Método para complementar la generación de códigos de aplicaciones web desde el diagrama de clases UML *(Method to complement the generation of web application codes from UML Class Diagram)*

Adolfo Vega Fajardo[i]

Resumen

En el desarrollo de aplicaciones web, el mayor esfuerzo se centra en la generación de código, sin embargo, los trabajos de investigación al respecto se enfocan en resolver problemas de diseño, por otro lado, la herramienta case genera código limitado o incompleto y no tiene las especificaciones formales para el desarrollo de aplicaciones web. En este artículo, se propone un nuevo enfoque para completar la generación de códigos a partir del diagrama de clases UML. El enfoque se basa en analizar el archivo del diagrama de clases para luego generar el código. Se ha definido un modelo de diseño como meta-modelo, el cual tiene el formalismo de la metodología de desarrollo de software por capas y está representado por un perfil con extensión XML, de este modo, se extrae las líneas de código XML del archivo que contiene el diagrama de clases, se comparan con el meta-modelo a través de algoritmos y después se genera el código para aplicaciones web. Para validar la propuesta, se ha desarrollado una herramienta situada entre la generación del código automático y la generación de código manual. A partir del uso del indicador *tiempo de generación*, se demuestra que la generación del código se reduce hasta en un 98 %.

Palabras clave: técnica de programación, programación orientada a objetos, arquitectura de capas, método de generación de códigos.

[i] AVF Perú. adolfovega71@hotmail.com

Abstract

In the development of web applications the greatest effort is focused on the generation of code, however, the research work on code generation is focused on solving design problems, in addition the case tools generate limited or incomplete code and do not have the specifications formal for web application development. In this article, we propose a new approach to complete the generation of codes from the UML class diagram, the approach is based on analyzing the file of the class diagram and then generating the code; we have defined a design model as a meta-model, the model has the formalism of the architecture by layers and is represented by a profile with an XML extension, the lines of XML code are extracted from the file that contains the class diagram, they are compared with the meta-model through algorithms and then the code is generated for web applications; a tool has been developed to validate our proposal between the generation of automatic code and the generation of manual code; using the generation time indicator, we show that code generation is reduced by more than 98%.

Keywords: Programming Technique, Object Oriented Programming, Layer Architecture, Code Generation Method.

I. Introducción

Las aplicaciones web a nivel empresarial tienen ventajas frente a las aplicaciones convencionales, por ejemplo, son multiplataforma (sistemas operativos y navegadores), lo cual hace que se adapten fácilmente a los dispositivos móviles, además, están disponibles desde cualquier parte del mundo y las actualizaciones son inmediatas, por lo que las empresas promueven que los sistemas de información sean desarrollados o migrados a aplicaciones web.

El desarrollo de aplicaciones web requiere de algunos componentes básicos, por ejemplo, se requiere de un modelo de datos para la información del sistema, un lenguaje programación para la programación y un marco de trabajo para el diseño de páginas web, estos componentes deben estar relacionados; por ejemplo, la página web requiere de un lenguaje de programación para ejecutar la funcionalidad de la aplicación y, a su vez, necesita un modelo de datos para guardar la información del sistema.

La generación de código (GC) es considerada como un elemento importante dentro del ciclo de desarrollo de las aplicaciones web, de mismo modo, concentra la mayor cantidad de tiempo y esfuerzo, debido a que se genera en forma manual.

La GC se realiza después haber terminado el diseño del sistema y posee diferentes tipos de código: el primer tipo sirve para las páginas web y utiliza el lenguaje de programación como JAVA SCRIPT, CSS, y/o HTML, el segundo sirve para la funcionalidad de la aplicación, y el último sirve para acceder a la base de datos. Todos utilizan lenguajes de programación como el java. El hecho de generar código requiere de tiempo, conocimiento y esfuerzo, sobre todo cuando el código es utilizado para aplicaciones web.

La GC basada en aplicación web es una técnica de programación que tiene como objetivo crear líneas de programación basadas en un lenguaje de programación que expresan instrucciones coherentes y comandos lógicos de la programación orientadas a objetos o la programación orientada a aspectos.

En estas últimas décadas se han creado generadores que crean código en forma automática desde los diagramas UML con el objetivo de reducir el tiempo en la programación (el código es parte de la programación), sin embargo, estos generadores crean códigos incompletos o insuficientes, porque solo abarcan una parte, la generación debe abarcar desde la elaboración de las páginas web hasta la generación de código para la funcionalidad del sistema y el acceso a la base de datos.

La GC se aplica en diferentes niveles de la programación, así, se ha identificado tres grupos de generadoras: el primer grupo son los generadores para las páginas web, donde el código se encuentra creado y disponible para ser usado como plantillas en las páginas web, por ejemplo, el jQuery es un marco de trabajo (framework) que contiene código abierto y puede ser usado para crear formularios, lista de información y ventanas emergentes que sirven para advertencias o mensajes;

el segundo grupo son los generados que están relacionado con la generación de código a través del entorno desarrollado integrado o, en sus siglas en inglés, IDE (Integrated Development Environment), el cual contiene herramientas que generan código básicos, tales como el NetBean, que es un IDE que tiene como base el lenguaje de programación java y cuenta con opciones y/o herramientas que permiten generar código, así como la opción para generar métodos Get() y Set() que son utilizados por las clases java para obtener y asignar información de la aplicación; por último, el tercero es el grupo de generados de códigos de tipo case, que permite generar código a partir de los diagramas de clases del UML, donde destaca IBM Rational Architect (IBM, 2018), ArgoUML (Tigris.org, 2018) y otros, sin embargo, estos cases generan código incompleto (Domínguez, 2012). La tabla 1 muestra los generadores de código por grupos.

El resto del artículo está estructurado de la siguiente manera: en la sección 2 se presenta la motivación del artículo, la sección 3 presenta la programación por capas, la sección 4 muestra una visión general del enfoque propuesto, en la sección 5 se procede a la validación, la sección 6 presenta las pruebas y los resultados del enfoque y, por último, en la sección 7 se exponen las conclusiones.

Tabla 1: Generadores de código por grupos

Grupo 1: Páginas web	**Referencia**	**Lenguaje de Programación**	**Genera Código**	**Tipo**
JQuery	(Resig, 2019)	JavaScript	Formularios, ventanas, mensajes, menús, listados, tablas, etc.	Marco de Trabajo para HTML
PrimeFace	(PrimeFaces, 2019)			Marco de Trabajo para Java Server Face (JSF).
RichFace	(JBOSS, 2019)			
Grupo 2: IDE				
Eclipse	(Eclipse Luna, 2018)	Java, PHP	Métodos, Clases, Propiedades, etc.	IDE que contiene opción y/o herramientas para generar código.
NetBeans		Java, PHP		
Grupo 3: Cases				
ArgoUML	(Tigris.org, 2018)		Genera código	Case
IBM Rational Architect	(IBM, 2018)		Genera código	Case
StartUML	(MKLab, 2019)		No genera código	Case

2. Motivación

La GC es un proceso que permite producir líneas de código basado en un lenguaje de programación. Para su ejecución, es necesaria la experiencia en programación web. En proyectos de tecnología de información, casi el 40 % se dedica a la programación.

El estudio de Piraquive et al. (Piraquive, 2015) analiza las causas que originan que los proyectos de software fracasen, donde se destaca la administración y las metodologías para el desarrollo de software. La figura 1 exhibe las causas que producen el fracaso del proyecto de desarrollo de software.

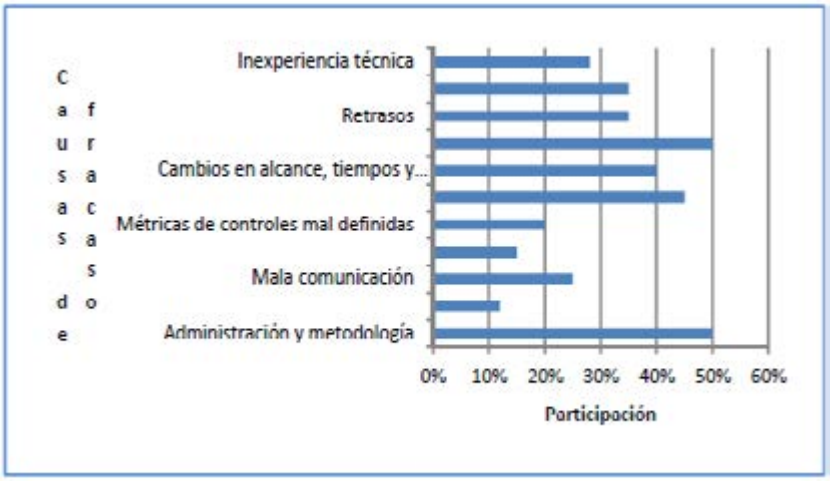


Figura 1. Causas de fracasos en Proyectos de Software

El fracaso de los proyectos se debe a la ausencia de una metodología de desarrollo de software, además, en el ciclo de desarrollo se considera las etapas de diseño y generación de código, donde ambas etapas deben estar relacionadas pues son dependientes, así, cuando está mal el diseño se afecta el código

Se ha revisado una serie de estudios relacionados con la generación de código desde el diagrama UML, por ejemplo, el estudio (Domínguez, 2012), literatura científica que proporciona una vista y comparación de 53 propuestas sobre cómo implementar máquinas de estado y también realiza una comparación de métodos usando patrones para generar código, sin embargo, estos métodos son usados más para resolver problemas del diseño.

En otro estudio (Manoli, 2011), toma de entrada un diagrama de clases del UML que tiene solo aspectos específicos estáticos y devuelve un diagrama donde las clases se han extendido en operaciones básicas (CRUD), sin embargo, al igual

que Domínguez et al., este estudio también se centra en la etapa de diseño, donde usa el método basado en reglas de diseño para transformar un diseño estático a otro diseño que contiene operaciones.

Después de haber revisado más estudios (Domínguez, 2012), (Bennett, 2010), (Pinto, 2012), (Piraquive, 2015) y (Rincón, 2011) que tienen como principio el concepto de la generación de código, se concluye que estos estudios no han tenido la madurez en la comunidad de desarrolladores, porque la generación de código debería resolver el problema de escribir o producir código bajo un lenguaje programación, más aún cuando se trata de aplicaciones web que requiere de código para la funcionalidad y el acceso a la base de datos del sistema. En la figura 2, con la ayuda de un lenguaje de programación web Java Server Face y el lenguaje de programación Java, se muestra el código que se requiere para la página web, la funcionalidad y el acceso a las bases de datos, además, se resalta la forma o lógica de programación.

Otro punto importante que se encontró en la revisión de los estudios fue que existen pocas herramientas case que permite generar código (Domínguez, 2012), además, las herramientas cases actuales, tales como IBM Rational Architec (IBM, 2018), Eclipse Luna (Eclipse Luna, 2018), ArgoUML (Tigris.org, 2018) y otros, generan código incompleto (Domínguez, 2012) y solo sirven para diseñar o generar código sin el formalismo de metodología de desarrollo de software para aplicaciones web.

Sin embargo, las herramientas cases han sido usadas por los investigadores para poner en práctica sus trabajos de generación código, como en el estudio de (Manoli, 2011) , donde se usa el ATL (Transformation Language) del Eclipse como una herramienta case para poner en práctica su trabajo de investigación.

Además, Eclipse cuenta con otras opciones que recientemente han sido introducidas, tales como el componente Papyrus (Papyrus, 2018) que permite la creación de metamodelo para el diseño específico y también tiene una opción que permite la exportación del diseño al formato XML (Extensible Markup Language) con las propiedades del UML (Lenguaje de Modelado Unificado).

Se ha considerado tres aspectos importantes que son la base del presente aporte de investigación en la generación de código como un complemento después del diseño: primero, el mayor esfuerzo en el desarrollo de aplicaciones web no es el diseño, es la generación de código basado en un lenguaje de programación; segundo, el metamodelo y el formato XML permiten desarrollar un modelo de diseño y ser comparado por medio del XML con los diagramas de clases; y tercero, no existe una herramienta case que genere código completo para aplicación web, debido a que carecen de una metodología.

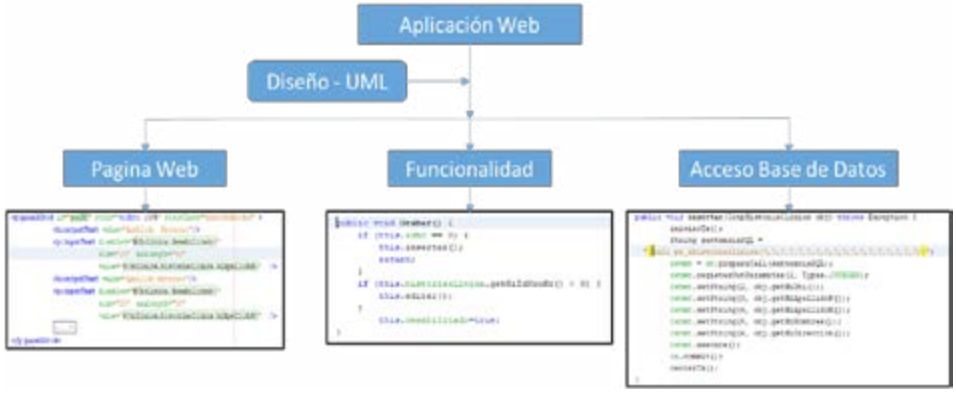


Figura 2. Generación de código para la aplicación web

III. Programación web por capas

La programación web (PW) es un conjunto instrucciones o códigos que permite manipular el funcionamiento de una aplicación, las cuales deben estar orientadas a las tecnologías web. La programación orientada a objeto (POO) ha sido usada en la PW como un medio para especificar y describir el comportamiento de los objetos del sistema y permite mejor la calidad de la programación (Clemente, 2011), sin embargo, esto es insuficiente en la PW, porque se requiere de código avanzado, no solo para especificar el comportamiento de los objetos, sino que también es necesario para las páginas web y para acceder a la base de datos.

El modelo de desarrollo por capas (MDC) y el modelo vista controlador (MVC) han sido ampliamente usados (Basso, 2016) (Garrigós, 2010) en el desarrollo de aplicaciones web.

La programación web basada en el modelo de desarrollo por capas (PWMDC), es un método que permite programar en capas (Basso, 2016) y usa el concepto de la arquitectura cliente servidor, donde las capas están compuestas por presentación, negocio y datos. Se ha identificado e incluido una cuarta capa: la "capa entidad", porque a través de esta capa se permite el mapeo de atributos entre una base de datos y el modelo de objetos. Se han desarrollado marcos de trabajo, uno de ellos es el Hibernate (Hat, 2019), donde el código es reutilizado y aplicado en el desarrollo de aplicaciones a través de librerías que permite generar código para el mapeo de la base de datos.

La PWMDC se torna compleja cuando se genera código para cada capa, y peor aun cuando las capas deben estar interactuando entre ellas, por lo que es necesario describir el contenido de cada capa y el código que se genera en ellas. A continuación, se detalla las capas:

Capa de presentación: Está basada en la arquitectura cliente servidor. Esta capa contiene las páginas web y se genera código de tipo HTML, Java Script y CSS al lado del cliente. Recientemente, los lenguajes de programación han incorporado nuevos marcos de trabajo. El lenguaje de programación Java ha desarrollado el Java Server Faces (Microsystems, 2016) que permite construir interfaces a nivel de usuario al lado del cliente, tales como formularios, tablas, listas, botones, entre otros componentes gráficos. El PHP (Personal Hypertext proceso) (PHP, 2018) es otro lenguaje de programación que es usado también para elaborar aplicaciones web.

Capa de negocio: Contiene los programas o clases que representan la funcionalidad de la aplicación al lado del servidor. En las clases se genera código para los métodos del CRUD (Create, Read, Update y Delete) que sirven en la funcionalidad del sistema. El código es generado bajo un lenguaje de programación, por ejemplo, el lenguaje de programación Java. El código que se genera en la capa de negocio es representado por los diagramas de comportamiento del UML, tales como el diagrama de clases y estados.

Capa de datos: Contiene las clases que permiten intercambiar información entre la aplicación web con la base de datos, las cuales a su vez interactúan con las clases de la capa de negocio. En la capa de datos se genera código para los métodos que son utilizados por las clases de la capa de negocio, por ejemplo, el método grabar de la capa de negocio tiene un método insertar de la capa de datos, este método permite insertar nuevos registros en la tabla de la base de datos desde la capa de negocio, de igual forma sucede con los métodos modificar – update, buscar – select, y eliminar – delete.

Capa de entidad: Contiene las clases que representan la base de datos del sistema. Siguiendo los principios del modelado de la base de datos, se genera código que representa los atributos, relaciones, claves primarias y claves foráneas de la base de datos, por ejemplo, se genera código para los atributos de la tabla y se representa en las clases como objetos instanciados con sus respectivos métodos get() y set() que sirve para obtener y poblar la información, y la relación entre las tablas es representada por objetos instancias entre clases, facilitando el intercambio de información entre las capas de negocio y la capa de datos. En la figura 3 se muestra la estructura de la PWMDC con sus respectivas capas.

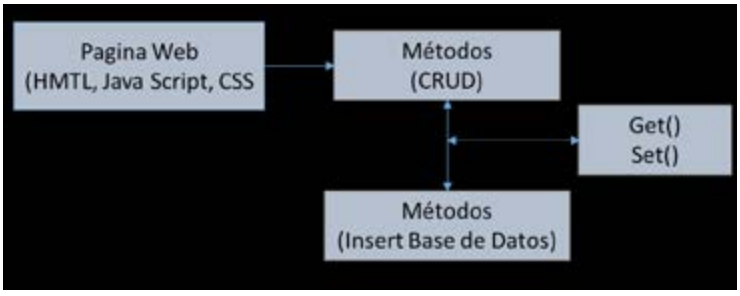


Figura 3. Estructura de la PWMDC

IV. Visión general

En esta sección, se describe la visión general del enfoque propuesto, el diagrama de clases se toma como entrada para ser comparado con un meta-modelo que contiene el diseño basado en el modelo de desarrollo por capas (MDC), después, a través de un método, se genera código basado en la PWMDC. La figura 4 muestra la visión general del enfoque.

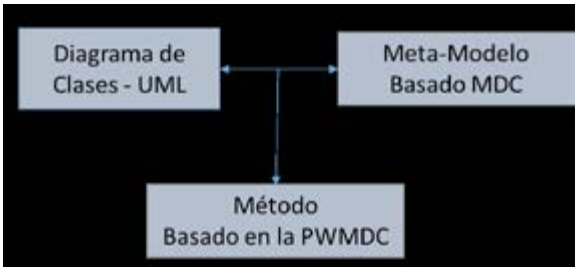


Figura 4. Visión General de la generación de códigos

Diagrama de clases del UML

El diagrama de clases es un gráfico que representa el comportamiento del sistema en forma gráfica y es parte del diseño de software. Debe desarrollarse antes de la generación y, a su vez, contar con especificación formal para aplicaciones web, tal como el modelo de desarrollo por capas (MDC) (Basso, 2016).

Tomando como base el trabajo de investigación de Manoli et al. (Manoli, 2011), sobre método basado en reglas de diseño, se ha creído conveniente elaborar reglas de diseño que sirven como guía a los desarrollados de software para la elaboración del diagrama de clases con el formalismo de la MDC, las cuales se detallan a continuación:

En la capa de negocio: Se ha definido las reglas para las clases que representan la funcionalidad del sistema. La tabla 1 muestra las reglas de diseño para la capa de negocio, por ejemplo, una de las reglas es definir como objetos dos variables: la primera variable se define como objeto a la clase entidad, y la segunda variable se define como objetos a la clase de datos, estos objetos son usados por los métodos del CRUD.

En la capa de entidad: Se ha definido las reglas para las clases que representa las tablas del modelado de la base de datos. La tabla 2 muestra las reglas de diseño para la capa de entidad.

En la capa de datos: Se ha definido las reglas para las clases que permiten intercambiar información entre la aplicación web con la base de datos. La tabla 3 muestra las reglas de diseño para la capa de datos.

Meta–modelo basado en el MDC

El meta-modelo permite especificar el modelo de diseño propuesto basado en el MDC. Con ayuda de la herramienta case Eclipse con su componente Papyrus (Papyrus, 2018), se ha creado el meta-modelo para luego ser comparado con el diagrama de clases. La figura 8 muestra el meta-modelo elaborado con los stereotypes basados MDC, el cual puede ser descargado (Vega Fajardo, 2018).

El Eclipse-Papyrus soporta el XML (Extensible Markup Language) y UML, los cuales permiten crear el meta-modelo. El meta-modelo es un archivo generado que contiene las líneas de códigos XML, esto ha permitido comparar las líneas de código entre el diagrama de clases y el meta-modelo, ambos archivos contienen líneas código XML.

Método de generación basado en la PWMDC

El enfoque propuesto permite completar el ciclo de la generación de códigos desde el diseño hasta la producción, sobre todo en la parte final, así, se extrae líneas de código con extensión XML del archivo del diagrama de clases del UML, posteriormente, es comparada con un meta-modelo, para que finalmente se genere códigos basado en la PWMDC.

El Eclipse-Papyrus, que soporte el UML y XML, permite la creación del diagrama de clase a través de un archivo que contiene las líneas de código XML. La figura 5 muestra las líneas de códigos XML del diagrama de clases, de este modo, se ha identificado las líneas que sirven para el enfoque propuesto.

```
<?xml version="1.0" encoding="UTF-8"?>
<xmi:XMI xmi:version="20131001" xmlns:xmi="http://
www.omg.org/spec/XMI/20131001" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance">
<uml:Model xmi:id="_Z-f5wHAKEeiTZJpT9UPw0A"
name="Model">
<packagedElement xmi:type="uml:Package"
name="namePackage"/>
<packagedElement xmi:type="uml:Class"
name="nameClass"/>
<ownedAttribute xmi:type="uml:Property"
name="nameAttribute"/>
<ownedOperation xmi:type="uml:Operation"
name="nameOperation"/>
</uml:Model>
</xmi:XMI>
```

Figura 5. Línea de código XML del archivo del diagrama de clases

También se ha definido tres fases: fase de inicio, fase de análisis y fase de generación. La figura 5.1 muestra el enfoque propuesto denominado método de generación automática basado en la PWMDC.

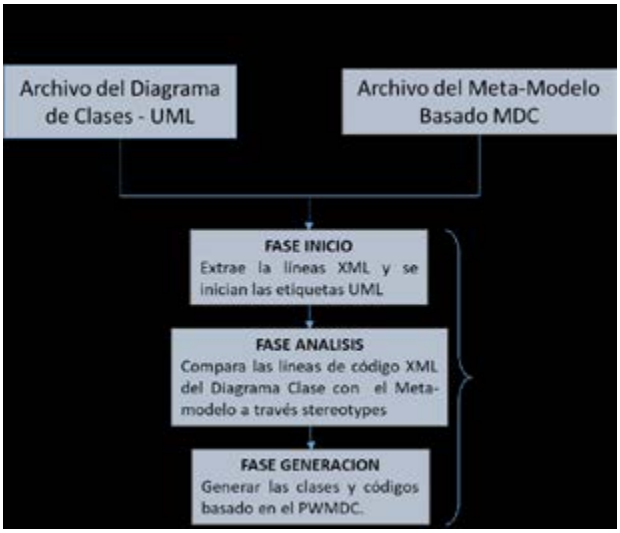


Figura 5.1. Método de generación basada en la PWMDC

Fase Inicio: En esta fase se identifican las líneas código XML del archivo diagrama de clase, se extraen las líneas de código XML y se inician las etiquetas UML.

Utilizando el lenguaje de programación Java, se ha desarrollado un algoritmo que tiene como entrada las líneas de código XML del archivo diagrama de clases, estas son relacionada con las etiquetas definidas en la tabla 4, en el caso que cumplan, pasan a la siguiente fase, el resto son rechazadas e informadas. La figura 6 muestra la clase de nombre "iniciarEtiquetas" encargada de iniciar las etiquetas, por su parte, la figura 7 muestra la clase readStereotype que contiene el algoritmo.

Tabla 2. Reglas de diseños de la capa de negocio

Etiqueta	Código de línea XML	Cód.	Reglas de Diseño
Paquete	<packagedElementxmi:type=”uml:Package” name=”namePackage”/>	RN1	Especificar un nombre al paquete
		RN2	Las clases deben estar dentro del paquete
Clases	<packagedElementxmi:type=”uml:Class” name=”nameClass”/>	RN3	La clase deben tener métodos CRUD
		RN4	Especificar el nombre del paquete
		RN6	Especificar dos variables: la primera variable se define como objeto a la clase del paquete entidad y la segunda variable a la clase del paquete de datos.
Atributo	<ownedAttributexmi:type=”uml:Property” name=”nameObjectEntidad” type=”nameClassEntidad”/>	RN7	El atributo debe instanciar la clase entidad como objeto
	<ownedAttributexmi:type=”uml:Property” name=”nameObjectDatos” type=”nameClassEntidad”/>	RN8	El atributo debe instanciar la clase datos como objeto
Métodos	<ownedOperation xmi:type=”uml:Operation” name=”nameOperation”/>	RN9	El método debe estar en la clase, el método debe representar la funcionalidad de la aplicación web.
	<Profile:MetodosNegocio nameObject=”nameObjecto” nameClassDAO=”nameClassDAO” base_Operation=”namePackage”/>	RN10.1	Especificar el stereotype “MetodosNegocio”
		RN10.2	Especificar el nombre del objeto instanciado
		RN10.3	Especificar el nombre de la clase DAO y el nombre del paquete

Tabla 3. Reglas de diseños de la capa de entidad

Etiqueta	Código XML	Cód.	Reglas de Diseño
Paquete	<packagedElement xmi:type=”uml:Package” name=”namePackage”/>	RE1	Especificar un nombre al paquete
		RE2	Las clases deben estar dentro del paquete
		RE3	Debe asignar el StereoType “Entidad”
Clases	<packagedElement xmi:type=”uml:Class” name=”nameClass”/>	RE4.1	La clase no deben tener métodos
		RE4.2	La clase se deben señalar un primer key (idNamePK)
		RE4.3	La clase se debe asignar el StereoType “Entidad”
		RE4.4	Especificar el nombre del paquete
Atributos	<ownedAttribute xmi:type=”uml:Property” name=”nameAttribute”/>	RE5	La clase debe tener por lo menos un atributo
		RE6	El atributo debe contar un tipo de datos (Integer, String, boolean, int.)
Property (idNamePK)	<Profile:idNamePK base_ Property=”nameIdEntidad”/>	RE7	El atributo id de la clase debe tener el StereoType “idNamePK”
Asociación (Entre Clases)	<packagedElement xmi:type=”uml:Association” name=”nameAsociacion” visibility=”public” >	RE8.1	Especificar el stereotype “EntidadChildrenAsoc”
	<Profile:EntidadChildrenAsoc idNameToFK=”idNameFK”/>	RE8.2	Especificar idNameToFK en la relación ambas clases

Tabla 4. Reglas de diseño de la capa de datos

Etiqueta	Código XML	Cód.	Reglas de Diseño
Paquete	<packagedElement xmi:type="uml:Package" name="namePackage"/>	RC1	Especificar un nombre al paquete
		RC2	Las clases deben estar dentro del paquete
Clases	<packagedElement xmi:type="uml:Class" name="nameClass"/>	RC3	Especificar el nombre del paquete
		RC4.1	La clase deben tener métodos que interactúan con las tablas del modelado de datos
Métodos	<ownedOperation xmi:type="uml:Operation" name="nameOperation"/>	RC5	El método debe estar en la clase
	<Profile:MetodosDatos nameObject="" nameTabla="" nameProcedure=""/>	RC6.1	Especificar el stereotype "MetodosDatos"
		RC6.2	Especificar el nombre del objeto, nombre de la tabla y el nombre del procedimiento almacenado

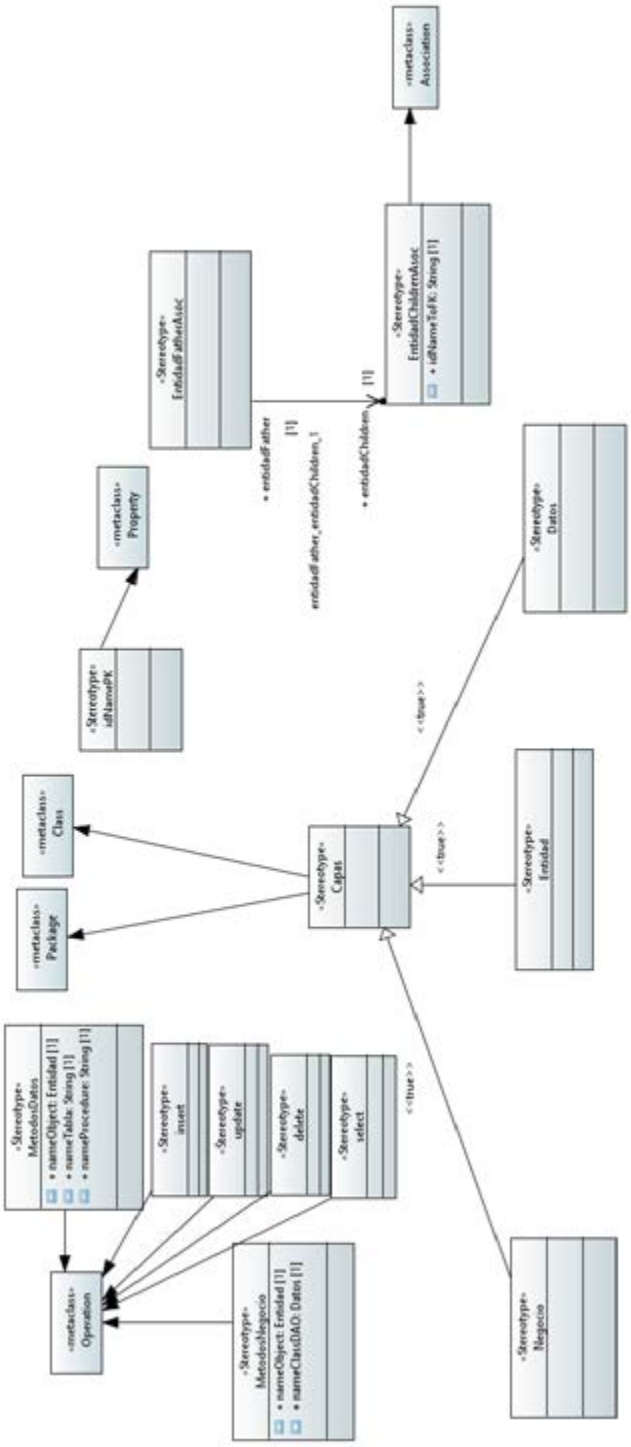


Figura 8. Meta-modelo basado en la MDC

Tabla 1. Etiquetas con las líneas de código XML

Etiqueta UML	Línea de código XML
Paquete	packagedElement xmi:type=”uml:Package”
Clase	packagedElement xmi:type=”uml:Class”
Atributos	ownedAttribute xmi:type=”uml:Property”
Metodos	ownedOperation xmi:type=”uml:Operation”

```
private void iniciarEtiquetas() {
    etiqueta[0]="uml:Package";
    etiqueta[1]= "uml:Class";
    etiqueta[2]= "uml:Property";
    etiqueta[3]= "uml:Operation";
    etiqueta[4]= "uml:Association";
}
List<Lineas> cargarRLst(List<Lineas> lst) {
    iniciarEtiquetas();
    String lin="";
    rlst = new ArrayList<Lineas>();
    for (int e = 0; e < 5; e++) {
        readEtiqueta(etiqueta[e]);
    }
    return rlst;

}
```

Figura 6. Clases de inicio del algoritmo

```
private void readStereotypes(String texto_ing) {
    Character comillas=':';
    String eti = comillas+texto_ing;
    Lineas rlineas;
      for (int li = 0; li < lst.size(); li++) {
        String llx=lst.get(li).getLinea();
        if (llx.contains(eti)) {
            rlineas = new Lineas();
            rlineas.setItem(li);
            rlineas.setEtiqueta(texto_ing);
            rlineas.setLinea(llx);
            rlineas.setStereotype(texto_ing);
            slst.add(rlineas);
        }
    }
}
```

Figura 7. Algoritmo de la fase de inicio

Fase Análisis: Esta fase consiste en comparar las líneas de código XML entre el archivo del diagrama de clases con el archivo del meta-modelo, este último contiene también líneas de código XML. La figura 8 muestra el meta-modelo clasificado por capas (negocio, datos y entidad), el cual ha sido creado con la herramienta Eclipse-Papayrus con sus respetivos stereotypes que permiten definir el modelo de diseño MDC.

Se ha desarrollado un algoritmo que tiene de entrada las etiquetas del diagrama de clases definidas en la fase de inicio, posteriormente, el algoritmo busca los stereotypes del meta-modelo para ser comparados con las etiquetas del diagrama de clase, en el caso que cumplan, los stereotypes son cargados a una lista, para luego ser procesados en la fase de generación. La figura 9 muestra el algoritmo de la fase de análisis.

```
List<Lineas> analizar(List<Lineas> rlst, List<Lineas> slst) {
    String idr="";
    flst =  new ArrayList<Lineas>();
    for (int i = 0; i < rlst.size(); i++) {
        idr = rlst.get(i).getIdetiqueta();
        if (buscarStereotype(idr,slst)){
              cargarFLlst(rlst.get(i),slst);
        }else{
              nocargarFLlst(rlst.get(i));
        }

    }
    return flst;
}
private boolean buscarStereotype(String vidr, List<Lineas> slstv) {
    String slinea="";
     for (int j = 0; j < slstv.size(); j++) {
              slinea= slstv.get(j).getLinea();
              if (slinea.contains(vidr)){
                return true;

              }
        }
     return false;
}
```

Figura 9. Algoritmo de la fase de análisis

Fase Generación: Se crean los archivos que contienen las clases y se genera código basado en PWMDC. Se ha desarrollado un algoritmo que tiene de entrada los stereotypes definidos en la fase de análisis, los cuales están clasificados por capas y contienen la información del MDC, seguidamente, el algoritmo crea la clase con su respectiva cabecera y genera código para los métodos y variables, según el diagrama de clases y el meta-modelo. La figura 10 muestra el algoritmo de la fase de generación.

```
public void crear(Clase clase, String paquete, String packe) {
    try {
        this.nameclase = clase;
        this.namepaquete = paquete;
        this.namepacke = packe;
        this.stereotype = clase.getStereoClase();
        crearCabecera();
        if (stereotype=="Negocio"){
           NegocioVariables();
           NegocioMetodos();
        }
        if (stereotype=="Entidad"){
            Entidad();
        }
        if (stereotype=="Datos"){
            Datos();
        }
        cerrarArchivo();
    } catch (IOException ex) {
        Logger.getLogger(Funciones.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

Figura 10. Algoritmo de la fase generación

V. Validación

Esta sección consiste en utilizar una aplicación web concluida para validar el enfoque propuesto. Se ha utilizado la aplicación de nombre SIS_HC (usada para los servicios de salud de los centros médicos ocupacionales), desarrollada en el lenguaje de Programación Java Server Face. La funcionalidad registro de empresa del SIS_HC ha sido tomada como ejemplo. Se ha elaborado el diagrama de clases de la aplicación SIS_HC con la herramienta case Eclipse- Papyrus.

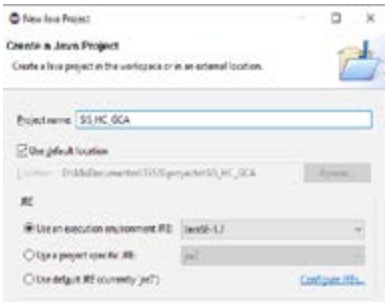
Aplicación del Meta-modelo: Con el uso de Eclipse, se explica los pasos a seguir para la aplicación del meta-modelo. El primer paso consiste en la creación de un proyecto de nombre SIS_HC (ver figura 10 (a)), el segundo paso consiste en seleccionar el tipo de lenguaje UML y los diagramas UML (ver figuras 10 (b) y (c)), y el último paso consiste en aplicar el meta-modelo (definido en el capítulo 4) al proyecto SIS_HC. La figura 11 muestra la aplicación del meta-modelo, donde se selecciona el archivo profile que tiene el meta-modelo y se agrega al proyecto.

Diagrama de Clases: Se ha elaborado el diagrama de clases usando la funcionalidad registro de empresa de la aplicación web SIS_HC, posteriormente,

se debe seleccionar los stereotypes del meta-modelo para ser aplicados en los paquetes, clases, atributos y métodos del diagrama de clases. La figura 12 muestra el diagrama de clases terminado para la capa de negocio, donde se muestra la capa entidad con sus respectivos stereotypes. El diagrama de clases se ha elaborado siguiendo las reglas de diseños descritas en las tablas 1,2 y 3.

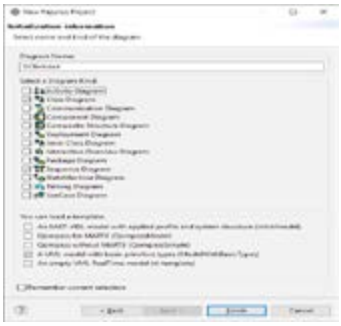
Generación del Código: Se ha elaborado una herramienta case de nombre "Generación de Código Automático" (GCA) que permite la generación de códigos, la cual fue desarrollada con el lenguaje de programación Java y tienen los algoritmos descritos en las fases (inicio, análisis y generación). La figura 13 muestra la herramienta, donde se selecciona el diagrama clases de la aplicación web SIS_HC y se generan los códigos basado PWMDC. La figura 14 muestra la generación de códigos para la clase de nombre "EmpresaBL" del paquete de negocio, esta clase representa la funcionalidad (registro de empresa) de la aplicación web, la clase entidad de nombre "Empresa" representa la tabla empresa del modelo de datos, y la clase de datos de nombre "EmpresaDAO" representa la subclase creada para insertar los datos a la tabla empresa.

Finalmente, las clases son incluidas en el proyecto de la aplicación web SIS_HC. La figura 15 muestra la interface gráfica (página web), donde se usa las clases generadas.



(a)

(b)



(c)

Figura 10. Pasos con el Eclipse: a) creación del nuevo proyecto, b) seleccionar el lenguaje diagrama, c) seleccionar el diagrama

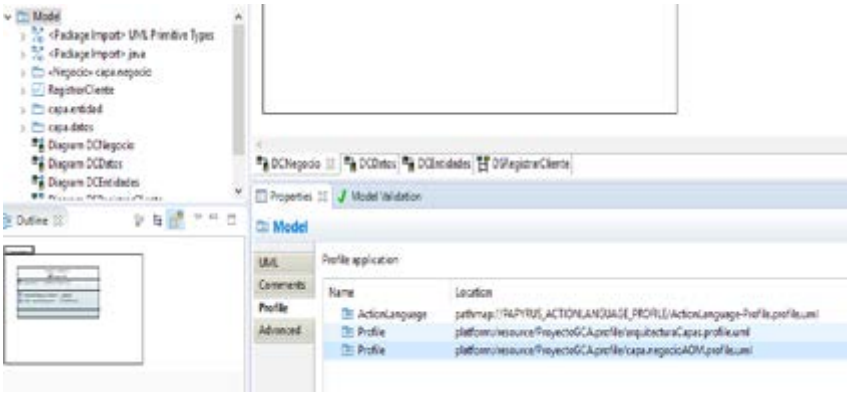
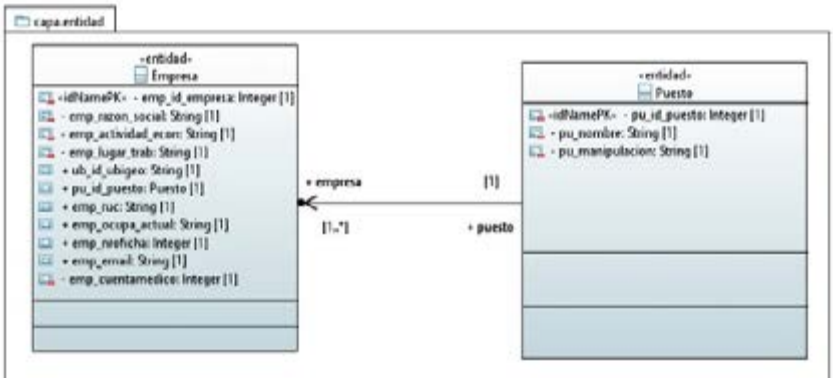
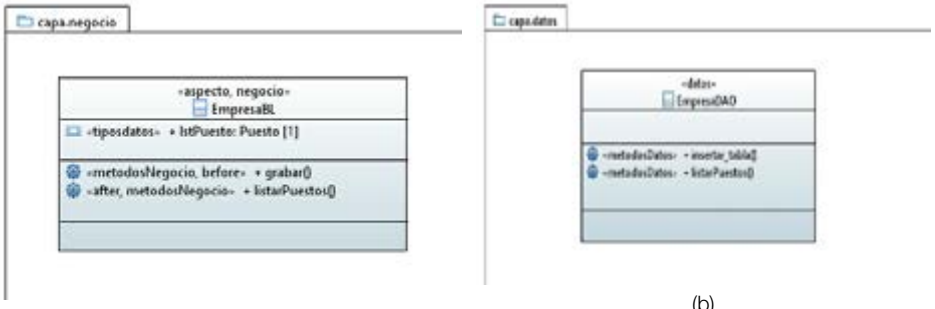


Figura 11. Aplicación del Meta-modelo con Eclipse



(a)



(c) (b)

Figura 12. Diagramas de clases (DC) del SIS_HC: a) DC de la Capa Negocio, b) DP de la Capa de Entidad, c) DC de la capa de datos

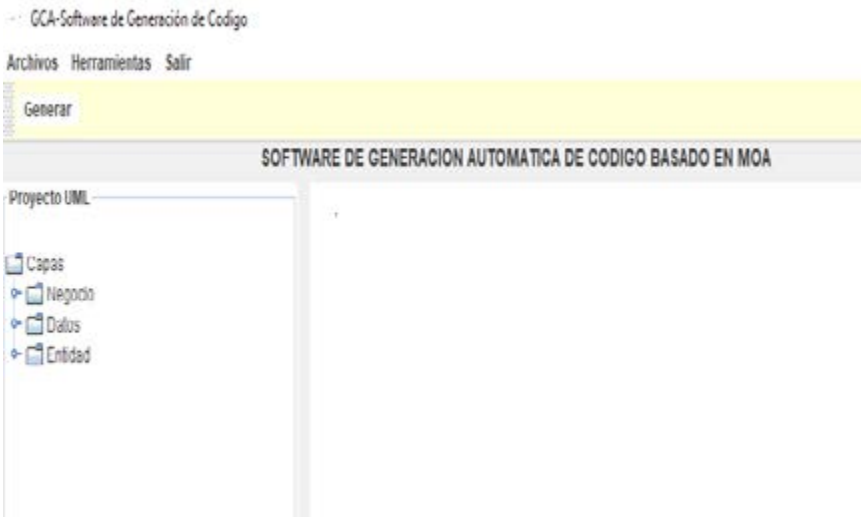


Figura 13. Herramienta Case para generar código

```
1  package capa.negocio;
2
3  import java.util.ArrayList;
4  import java.util.List;
5  import capa.datos.EmpresaDAO;
6  import capa.entidad.Empresa;
7  import capa.entidad.Puesto;
8
9   public class EmpresaBL {
10
11          private Empresa empresa = new Empresa();
12          private List<Puesto> lst_puesto = new ArrayList<Puesto>();
13          private EmpresaDAO empresadao= new EmpresaDAO();
14
15          public Empresa getEmpresa() {
16              return empresa;
17          }
18          public void setEmpresa(Empresa empresa) {
19              this.empresa = empresa;
20          }
21          public List<Puesto> getLst_puesto() {
22              return lst_puesto;
23          }
24          public void setLst_puesto(List<Puesto> lst_puesto) {
25              this.lst_puesto = lst_puesto;
26          }
27
28
29
30          public void grabar(){
31                 empresadao.grabar(empresa);
32          }
33
```

(a)

```
package capa.entidad;

import java.io.Serializable;

public class Empresa implements Serializable {

    private Integer emp_id_empresa;
    private String  emp_razon_social;
    private String  emp_actividad_econ;
    private String  emp_lugar_trab;
    private Puesto  pu_id_puesto;
    private String  emp_ruc;
    private String  emp_ocupa_actual;
    private Integer  emp_nroficha;
    private String emp_email;
    private Boolean emp_cuentamedico;

     public String getEmp_email() {
         return emp_email;
     }

     public void setEmp_email(String emp_email) {
         this.emp_email = emp_email;
     }

     public Boolean getEmp_cuentamedico() {
         return emp_cuentamedico;
     }

     public void setEmp_cuentamedico(Boolean emp_cuentamedico) {
         this.emp_cuentamedico = emp_cuentamedico;
     }
```

(b)

```
package capa.dato;

import capa.entidades.Empresa;
import java.io.Serializable;
import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.ResultSet;

public class EmpresaDAO implements Serializable{
        static  Connection conn=null;
        static  ResultSet rs=null;
        static  ResultSet rsI=null;
        static  CallableStatement cstmt = null;
        static  CallableStatement cstmtI = null;

    public void insertar_empresa(Empresa empresa, Integer id_ubigeo) throws Exception {
                conn=capa.dato.Conexion.getDataSource();
                String sentenciaSQL="{call pa_empresa_ins(?,?,?,?,?,?)}";
                cstmt = conn.prepareCall(sentenciaSQL);
                cstmt.setString(1,empresa.getEmp_razon_social().trim().toUpperCase());
                cstmt.setString(2,empresa.getEmp_actividad_econ().trim().toUpperCase());
                cstmt.setString(3,empresa.getEmp_lugar_trab().trim().toUpperCase());
                cstmt.setInt(4,id_ubigeo);
                cstmt.setString(5,empresa.getEmp_ruc());
                cstmt.setString(6,empresa.getEmp_ocupa_actual());
                rs = cstmt.executeQuery();
                conn.close();
                cstmt.close();
    }
```

(c)

Figura 14. Códigos generados: a) Clase negocio Registrar Empresa, b) Clase Entidad Empresa, c) Clase Datos EmpresaDAO

Figura 15. Proyecto de la aplicación web SIS_HC, funcionalidad registro de empresa

VI. Pruebas

En esta sección se probará el enfoque propuesto, así, se ha considerado un indicador: *tiempo de generación de código* (Thuma, 2012), porque permite medir el tiempo que demora la generación códigos después de haber concluido con el diagrama de clases.

Las pruebas consistieron en desarrollar 40 funcionalidades de la aplicación web SIS_HC para medir el tiempo que demora la generación de los códigos. Se asignaron dos funcionalidades a cinco ingenieros de sistemas de la especialidad de desarrollo de software, quienes se encargaron de crear el CRUD (Create, Read, Update y Delele) para cada funcionalidad. La tabla 5 muestra la relación de las funcionalidades asignadas a los ingenieros. Se utilizó la técnica guía de observación para la recolección de datos, porque permite observar el tiempo que demora el proceso de la generación de códigos, así, se realizaron las pruebas con un tiempo de inicio y tiempo final, donde no se consideró el tiempo de la elaboración del diagrama de clase.

Resultados

En esta sección también se analiza los datos obtenidos de la recolección de datos. La tabla 6 muestra la recolección de datos obtenidos del tiempo de la generación de códigos sin el enfoque propuesto. Los datos obtenidos corresponden al tiempo que tardaron los ingenieros en generar el CRUD para cada funcionalidad.

La tabla 7 muestra la recolección de datos obtenidos del tiempo de la generación de códigos con el enfoque que se propone. Los datos obtenidos corresponden al tiempo que demoró la herramienta en generar el CRUD para cada funcionalidad.

La tabla 8 muestra el tiempo promedio de la generación códigos sin el enfoque propuesto (TPs) y el tiempo promedio con el enfoque (TPc), donde el TPs es de 1693 seg., y el TPc es de 40 seg. También se muestra la diferencia entre ambos tiempos, siendo el tiempo promedio reducido (TPr), el cual es de 1653 seg.

El gráfico 1 muestra el porcentaje del tiempo reducido (TPr), el cual es de 98 %, por lo que se concluye que el enfoque propuesto ha permitido reducir el tiempo de generación de código respecto a la forma manual en que se generan códigos.

VII. Conclusiones

En este trabajo se ha presentado un enfoque que permite completar el ciclo de la generación de códigos desde el diagrama de clases del UML hasta la generación de códigos para las aplicaciones web, donde en la fase final se destaca lo propuesto, habiendo definido antes un meta-modelo que tiene el formalismo del MDC y la programación PWMDC.

Se ha presentado un método y una herramienta para la Generación de Automática de Código –GCA , donde se extrae las líneas de código XML para ser comparadas entre los archivos del diagrama de clases y del meta-modelo, por medio de algoritmos, para que finalmente generan códigos de programación web basados en la PWMDC. Se ha definido tres fases: fase inicio, fase de análisis y fase de generación, en cada fase se ha desarrollado algoritmos que permite iniciar, leer, analizar y generar código para las aplicaciones web.

Se validó el enfoque propuesto utilizando una aplicación web concluida de nombre SIS_HC, con el objetivo de generar códigos. En el primer paso se desarrolló un meta-modelo que contiene las especificaciones formales de la MDC. Posteriormente, se utilizó Eclipse Luna-Papyrus para elaborar el meta-modelo, donde se usaron los stereotypes del meta-modelo para ser aplicados en el diagrama clases. Para culminar, se desarrolló una herramienta que tienen los algoritmos. Debido a que las actuales herramientas generan código incompleto, la herramienta permite seleccionar el diagrama de clases y genera los códigos en forma automática. En este artículo se presenta los códigos generados.

En la generación de los códigos se usó Java como lenguaje de programación y Java Server Face para la elaboración de las páginas web.

Finalmente, se realizaron las pruebas al enfoque propuesto. Gracias al indicador tiempo de generación de códigos, se presentó los resultados obtenidos del tiempo que tardaron los ingenieros de sistemas y la herramienta en la generación de códigos.

A partir de estos resultados, se puede concluir que el enfoque propuesto permitió reducir el tiempo de generación de código hasta en un 98 %.

Tabla 5. Funcionalidades de la aplicación web SIS_HC

ítem	Funcionalidades del SIS_HC	Métodos			
		Create	Read	Update	Delete
1	Filiación de Trabajador	x	x	x	x
2	Puesto de Trabajo	x	x	x	x
3	Ficha médica evaluación	x	x	x	x
4	Ficha médica conclusiones	x	x	x	x
5	Triaje médico	x	x	x	x
6	Antecedentes médicos	x	x	x	x
7	Ficha Radiológica	x	x	x	x
8	Cita - Laboratorio	x	x	x	x
9	Resultados - Laboratorio	x	x	x	x
10	Atención de Pacientes	x	x	x	x
	muestra (n)	40			

Tabla 6. Tiempo de la generación de código sin el enfoque

ítem	Funcionalidades del SIS_HC	Tiempo generación de código sin el enfoque											*TT seg.
		Capa Negocio					Capa Entidad	Capa Datos					
		C	R	U	D	Tiempo		C	R	U	D	Tiempo	
1	Filiación del Trabajador	x	x	x	x	300	669	x	x	x	x	664	1633
2	Puesto de Trabajo	x	x	x	x	321	520	x	x	x	x	702	1544
3	Ficha médica	x	x	x	x	289	672	x	x	x	x	672	1633
4	Ficha médica conclusiones	x	x	x	x	295	679	x	x	x	x	865	1839
5	Triaje médico	x	x	x	x	279	665	x	x	x	x	708	1653
6	Antecedentes Médicos	x	x	x	x	259	732	x	x	x	x	782	1774
7	Ficha Radiológica	x	x	x	x	355	637	x	x	x	x	869	1861
8	Cita	x	x	x	x	310	570	x	x	x	x	801	1682
9	Resultados Lab.	x	x	x	x	274	721	x	x	x	x	805	1800
10	Atención de Pacientes	x	x	x	x	317	490	x	x	x	x	703	1510

Tabla 7. Tiempo de la generación de código con el enfoque

ítem	Funcionalidades del SIS_HC	Tiempo de generación de código con el enfoque									*TT seg.
		Capa Negocio				Capa Entidad	Capa Datos				
		C	R	U	D		C	R	U	D	
1	Filiación del Trabajador	x	x	x	x	x	x	x	x	x	42
2	Puesto de Trabajo	x	x	x	x	x	x	x	x	x	46
3	Ficha médica	x	x	x	x	x	x	x	x	x	40
4	Ficha médica conclusiones	x	x	x	x	x	x	x	x	x	47
5	Triaje médico	x	x	x	x	x	x	x	x	x	46
6	Antecedentes Médicos	x	x	x	x	x	x	x	x	x	40
7	Ficha Radiológica	x	x	x	x	x	x	x	x	x	35
8	Cita	x	x	x	x	x	x	x	x	x	37
9	Resultados Lab.	x	x	x	x	x	x	x	x	x	33
10	Atención de Pacientes	x	x	x	x	x	x	x	x	x	32

* TT: Tiempo Total

Tabla 8. Tiempo Promedio de generación de código

ítem	Funcionalidades del SIS_HC	SIN ENFOQUE	CON ENFOQUE	2TPr(seg.)
		TPs (seg.)	TPc (seg.)	
1	Filiación del Trabajador	1633	42	1591
2	Puesto de Trabajo	1544	46	1498
3	Ficha médica	1633	40	1593
4	Ficha médica conclusiones	1839	47	1792
5	Triaje médico	1653	46	1607
6	Antecedentes Médicos	1774	40	1734
7	Ficha Radiológica	1861	35	1826
8	Cita	1682	37	1645
9	Resultados Lab.	1800	33	1767
10	Atención de Pacientes	1510	32	1478
TP = Tiempo Promedio		1693	40	1653

* **TPr** TPs – TPc
TPs = Tiempo promedio sin el enfoque de generación de código
TPc = Tiempo promedio con el enfoque de generación de código
TPr = Tiempo promedio reducido

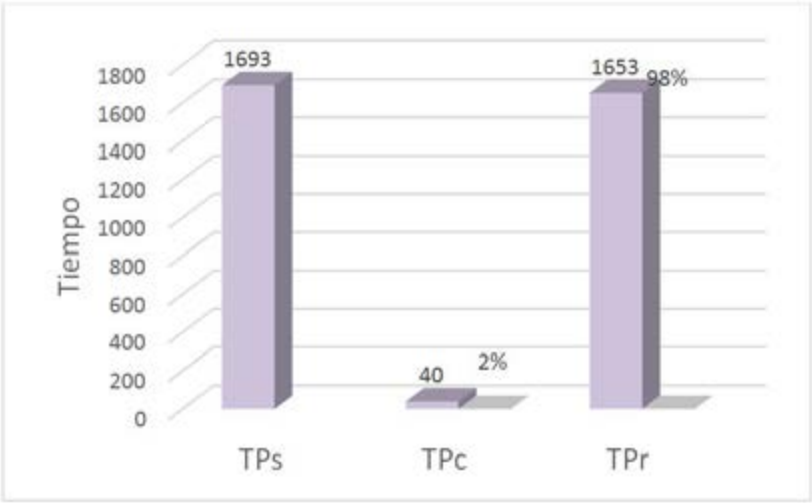


Gráfico 1. Tiempo Promedio de la generación de código
TPs = Tiempo promedio sin el método de generación de código
TPc = Tiempo promedio con el método de generación de código
TPr = Tiempo promedio reducido

Referencias

Almuallim, H. (1996). An efficient algorithm for optimal pruning of decision trees. *Artif. Intell*, 347-362.

Basso, F. P. (2016). Automated design of multi-layered web information systems. *Journal of Systems and Software, vol. 117*, 612-637.

Bennett, J. (2010). Aspect-oriented model-driven skeleton code generation: A graph-based transformation approach. *Science of Computer Programming 75*, 689_725.

Clemente, P. (2011). Managing crosscutting concerns in component based systems using a model driven development approach,. *The Journal of Systems and Software 84* , 1032–1053.

Domínguez, B. P. (2012). A systematic review of code generation proposals from state machine specifications. *Information and Software Technology 54*, 1045–1066.

Eclipse Luna. (21 de Abril de 2018). Obtenido de www.eclipse.org/downloads/packages/eclipse-modeling-tools/lunasr2

Garrigós, I. (2010). Specification of personalization in web application design. *nformation and Software Technology*, 991–1010.

Hat, R. (30 de 09 de 2019). *http://www.hibernate.org/*.

IBM. (20 de Agosto de 2018). *IBM Rational Rose*. Obtenido de http://www-01.ibm.com/software/awdtools/ developer/rose/

JBOSS. (23 de 09 de 2019). *http://richfaces.jboss.org*.

Magdalenic, I. (2012). Autogenerator: Generation and execution of programming code on demand. *Expert Systems with Applications*, 2845–2857.

Mamas, E. (2000). Towards PortableSource Code Representation Using XML. *7th WCRE'2000*.

Manoli, A. (2011). Generating operation specifications from UML class diagrams: A model transformation approach. *Data & Knowledge Engineering, 70*, 365–389.

Mehmood, A. (2013). Aspect-oriented model-driven code generation: A systematic mapping study. *Information and Software Technology vol. 55*, 395–411.

Microsoft. (23 de 09 de 2019). *https://visualstudio.microsoft.com*.

Microsystems, S. (04 de Febrero de 2016). *JSF*. Obtenido de https://javaee.github.io/javaserverfaces-spec/

MKLab. (29 de 05 de 2019). *http://staruml.io/*.

Papyrus. (30 de Octubre de 2018). Obtenido de http://www.papyrusuml.org/

PHP. (30 de 06 de 2018). Obtenido de PHP: www.php.net

Pinto, M. (2012). Deriving detailed design models from an aspect-oriented ADL using MDD". *The Journal of Systems and Software, 85*, 525– 545.

Piraquive, R. G. (2015). Analysis and Improvement of the Management. *IEEE Latin America Transactions*.

PrimeFaces. (23 de 09 de 2019). *https://www.primefaces.org/*.

Resig, J. (29 de 05 de 2019). *https://jquery.com*.

Rincón, M. (2011). Generación Automática de Código a Partir de Máquinas de Estado Finito. *Computación y Sistemas Vol. 14 No. 4*, 405-421.

Ruiz Catalan, J. (2010). *Compiladores Teroria y Implmentación.* Madrid: Grupo RC.

Selic, B. (2003). The pragmatics of model-driven development. *IEEE Software 20 (5)*, 19–25.

Thuma, T. (2012). FeatureIDE: An Extensible Framework for. *Science of Computer Programming*.

Tigris.org. (25 de Setiembre de 2018). *Open Source Software Engineering Tools*. Obtenido de ArgoUML Modeling Tool: http://argouml.tigris.org

Vega Fajardo, A. (30 de Octubre de 2018). *Generación Codigo*. Obtenido de www.speeddatasoftware.com/speed/gca.html