en que se presenta el modelo por tanto se podría decir que el controlador hace de intermediario entre la vista y el modelo. En las clases controladoras creadas se encuentra presente el patrón controlador encargado de atender las solicitudes del usuario y trabajar como intermediario entre el modelo y la vista.

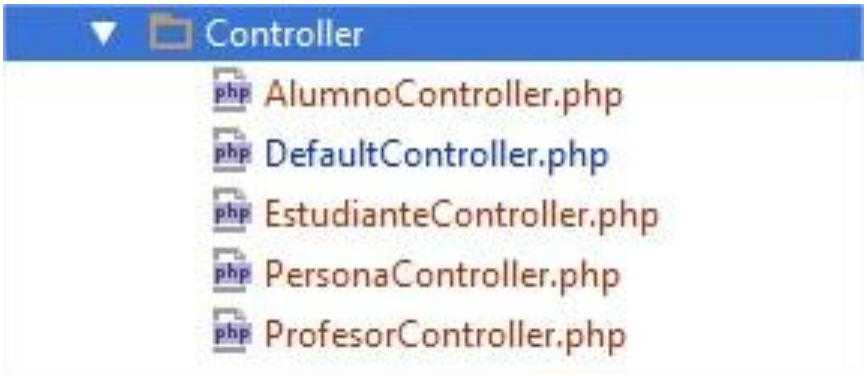

Figura 5 Ejemplo donde se evidencia el patrón controlador.
Fuente: elaboración propia.

2.5 Patrones de diseño

Los patrones de diseño son soluciones a problemas repetidos en la construcción de software y en ocasiones pueden incluir sugerencias para aplicar estas soluciones en diversos entornos (Kaisler 2005).

2.5.1 Patrones GRASP

GRASP[16], se considera que más que patrones propiamente dichos, una serie de "buenas prácticas" de aplicación recomendable en el diseño de software (Larman 2014). Entre los patrones utilizados para el desarrollo del sistema se encuentran los siguientes:

- **Experto:** el patrón fue usado con el objetivo de darle a las clases la responsabilidad necesaria siempre que contaran con la información para cumplirla, logrando así un mejor comportamiento entre las clases, fáciles de comprender y mantener (Larman 2014). Este patrón se evidencia en la clase Crud, ella contiene toda la información y la lógica del negocio que comprende el proceso.

En la siguiente Figura 6 se muestra el ejemplo de lo antes expuesto:

[16] Patrones Generales de Software para Asignación de Responsabilidades (acrónimo de General Responsibility Assignation Software Pattern por sus siglas en inglés).

```
class Crud
{
    /**
     * @var string
     * @Assert\NotBlank()
     */
    private $name;
```

Figura 6 Ejemplo de código de uso del patrón experto.
Fuente: elaboración propia.

- **Controlador:** es un objeto de interfaz no destinada al usuario que se encarga de manejar un evento del sistema (Moreno y Elena 2012). La clase DefaultControler se encarga de llevar el control de todos los eventos relacionados con el negocio. Implementa las funcionalidades que dan respuesta a las peticiones del usuario.
- **Creador:** asigna a la clase B la responsabilidad de crear una instancia de clase A. Su uso se puede ver en la clase controladora DefaultControler, ya que ella es la encargada de la creación de objetos de varias clases como son crud y listViewFields.

En la siguiente Figura 7 se muestra la clase DefaultControler:

```
public function createAction(Request $request)
{
    $entity = new Estudiante();
    $form = $this->createCreateForm($entity);
    $form->handleRequest($request);

    if ($form->isValid()) {
        $em = $this->get('doctrine.orm.entity_manager');
        $em->persist($entity);
        $em->flush();

        return new JsonResponse(array(
            'message' => 'La entidad Estudiante se creó correctamente.'
        ));
    }

    $errors = $this->getAllErrorsMessages($form);
    return new JsonResponse($errors, Response::HTTP_INTERNAL_SERVER_ERROR);
}
```

Figura 7 Ejemplo de código del patrón controlador.
Fuente: elaboración propia.

```
private function createCreateForm(Estudiante $entity)
{
    $form = $this->createForm(EstudianteType::class, $entity, array(
        'action' => $this->generateUrl('estudiante_create'),
        'method' => 'POST',
    ));

    return $form;
}
```

Figura 8 Ejemplo de código del patrón creador.

Fuente: elaboración propia.

- **Alta cohesión:** asegura la necesidad de mantener el sistema con un bajo nivel de complejidad, es decir, una clase tiene una responsabilidad moderada en un área funcional y colabora con otras clases para llevar a cabo las tareas (Moreno y Elena 2012).
- **Bajo acoplamiento:** permite mantener bajas dependencias, bajo impacto al cambio e incrementar la reutilización (Moreno y Elena 2012).

La alta cohesión y el bajo acoplamiento están presentes en las demás clases del negocio, ya que cada una de ellas posee el trabajo de realizar las labores que solo le competen a ella y favorecen a mantener bajas dependencias. Para sustentar lo antes expuesto se aplican las métricas para la validación del diseño TOC y RC que serán descritas en el epígrafe 3.3 de la presente investigación.

2.6 Estándares de codificación

Un estándar de codificación comprende aspectos de la generación de código que los programadores deben implementar de forma prudente. El código fuente debe reflejar un estilo armonioso, como si un único programador hubiese escrito todo el código de una sola vez. Es importante establecer un estándar de codificación al comenzar un proyecto para propiciar el trabajo de forma coordinada. El propósito principal para llevar a cabo revisiones del código a lo largo de todo el desarrollo es localizar defectos en el mismo, las revisiones también pueden afianzar los estándares de codificación de manera uniforme. Usar técnicas de codificación sólidas y aplicar buenas prácticas de programación con vistas a generar un código de alta calidad es de gran importancia para obtener un buen rendimiento y contribuir a la calidad del software (Microsoft 2017).

2.6.1 Nomenclatura de las clases

Los nombres de las clases siempre comienzan con la primera letra en mayúscula y el resto en minúscula, en caso de que sea un nombre compuesto se empleará notación **UpperCamelCase**, la cual define que la primera letra de cada una de las palabras es mayúscula y con leerlo se reconoce el propósito de la misma.

Ejemplo: CrudType

En este caso el nombre de la clase está compuesto por dos palabras iniciadas cada una con letra mayúscula.

2.6.2 Nomenclatura según el tipo de clases

Clases Controladoras: Las clases que se encuentran dentro de la carpeta **controllers** después del nombre de la clase llevan la palabra: Controller.
Ejemplo: DefaultController

2.6.3 Nomenclatura de las funcionalidades y atributos

El nombre a emplear para las funciones y los atributos se escriben con la inicial del identificador en minúscula, en caso de que sea un nombre compuesto se empleará notación **CamelCasing**.
Ejemplo de función: generateControllerClass()
El nombre de este método está compuesto por unas tres palabras, debido a esto es que se escribe la primera en minúscula, la segunda y la tercera se inician con letra mayúscula.
Ejemplo de atributo: $entityNamespace
El nombre del atributo está compuesto por dos palabras, la primera en minúscula y la segunda iniciando con letra mayúscula.

2.6.4 Normas de comentarios

Se debe comentar todo lo que se haga dentro del desarrollo, para lograr establecer un código legible y reutilizable y así se pueda aumentar su mantenibilidad a lo largo del tiempo.
Comentarios de clases: antes de declarar una clase se brinda una descripción de esta, donde se explica el propósito de la misma y se escribe en la Figura 9.

```php
/**
 * Este métodos devuelve un array
 */
public function generateIndexTemplate()
{
    $dir = $this->bundle->getPath();

    $target = sprintf(
        '%s/Resources/views/%s/index.html.twig',
        $dir,
        $this->entityName
    );

    $options = array(
        'namespace' => $this->bundle->getNamespace(),
        'entity_name' => $this->entityName,
        'route_name_prefix' => $this->routeNamePrefix,
        'fields' => $this->getFields($this->crud->getListViewFields()),
        'target_dir' => StringUtil::bundleAssetsDir($this->bundle->getName())
    );

    $this->renderFile('AngularGeneratorBundle:Skeleton:crud/views/index.html.twig', $target, $options);

    return array(
        'name' => 'Vista principal',
        'file' => $target
    );
}
```

Figura 9 Nomenclatura de los comentarios en las clases del generador.
Fuente: elaboración propia.

2.6.5 Estilo del código

En la implementación, al escribir las sentencias en php, se utilizarán los tabs del lenguaje como se muestra en la Figura 10

```php
<?php
//codigo
?>
```

Figura 10 Estilo del código.
Fuente: elaboración propia.

Sangría o indexado: la política de sangría a utilizar en la implementación es por tab. Como se muestra en la Figura 11.

```
public function generateCrudAction(Request $request)
{
    $crud = new Crud();
    $form = $this->createForm(CrudType::class, $crud);
    $form->handleRequest($request);

    if ($form->isValid()) {
        $result = $this->get('angular_generator.crud_generator')->generate($crud);
        return new JsonResponse($result);
    }
    return new JsonResponse(FormUtil::getAllErrorsMessages($form));
}
```

Figura 11 Sangría o indexado.
Fuente: elaboración propia.

2.7 Conclusiones del capítulo

La aplicación de las técnicas Análisis documental y Encuesta permitió la obtención y descripción de requisitos, logrando así, un mejor entendimiento de los mismos. La validación de los requisitos se realizó mediante las métricas: no ambigüedad, comprensible, correcto y traceable. Dicho proceso permitió comprobar la calidad de estos requerimientos. Como parte del modelo de diseño se identificaron los patrones a utilizar en el desarrollo de la solución. Con la presencia del MVC se garantiza una correcta comunicación entre el usuario con las interfaces, las clases controladoras y las entidades creadas. Con la implementación de los patrones GRASP se aseguran las buenas prácticas recomendadas para en el diseño de la propuesta de solución. Los estándares de codificación aplicados en el desarrollo de la aplicación permitieron obtener un código fuente estándar y fácil de entender.

CAPÍTULO 3. VALIDACIÓN DE LA SOLUCIÓN

3.1 Introducción

Para determinar si el resultado de un producto es el deseado, se hace necesario realizarle una serie de pruebas que determinarán su calidad. El siguiente capítulo muestra las técnicas empleadas de validación y pruebas al sistema, para garantizar que las tareas realizadas en el diseño y los requisitos cumplen con las normativas planteadas por los clientes.

3.2 Métricas de evaluación

A continuación se detallan los resultados obtenidos al aplicar las métricas para la validación del diseño, mencionadas en el marco teórico referencial de la presente investigación.

3.2.1 Instrumento de evaluación de la métrica TOC

Clase	Cantidad de Procedimientos	Responsabilidad	Complejidad	Reutilización
DefaultControler	2	Baja	Baja	Alta
AngularGeneratorExtension	1	Baja	Baja	Alta
Configuration	1	Baja	Baja	Alta
CrudType	3	Baja	Baja	Alta
AngularCrudGenerator	13	Alta	Alta	Baja
Crud	8	Alta	Alta	Baja
FormUtil	2	Baja	Baja	Alta
MetadataUtil	1	Baja	Baja	Alta
StringUtil	3	Baja	Baja	Alta

Tabla 10 Instrumento de evaluación de la métrica TOC.
Fuente: (Lorenz y Kidd 1994).

- **Procedimiento para la aplicación de la técnica**

- Se determinó la cantidad de procedimientos con que cuenta cada clase del sistema.
- Se calcula el promedio de los procedimientos de la siguiente forma:

$$promedio = \frac{\sum_{i=1}^{n} CantProcedi}{n}$$

CantProced$_i$=cantidad de procedimientos de la clase i

N= total de clases

Promedio = 3.778

3.2.2 Resultados de la aplicación de la métrica TOC

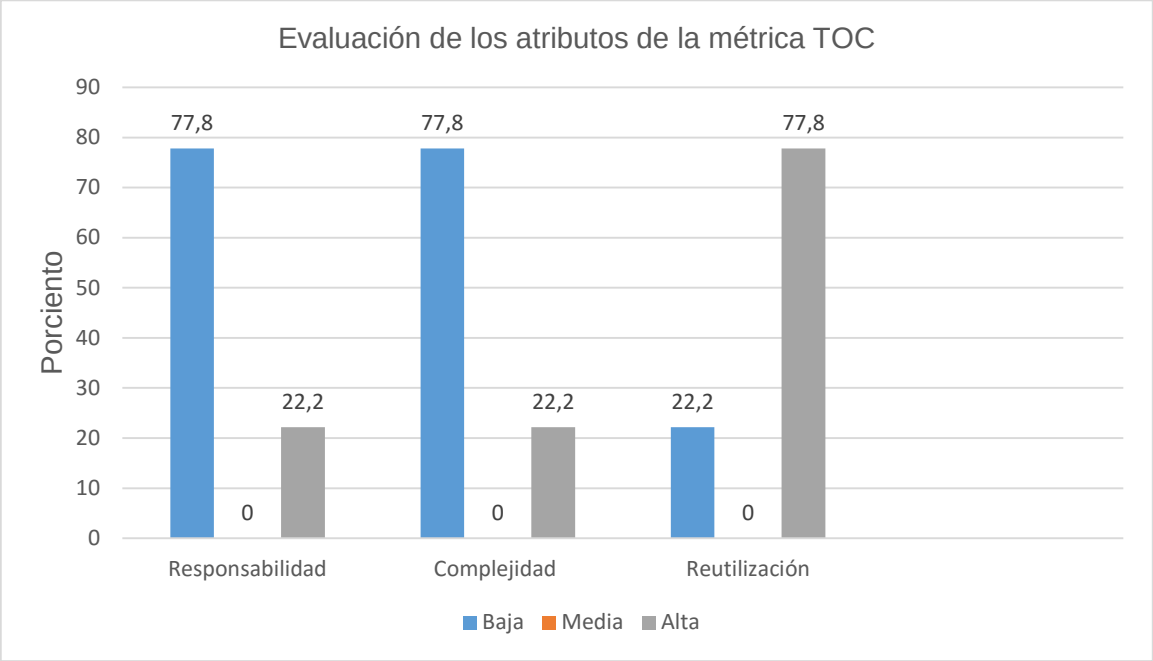

Figura 12 Resultado de la aplicación de la métrica TOC.
Fuente: elaboración propia.

Haciendo un análisis de los resultados obtenidos en la evaluación del instrumento de medición de la métrica TOC, se puede concluir, que el diseño realizado para los requisitos identificados en el desarrollo del generador de código tiene buena calidad. Por lo que se puede afirmar que se obtuvo una solución eficiente con altos niveles de reutilización (77.8%) lo que garantiza que las operaciones realizadas puedan ser utilizadas en otras aplicaciones. También se evidencia que el diseño realizado fue correcto, ya que se obtuvieron altos niveles de responsabilidad (77.8%) y complejidad de implementación (77.8%), pues se les asignaron correctamente las responsabilidades a las clases involucradas en la solución.

3.2.3 Instrumento de evaluación de la métrica RC

Clase	Cantidad de relaciones de uso	Acoplamiento	Complejidad del mantenimiento	Cantidad de pruebas	Reutilización
DefaultControler	0	Ninguno	Baja	Baja	Alta
AngularGeneratorExtension	0	Ninguno	Baja	Baja	Alta
Configuration	1	Baja	Baja	Baja	Alta

CrudType	1	Baja	Baja	Baja	Alta
AngularCrudGenerator	1	Baja	Baja	Baja	Alta
Crud	3	Alta	Alta	Alta	Baja
FormUtil	1	Baja	Baja	Baja	Alta
MetadataUtil	2	Media	Media	Media	Media
StringUtil	2	Media	Media	Media	Media

Tabla 11 Instrumento de evaluación de la métrica RC.
Fuente: (Lorenz y Kidd 1994).

- **Procedimiento para la aplicación de la métrica**

1. Se determinó la cantidad de relaciones de uso con que cuenta cada clase del sistema.
2. Se calcula el promedio de relaciones de uso:

$$promedio = \frac{\sum_{i=1}^{n} CantRelUsoi}{n}$$

CantRelUso i = cantidad de relaciones de uso de la clase i

n = total de clases

Promedio = 1.222

3.2.4 Resultado de evaluación de la métrica RC

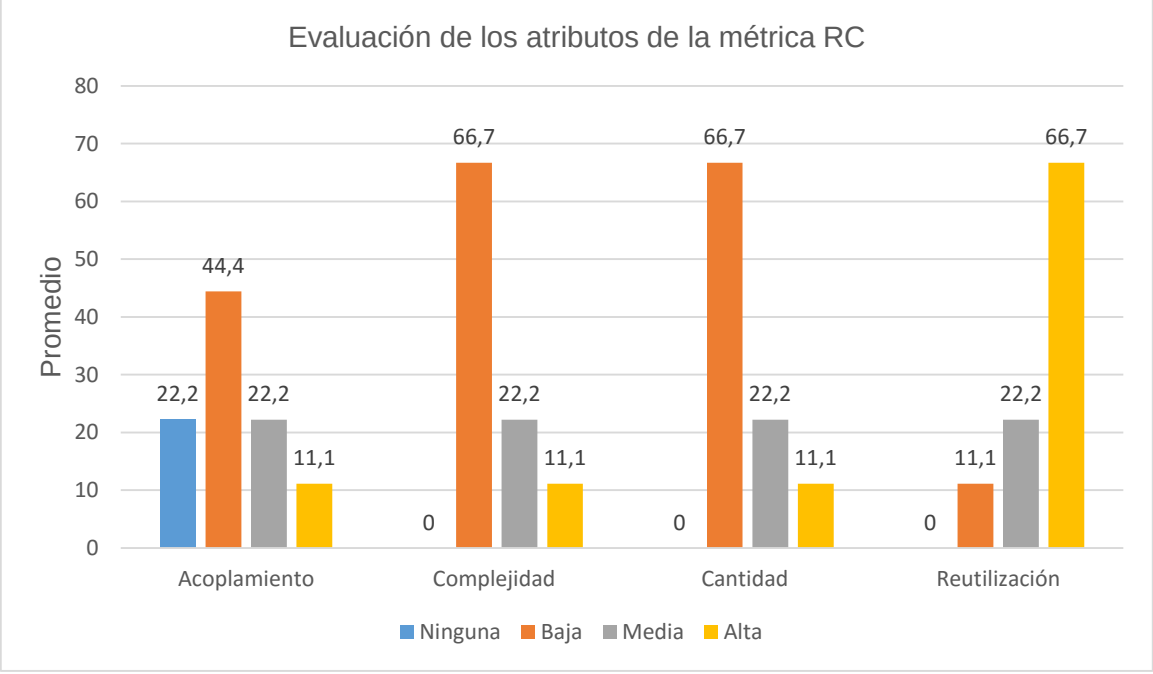

Figura 13 Resultado de evaluación de la métrica RC.
Fuente: elaboración propia.

Haciendo un análisis de los resultados obtenidos en la evaluación del instrumento de medición de la métrica RC, se puede concluir, que el diseño realizado para los requisitos identificados en el desarrollo del generador de código tiene una buena calidad. Al evaluar el acoplamiento se pudo apreciar que no existe, pues el valor que se obtuvo en este parámetro solo es de un 22.2% lo cual demuestra que una clase solo depende de las clases necesarias.

También arrojó resultados satisfactorios en el atributo complejidad de mantenimiento con un 66.7% de las clases con índices bajos, lo que facilita las tareas de corrección, modificación y mantenimiento de las clases. El número o el grado de esfuerzo para realizar las pruebas de calidad del producto diseñado es bajo dado que la métrica aplicada arroja un 66.7% de baja cantidad de pruebas fomentando así un alto índice de reutilización con un 66.7%.

3.2.5 Resultados de la evaluación de la relación atributo/métrica

Atributos\Métricas	TOC	RC	Promedio
Responsabilidad	1	(-)	1
Complejidad de implementación	1	(-)	1
Reutilización	1	1	1
Acoplamiento	(-)	1	1
Complejidad de Mantenimiento	(-)	1	1
Cantidad de Pruebas	(-)	1	1

Tabla 12 Resultado de la evaluación de la relación atributo/métrica.
Fuente: elaboración propia.

Una vez obtenidos los resultados de la evaluación del instrumento de medición de la métrica TOC y RC, se puede concluir que el diseño propuesto tiene una calidad aceptable teniendo en cuenta que en la matriz de cubrimiento todos los atributos de calidad dieron 1 como resultado. A continuación se muestran los resultados obtenidos de la aplicación de las pruebas aplicadas a la herramienta.

3.3 Resultados de las pruebas de caja blanca

Para las pruebas de caja blanca se aplicó la técnica de caso de prueba, la misma se llevó a cabo de forma automatizada mediante el marco de trabajo PHPUnit (Bergmann 2006; Zandstra 2016). Este marco de trabajo permite crear y ejecutar juegos de pruebas unitarias de manera sencilla y utiliza afirmaciones para verificar que el comportamiento de una *unidad* de código es el esperado. Se realizaron un total de 10 pruebas a las entidades y a los controladores del sistema, todos los errores fueron corregidos en la medida en que fueron identificados. Estas pruebas arrojaron como resultado que al final de las 10 iteraciones no se detectaron más no conformidades.

	Code Coverage								
	Lines			Functions and Methods			Classes and Traits		
Total		100.00%	251 / 251		100.00%	41 / 41		100.00%	7 / 7
Controller		100.00%	9 / 9		100.00%	2 / 2		100.00%	1 / 1
Form		100.00%	13 / 13		100.00%	3 / 3		100.00%	1 / 1
Generator		100.00%	187 / 187		100.00%	16 / 16		100.00%	1 / 1
Model		100.00%	15 / 15		100.00%	10 / 10		100.00%	1 / 1
Util		100.00%	27 / 27		100.00%	10 / 10		100.00%	3 / 3
AngularGeneratorBundle.php		n/a	0 / 0		n/a	0 / 0		n/a	0 / 0

Legend

Low: 0% to 50% Medium: 50% to 90% High: 90% to 100%

Generated by php-code-coverage 5.2.1 using PHP 7.0.1 with Xdebug 2.4.0 and PHPUnit 6.1.3 at Wed May 24 21:24:57 CEST 2017.

Figura 14 Resultado de las pruebas de caja blanca.
Fuente: elaboración propia.

La realización de estas pruebas permitió llevar un control de la implementación del sistema, arrojando como resultado que las funcionalidades del código responden a los requerimientos para los que fueron creados.

3.4 Resultados de las pruebas de caja negra

No	Nombre de campo	Clasificación	Descripción
1	Entidad	Campo de selección	Seleccionar la entidad a generar.
	Prefijo de la ruta	Campo de texto	Se genera automáticamente la ruta de la entidad que fue seleccionada.
2	Campos de la vista listar	Campo de selección	Selecciona al menos uno de los atributos de la vista listar.
3	Campos de los formularios	Campo de selección	Selecciona todos los atributos del formulario.

Tabla 13 Descripción de las variables utilizadas en el caso de pruebas: Generar CRUD.
Fuente: elaboración propia.

Escenario	Descripción	Entidad	Prefijo para la ruta	Campos de la vista listar	Campos de los formularios	Respuesta del sistema
EC 1.1 Generar CRUD	Permite seleccionar la entidad a la que se le quiere generar la interfaz.	V	I	NP	NP	Muestra todas las entidades creadas.
		estudiante	/estudiante			
		NP	NP	NP	NP	

Flujo central
1. Seleccionar del menú la entidad a la que se le desea generar la interfaz
2. El sistema completa la ruta a la que se debe acceder.
3. Se seleccionan al menos uno de los campos que se desean listar en la interfaz.
4. Se seleccionan todos los atributos del formulario.

Tabla 14 Resultado de la prueba de caja negra.
Fuente: elaboración propia.

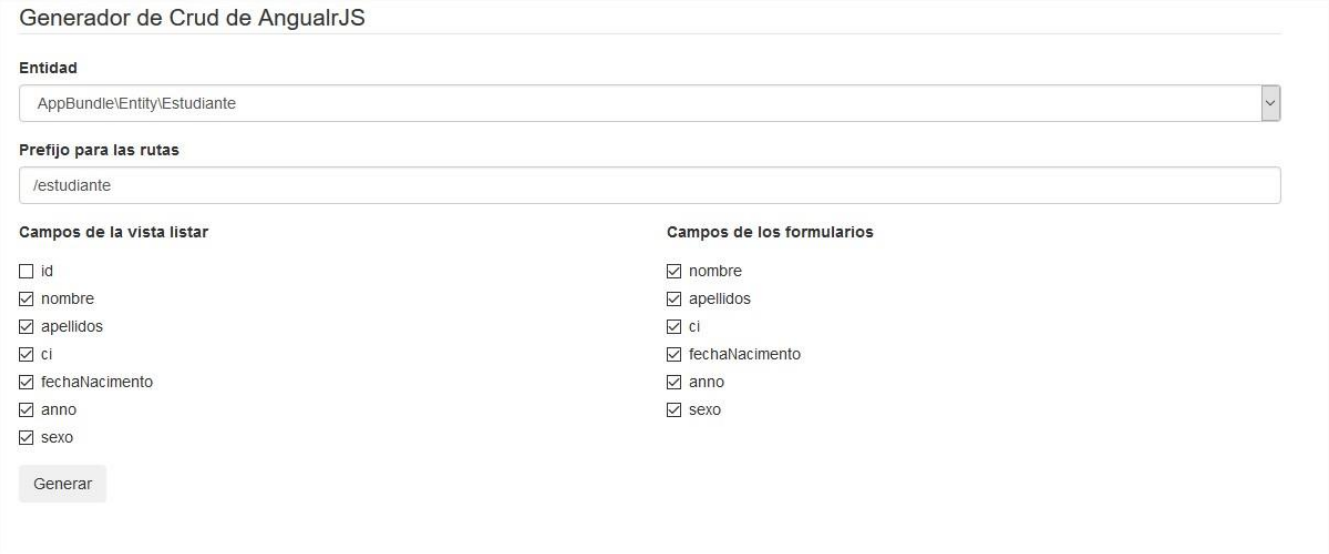

Figura 15 Escenario de prueba: Generar CRUD.
Fuente: elaboración propia.

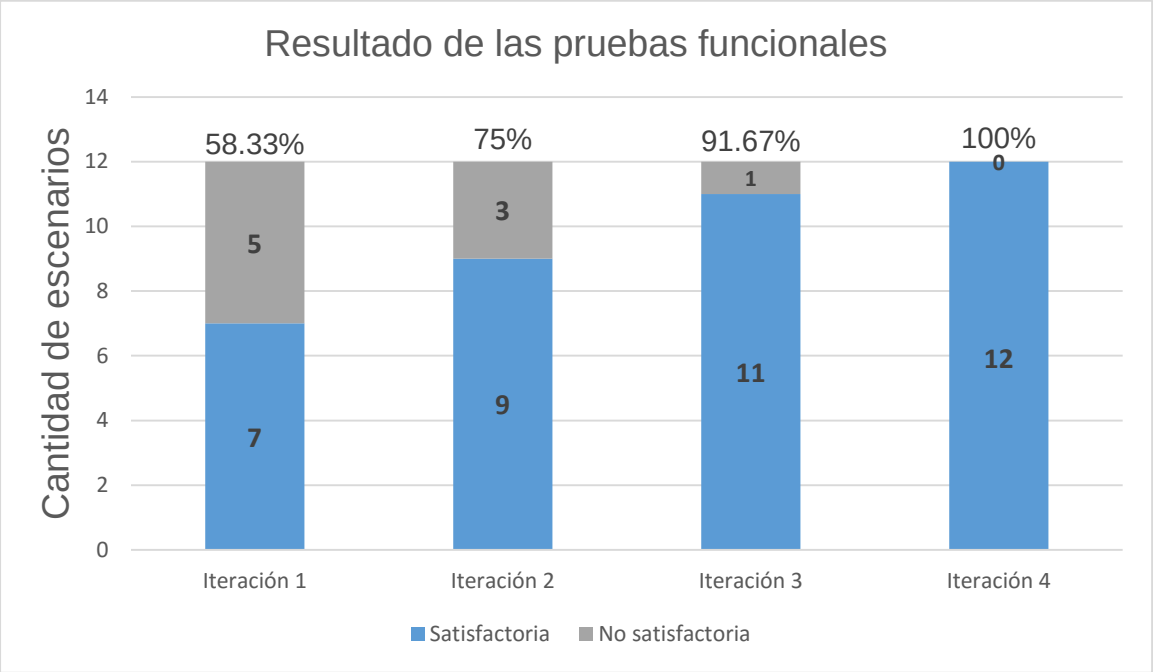

Figura 16 Resultado de las pruebas funcionales.
Fuente: elaboración propia.

La Figura 16 muestra los resultados de las pruebas funcionales realizadas al componente. En sentido general, se refleja un incremento en la efectividad, disminuyendo en cada iteración la cantidad de no conformidades.

En la primera iteración se diseñaron y probaron tres escenarios, correspondientes a tres casos de prueba, de las cuales siete resultaron satisfactorias para un 58.33% de efectividad. En la segunda iteración se realizaron correcciones a las 5 pruebas no satisfactorias que quedaron pendientes de la iteración anterior. Se ejecutaron un total de 12 pruebas, teniendo como resultado 9 pruebas satisfactorias y 3 no satisfactorias, para un 75% de efectividad. Todas las no conformidades pendientes de la primera iteración fueron resueltas.

En la tercera iteración se corrigieron los problemas de la iteración anterior y se detectó solo una prueba no satisfactoria (91.67% de efectividad). Por último, se repitieron todas las pruebas resultando la herramienta 100% funcional.

3.5 Resultados de las pruebas de aceptación

Para realizar estas pruebas se utilizó el mismo principio de las pruebas funcionales, permitiendo que sea el propio cliente el que realice las mismas. Para esto, se realizaron 2 iteraciones, dichos resultados quedaron reflejados en la Figura 17.

En la primera Iteración se realizaron pruebas al requisito funcional Generar CRUD, donde se identificaron 9 no conformidades. Para la segunda iteración fueron probadas nuevamente las funcionalidades de la primera iteración, sumándole también pruebas al requisito Visualizar archivos generados, en la que fueron identificadas 5 no conformidades. En la última iteración se probó el sistema en su totalidad, demostrando que la herramienta creada se encuentra apta para ser usada en el entorno real para el que fue creada.

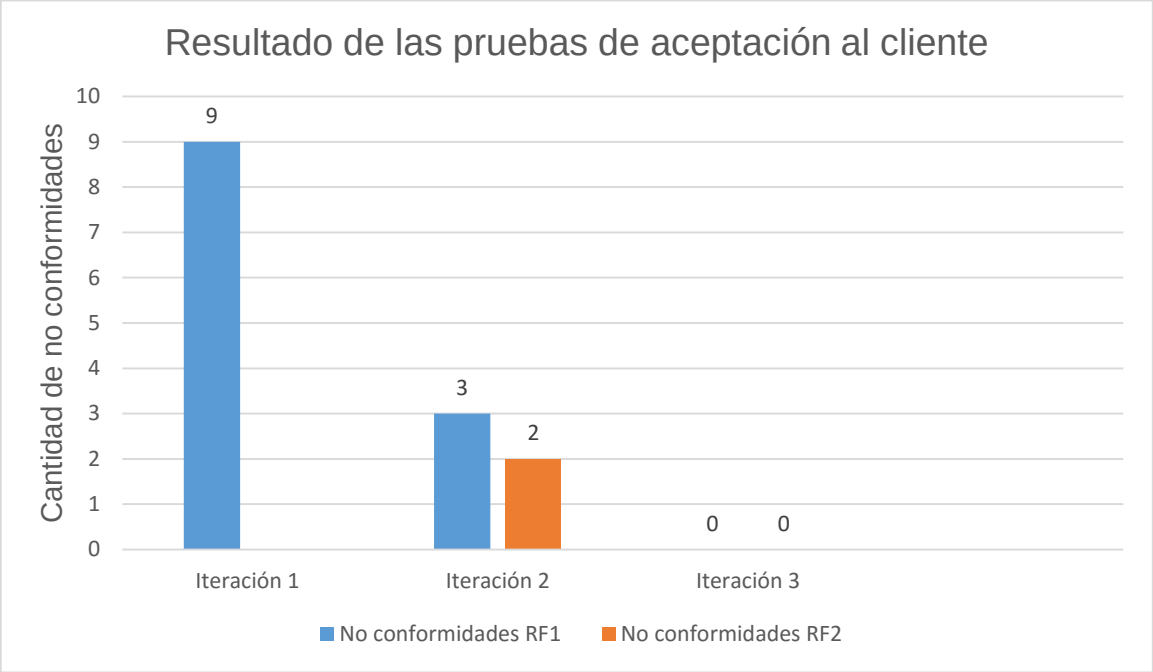

Figura 17 Resultado de las pruebas de aceptación al cliente.
Fuente: elaboración propia.

3.6 Técnica Iadov

La técnica Iadov permite medir la satisfacción del cliente con un producto, se compone de cinco preguntas claves: tres cerradas y dos abiertas, las cuales se reformulan en la investigación para valorar el grado de satisfacción de los clientes sobre un tema en específico. Una vez establecidas las preguntas se conforma el cuadro lógico de Iadov y el número resultante de la interrelación de las tres preguntas, indica la posición de los sujetos en la escala de satisfacción. La escala de satisfacción está dada por los criterios (Cuscó y Wells 2009; López Rodríguez y Maura 2004).

1. Máxima satisfacción.
2. Más satisfecho que insatisfecho.
3. No definida.
4. Más insatisfecho que satisfecho.
5. Máxima insatisfacción.

6. Contradictoria.

Para obtener el índice de satisfacción grupal (ISG) se trabaja con los diferentes niveles de satisfacción que se expresan en la escala numérica que oscila entre +1 y - 1 de la siguiente forma:

Índice de satisfacción	**Escala**
Máxima satisfacción	+1
Más satisfecho que insatisfecho	0,5
No definido y contradictorio	0
Más insatisfecho que satisfecho	-0,5
Máxima insatisfacción	-1

Tabla 15 Índice de satisfacción. Técnica de Iadov.
Fuente: (Cuscó y Wells 2009).

La satisfacción grupal (ISG) se calcula por la siguiente fórmula:

$$ISG = \frac{A(+1) + B(+0.5) + C(0) + D(-0.5) + E(-1)}{n}$$

Donde:

A representa el número de sujetos con índice individual 1

B representa el número de sujetos con índice individual 2

C representa el número de sujetos con índice individual 3 o 6

D representa el número de sujetos con índice individual 4

E representa el número de sujetos con índice individual 5

N representa el número total de sujetos del grupo

Para valorar el grado de satisfacción del cliente con la solución desarrollada respecto al control y disponibilidad de la información, se aplicó la técnica Iadov que permite el estudio del grado de satisfacción del personal involucrado en el proceso de generar código.

¿Considera usted oportuno continuar desarrollando las interfaces gráficas de usuarios cada vez que se va a comenzar el proceso de desarrollo de un proyecto, a pesar de tener un generador que le facilita el trabajo?	¿Le gustaría hacer uso del generador de código propuesto, para desarrollar interfaces gráficas de usuarios?								
	Si			**No se**			**No**		
	¿El generador de código contribuye a mejorar el tiempo de desarrollando las interfaces gráficas de usuarios?								
	Si	**No se**	**No**	**Si**	**No se**	**No**	**Si**	**No se**	**No**
Me gusta mucho	1	2	6	2	2	6	6	6	6

No me gusta mucho	2	2	3	2	3	3	6	3	3
Me da lo mismo	3	3	3	3	3	3	3	3	3
Me disgusta más de lo que me gusta	6	3	6	3	4	4	3	4	4
No me gusta nada	6	6	6	6	4	4	6	4	4
No sé qué decir	2	3	6	3	3	3	6	3	4

Tabla 16 Cuadro lógico de Iadov.

Fuente: elaboración propia.

El índice de satisfacción grupal se encuentra entre + 1 y – 1. Los valores que se encuentran comprendidos entre – 1 y – 0,5 indican insatisfacción; los comprendidos entre – 0,49 y + 0,49 evidencian contradicción y los que caen entre 0,5 y 1 indican que existe satisfacción.

Para medir la satisfacción se le aplicó la encuesta (ver anexo 2) en la que fungieron como clientes 30 de los trabajadores de CEIGE en la Facultad 3.

$$ISG = \frac{22 * 1 + 8 * 0.5}{30}$$

De manera que el ISG = 0,86

Los resultados de la satisfacción individual según las categorías empleadas fueron los siguientes:

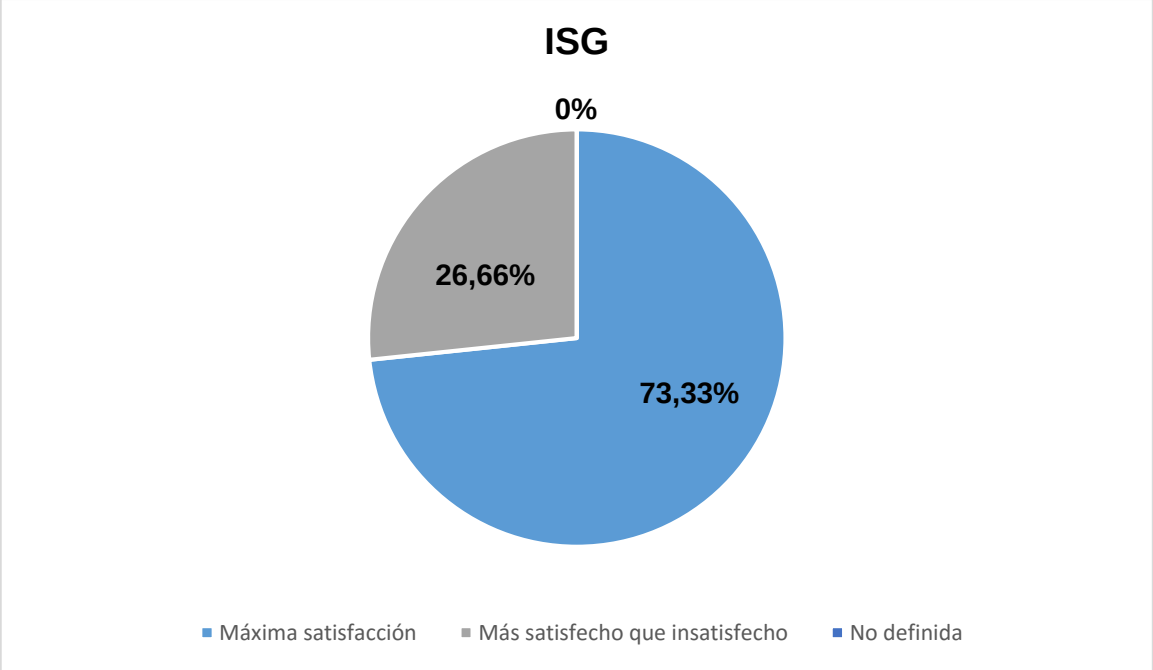

Tabla 17 Resultado de la técnica de Iadov.

Fuente: elaboración propia.

Al procesar las respuestas a las encuestas en el cuadro lógico de Iadov, se obtiene un grado de satisfacción grupal de 0.86, lo cual se traduce en una clara satisfacción con el uso del generador de interfaces.

El cuestionario aplicado contó además con tres preguntas complementarias para conocer la necesidad y el tiempo de desarrollo de las interfaces haciendo uso de la herramienta por parte del usuario. Ello posibilitó conocer que el tiempo que demoraba el 60% de los desarrolladores encuestados en generar un CRUD utilizando la herramienta se encontraba en el rango de 1 a 3 horas.

3.7 Conclusiones del capítulo

Al aplicar las métricas Tamaño Operacional de Clases y Relaciones entre Clases, se demostró que existe un bajo acoplamiento y alta cohesión entre las clases de la aplicación. Las pruebas de caja negra aplicadas demostraron que los requisitos funcionales definidos fueron implementados correctamente. Las pruebas de caja blanca permitieron probar todos los posibles caminos de la solución, obteniendo 11 respuestas satisfactorias. Las pruebas de aceptación y las de liberación permitieron validar el desarrollo de la herramienta y las funcionalidades de la misma, garantizando así una herramienta completamente funcional y que se ajusta a las necesidades del cliente. También se aplicó la técnica Iadov, que permitió medir la satisfacción del cliente respecto al generador de interfaces y comprobar la disminución del tiempo de desarrollo luego de realizada la herramienta.

CONCLUSIONES GENERALES

Al concluir la investigación para el desarrollo del Generador de interfaces en Angular JS para Symfony 3, se pudo arribar a las siguientes conclusiones:

- Se analizaron los fundamentos teóricos y las principales aplicaciones vinculadas al campo de acción, tanto a nivel nacional como internacional, demostrando la necesidad del nuevo sistema, ya que ninguna de las herramientas estudiadas era factible utilizarla.
- Se realizó el Análisis y Diseño de la Herramienta de Generación de interfaces en Angular JS para Symfony 3, empleando PHP v7.0 como lenguaje de programación, Apache v2.4 como servidor web, PhpStorm v2016.1 como entorno integrado de desarrollo, Visual Paradigm v8.0 como herramienta de modelado; además de utilizar Symfony v3.1.* como marco de trabajo y Angular JS v1.5 como lenguaje del código de salida sirvieron de base para llevar a cabo la correcta implementación de la solución.
- Aplicar los estándares de codificación para la obtención del generador de código permitió que la implementación se llevara a cabo de forma responsable y organizada.
- Se implementó el Generador de interfaces en Angular JS para Symfony 3, logrando reducir el tiempo empleado en la generación de interfaces. Se validó la solución desarrollada empleando técnicas para la validación de los requisitos, métricas para la validación del diseño y pruebas de caja blanca y caja negra para la validación de la aplicación.
- La verificación del diseño realizado, mediante la aplicación de las métricas TOC y RC, evidenció una baja *responsabilidad* y *complejidad de implementación*, la cual favoreció la *reutilización* corroborándose así una correcta asignación de *responsabilidades*.
- La validación de las variables de la investigación y la aplicación de la técnica Iadov permitió medir la satisfacción del cliente respecto al control y disponibilidad de la información con el generador de interfaces, donde los resultados obtenidos se traducen en un elevado agrado por parte del cliente con la solución. Todo esto permitió garantizar que se ha obtenido un software que responde a las necesidades del usuario y correctamente funcional.

RECOMENDACIONES

Para futuras investigaciones se recomienda lo siguiente:

- Hacer extensivo el uso de la herramienta al resto de los centros productivos de la UCI, para futuras mejoras en el tiempo de desarrollo.
- Que se le de mantenimiento frecuentemente para lograr mejoras en sus funcionalidades.

REFERENCIAS BIBLIOGRÁFICAS

ALEXANDER, C., ANGEL, S. y ISHIKAWA, S., 1980. *Un lenguage de patrones: ciudades, edificios, construcciones= A pattern language*. S.l.: Editorial Gustavo Gili. ISBN 84-252-0985-4.

BERGMANN, S., 2006. *PHPUnit*. S.l.: s.n.

BRAT TECH LLC, 2013. What Is Angular? [en línea]. [Consulta: 13 enero 2017]. Disponible en: https://docs.angularjs.org/guide/introduction.

CANDELA, L., 2011. Ext Designer 1.2 is Now Available. *Sencha* [en línea]. [Consulta: 21 junio 2017]. Disponible en: https://www.sencha.com/blog/ext-designer-1-2-final-is-here/.

CHANG, A.A.H. y CARBONELL, N.P., 2013. *"GENERADOR DE CÓDIGO PARA EL SISTEMA ALAS HIS"* [en línea]. Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas. La Habana: UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS. [Consulta: 9 diciembre 2016]. Disponible en: https://repositorio_institucional.uci.cu/jspui/bitstream/ident/8392/2/TD_06391_13.pdf.

CHAUDHARY, M. y KUMAR, A., 2014. *PhpStorm Cookbook*. S.l.: Packt Publishing Ltd. ISBN 1-78217-388-9.

CUSCÓ, M.B.I. y WELLS, J.G., 2009. La satisfacción del profesor de Educación Física. *Educación Física y Deporte*, vol. 27, no. 2, pp. 27-35.

DAVIS, A., OVERMYER, S., JORDAN, K., CARUSO, J., DANDASHI, F., DINH, A., KINCAID, G., LEDEBOER, G., REYNOLDS, P. y SITARAM, P., 1993. Identifying and measuring quality in a software requirements specification. *Software Metrics Symposium, 1993. Proceedings., First International*. S.l.: IEEE, pp. 141-152. ISBN 0-8186-3740-4.

DONAL, M.C. y ZURBANO, Y.A.R., 2013. *Herramienta para la generación de código de applets javacard a partir de diagramas de estado* [en línea]. Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas. Ciudad de La Habana: UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS. [Consulta: 9 diciembre 2016]. Disponible en: https://repositorio_institucional.uci.cu/jspui/bitstream/ident/8176/2/TD_06446_13.pdf.

ESPINOSA, S.G., LARA, R.B. y CUTIÑO, D.D., 2008. ESTRATEGIA PARA LA APLICACIÓN DE PRUEBAS DE CAJA BLANCA Y CAJA NEGRA EMPLEANDO LA METODOLOGÍA RUP. ,

FIGUEROA, R.G., SOLÍS, C.J. y CABRERA, A.A., 2008. Metodologías tradicionales vs. Metodologías ágiles. *Universidad Técnica Particular de Loja, Escuela de Ciencias en Computación.(En línea), Disponible en: http://adonisnet. files. wordpress. com/2008/06/articulo-metodologia-de-sw-formato. doc*,

FOWLER, M., 2003. La nueva metodología. *Programación extrema*,

GARRIDO, S.A., 2009. Introducción a Qt. *Programación gráfica en c++ con QT*, vol. 4.

GAUCHAT, J.D., 2012. *El gran libro de HTML5, CSS3 y Javascript*. S.l.: Marcombo. ISBN 84-267-1782-9.

GERACI, A., KATKI, F., MCMONEGAL, L., MEYER, B., LANE, J., WILSON, P., RADATZ, J., YEE, M., PORTEOUS, H. y SPRINGSTEEL, F., 1991. *IEEE standard computer dictionary: Compilation of IEEE standard computer glossaries*. S.l.: IEEE Press. ISBN 1-55937-079-3.

HOGAN, B.P., 2011. *HTML 5 & CSS 3*. S.l.: O'Reilly Germany. ISBN 3-89721-316-8.

JAIN, N., BHANSALI, A. y MEHTA, D., 2015. AngularJS: A modern MVC framework in JavaScript. *Journal of Global Research in Computer Science*, vol. 5, no. 12, pp. 17-23.

KAISLER, S.H., 2005. *Software paradigms*. S.l.: John Wiley & Sons. ISBN 0-471-70357-5.

KERSKEN, S., 2012. *Apache 2.4*. S.l.: Galileo Press. ISBN 3-8362-1777-5.

LANG, D.T. y TEAM, C., 2012. XML: Tools for parsing and generating XML within R and S-Plus. *R package version*, pp. 3.9-4.1.

LARMAN, C., 2014. *Applying UML and Patterns: An Introduction to Object Oriented Analysis and Design and Interative Development*. 2da. S.l.: Pearson Education India. ISBN 81-317-6236-X.

LOELIGER, J. y MCCULLOUGH, M., 2012. *Version Control with Git: Powerful tools and techniques for collaborative software development*. S.l.: O'Reilly Media, Inc. ISBN 1-4493-4505-0.

LÓPEZ RODRÍGUEZ, A. y MAURA, V.G., 2004. La Técnica de Iadov. *Ministerio de Educación Superior. La Nueva Universidad Cubana. La Habana*,

LORENZ, M. y KIDD, J., 1994. *Object-oriented software metrics: a practical guide*. S.l.: Prentice-Hall, Inc. ISBN 0-13-179292-X.

MARWEDEL, P. y GOOSSENS, G., 2013. *Code generation for embedded processors*. S.l.: Springer Science & Business Media. ISBN 1-4615-2323-0.

MICROSOFT, 2017. *Revisiones de código y estándares de codificación* [en línea]. 2017. S.l.: s.n. [Consulta: 5 mayo 2017]. Disponible en: https://msdn.microsoft.com/es-es/library/aa291591(v=vs.71).aspx#.

MORENO, M. y ELENA, M., 2012. Análisis y estudio de las herramientas libres para el desarrollo de aplicaciones móviles en teléfonos celulares con sistema operativo android, y la construcción de una aplicación para entretenimiento de usuarios en la carrera de informática y sistemas computacionales de la universidad técnica de Cotopaxi. ,

MORENO, P.J.M., 2003a. *Especificación del interfaz de usuario: de los requisitos a la generación automática*. S.l.: Universitat Politècnica de València.

MORENO, P.J.M., 2003b. *Interfaz de usuario: de la especificación a la generación automática - TesisPjmolina.pdf* [en línea]. Tesis Doctoral. España: Universidad politécnica de Valencia. [Consulta: 24 octubre 2016]. Disponible en: http://pjmolina.com/papers/TesisPjmolina.pdf.

MOSELEY, R., 2007. *Desarrollo de aplicaciones Web*. S.l.: s.n. ISBN 84-415-2265-0.

OLSSON, M., 2016. *PHP 7 quick scripting reference*. S.l.: Apress. ISBN 1-4842-1922-8.

ORCHARD, L.M., PEHLIVANIAN, A., KOON, S. y JONES, H., 2009. *Professional JavaScript Frameworks: Prototype, YUI, ExtJS, Dojo and MooTools*. S.l.: Wrox Press Ltd. ISBN 0-470-38459-X.

ORÉ B, A., 2014. Pruebas Unitarias Cap 1 - Software Testing and QA - Pruebas Unitarias. En: 00000 [en línea]. [Consulta: 5 abril 2016]. Disponible en: http://www.calidadysoftware.com/testing/pruebas_unitarias1.php.

PARADIGM, V., 2013. Visual paradigm for uml. En: 00014, *Visual Paradigm for UML-UML tool for software application development*,

PATIÑO CAMARGO, W., SUÁREZ VILLEGAS, R. y OTHERS, 2014. Optimización del proceso de pruebas de software. En: 00000 [en línea], [Consulta: 21 mayo 2016]. Disponible en: http://repositorioacademico.upc.edu.pe/upc/handle/10757/336106.

PHP. [en línea], 2001. [Consulta: 13 diciembre 2016]. Disponible en: http://php.net/manual/es/intro-whatis.php.

PRESSMAN, R.S., 2010. Software Engineering: A Practitioner's Approach, 7/e, RS Pressman & Associates. *Inc., McGraw-Hill, ISBN*, vol. 73375977.

QUINTERO, J.B., DE PÁEZ, R.A., MARÍN, J.C. y LÓPEZ, A.B., 2012. Un estudio comparativo de herramientas para el modelado con UML. *Revista universidad eafit*, vol. 41, no. 137, pp. 60-76.

RISCHPATER, R., 2014. *Application Development with Qt Creator*. S.l.: Packt Publishing Ltd. ISBN 1-78439-922-1.

RODRÍGUEZ SÁNCHEZ, T., 2015. *Metodología de desarrollo para la Actividad productiva de la UCI*. 2015. S.l.: Habana.

SANTANA, C.A. y CONI, C.V., 2011. *Un metalenguaje de programación orientado al diseño de interfaces gráficas*. S.l.: Facultad de Informática.

SOMASUNDARAM, R., 2013. *Git: Version control for everyone*. S.l.: Packt Publishing Ltd. ISBN 1-84951-753-3.

SOMMERVILLE, I., 2016. *Software engineering*. S.l.: Pearson. ISBN 0-13-394303-8.

SPINELLIS, D., 2012. Git. *IEEE software*, vol. 29, no. 3, pp. 100-101.

TÉLLEZ, A.L., 2014. *"Módulo para la generación de interfaces gráficas de usuario para el marco de trabajo Sauxe_v2.2"* [en línea]. Trabajo de Diploma para optar por el título de Ingeniero en Ciencias

Informáticas. S.l.: Universidad de las Ciencias Informáticas. Disponible en: https://repositorio_institucional.uci.cu/jspui/bitstream/ident/8952/2/TD_07302_14.pdf.

UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS, 2014. Modelo del Profesional. [en línea]. S.l.: [Consulta: 20 junio 2017]. Disponible en: http://intranet2.uci.cu/sites/default/files/pdf_formacion/Modelo_del_Profesional.pdf.

VLADIMIR, M.B.C., 2012. *"SISTEMA DE GESTIÓN DE PEDIDOS Y PROFORMAS DINÁMICAS POR INTERNET PARA LA EMPRESA JIMEMOR CIA.LTDA UTILIZANDO SYMFONY"* [en línea]. Trabajo de Diploma. Ecuador: UNIVERSIDAD TÉCNICA DEL NORTE. Disponible en: http://repositorio.utn.edu.ec/bitstream/123456789/1814/1/Tesis%20formato%20Pdf.pdf.

WEB, S., 2015. *W3C. 2015*. S.l.: s.n.

YANA, M. y PARODI, J.C., 2014. Optimizaci? n del proceso de pruebas de software. ,

ZANDSTRA, M., 2016. Testing with PHPUnit. *PHP Objects, Patterns, and Practice*. S.l.: Springer, pp. 435-464.

ANEXOS

Anexo 1: Encuesta para identificar tempo de desarrollo.

Encuesta sobre el uso del marco de trabajo AngularJS en los centros de desarrollo

Esta encuesta va dirigida a los desarrolladores del CEIGE que utilicen como marco de trabajo AngularJS. Su objetivo es determinar el tiempo promedio que demoran en diseñar la interfaz gráfica de usuario empleando este marco de trabajo.

1) ¿Usted ha desarrollado alguna interfaz con AngularJS?
   a) _____ si
   b) _____ no
2) ¿Ha desarrollado en esa(s) interfaz(es) algún CRUD?
   a) _____ si
   b) _____ no
3) ¿Utiliza alguna herramienta para generar interfaces en AngularJS?
   a) _____ si
   b) _____ no
4) ¿Qué tiempo demora crear un CRUD?
   a) _____ <30 min
   b) _____ 1 a 3 horas
   c) _____ 1 a 3 días
   d) _____ otro ¿Cuál?_____

Anexo 2: Encuesta de satisfacción aplicada

Preguntas para la validación de la investigación a partir de la técnica de Iadov

¿Le gustaría que se desarrolle un generador de código?

¿Cree realmente necesario un generador de código?

¿Le gustaría hacer uso del generador de código propuesto, para desarrollar interfaces gráficas de usuarios?

¿Considera usted oportuno continuar desarrollando las interfaces gráficas de usuarios cada vez que se va a comenzar el proceso de desarrollo de un proyecto, a pesar de tener un generador que le facilita el trabajo?

¿El generador de código contribuye a mejorar el tiempo de desarrollando las interfaces gráficas de usuarios?

¿Qué tiempo demora en crear un CRUD haciendo uso de la herramienta desarrollada?

a) _____ <30 min
b) _____ 1 a 3 horas
c) _____ 1 a 3 días
d) _____ otro ¿Cuál?_____