Revista Peruana de Computación y Sistemas 2019 2(2):25-42
DOI: http://dx.doi.org/10.15381/rpcs.v2i2.17015

ISSN Online: 2617-2003
Facultad de Ingeniería de Sistemas e Informática
Universidad Nacional Mayor de San Marcos

Artículo de Contribución

Método basado en la programación por capas para generar código automático desde el diagrama de clases

Method based on layered programming to generate code automatic from the class diagram

Adolfo Hans Vega Fajardo
Universidad Nacional de Ingeniería. Lima, Perú
Email: adolfovega71@hotmail.com

Resumen

El mayor esfuerzo en el desarrollo de aplicaciones web se centra en la generación de código de forma manual basado en un lenguaje de programación. Por otro lado, los trabajos de investigación se enfocan en resolver problemas de diseño, mientras que las herramientas CASE generan códigos limitados o incompletos sin las especificaciones formales para el desarrollo de aplicaciones web. En este artículo se propone un método basado en la programación por capas para generar código de manera automática desde el diagrama de clases del UML para aplicaciones web. De este modo, se analiza el archivo del diagrama de clases y se genera el código respectivo. Se ha definido un modelo de diseño como meta-modelo, el cual tiene el formalismo de la programación 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 en forma automática. Para validar la propuesta, se utilizó una aplicación concluida con el objetivo de comparar entre el proceso de la generación del código automático y la generación de código manual, teniendo como resultado que la generación del código se reduce hasta en un 98 %.

Palabras clave: Técnica de programación; UML; arquitectura de capas; método de generación de códigos.

Abstract

The greatest effort in the development of web applications is focused on the generation of code manually based on a programming language. On the other hand, research works focus on solving design problems, while CASE tools generate limited or incomplete codes without formal specifications for web application development. This article proposes a method based on layered programming to generate code automatically from the UML class diagram for web applications. In this way, the class diagram file is analyzed and the respective code is generated. A design model has been defined as a meta-model, which has the formalism of layered programming and is represented by a profile with XML extension, in this way, the lines of XML code are extracted from the file that contains the diagram of classes, are compared with the meta-model through algorithms and then the code is generated automatically. To validate the proposal, an application completed with the objective of comparing between the process of automatic code generation and manual code generation was used, resulting in the code generation being reduced by up to 98%.

Keywords: Programming Technique; UML; Layer Architecture; Code Generation Method.

Correspondencia:
Dirección: Universidad Nacional Mayor de San Marcos, Facultad de Ingeniería de Sistemas e Informática. Calle Germán Amézaga N° 375, Ciudad Lima 1.

Recibido 13/11/2019 - Aceptado 27/12/2019

Citar como:
Vega, A. (2019) Método basado en la programación por capas para generar código automático desde el diagrama de clases. Revista Peruana de Computación y Sistemas, 2(2):25-42. http://dx.doi.org/10.15381/rpcs.v2i2.17015

1. Introducción

Las aplicaciones web a nivel empresarial tienen ventajas frente a las aplicaciones convencionales [1]. Para el desarrollo de aplicaciones web es necesario tener un conjunto básico de componentes: 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, los cuales deben estar relacionados [2]; 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, del 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, el segundo sirve para la funcionalidad de la aplicación, y el último sirve para acceder a la base de datos.

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 [3].

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 para crear formularios, lista de información y ventanas emergentes; el segundo grupo estado conformado por los softwares de entorno de desarrollado integrado o, por sus siglas en inglés, IDE (Integrated Development Environment), el cual contiene herramientas que generan código básicos, uno de ellos es el NetBean, que tiene como base el lenguaje de programación java y cuenta con opciones que generan código para los métodos Get() y Set() que son utilizados por las clases java; 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 [4], ArgoUML [5] y otros, sin embargo, estos CASE generan código incompleto [3]. La tabla 1 muestra los generadores de código identificados por grupos, 1: El nombre del generador de código, 2: El lenguaje de programación en el que se basa el código generado, 3: La aplicabilidad del código en el desarrollo de las aplicaciones web, y 4: El tipo de generador (Marcos de Trabajo, IDE o CASE).

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 web por capas, la sección 4 muestra la propuesta, en la sección 5 muestra un caso de uso, la sección 6 presenta las pruebas y los resultados del enfoque y, por último, en la sección 7 se exponen las conclusiones.

2. Motivación

La generación de código (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 [13].

El estudio de Piraquive et al. [2] analiza las causas que originan que los proyectos de software fracasen, donde se destaca la administración y la metodología para el desarrollo de software. La figura 1 exhibe las causas que producen el fracaso del proyecto de desarrollo de software.

Tabla 1. *Generadores de código por grupos*

1 Nombre Grupo 1: Páginas web	Referencia	2 Lenguaje de Programación de base	3 Desarrollo de aplicaciones web	4 Tipo de Generador
JQuery	[6]	JavaScript	Formularios, ventanas, mensajes, menús, listados, tablas, etc.	Marco de Trabajo para HTML
PrimeFace	[7]			Marco de Trabajo para Java Server Face (JSF).
RichFace	[8]			
Vaadin Flow	[9]			
Grupo 2: IDE				
Eclipse	[10]	Java, PHP	Métodos, Clases, Propiedades, etc.	IDE que contiene opción y/o herramientas para generar código.
NetBeans	[11]	Java, PHP		
Grupo 3: CASE				
ArgoUML	[5]		Genera código	CASE
IBM Rational Architect	[4]		Genera código	CASE
StartUML	[12]		Genera código	CASE

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 de Domínguez et al. [3], 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 et al. [14] 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 [3], [15], [16], [2] y [17] 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 framework 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 [3], además, las herramientas CASE actuales, tales como IBM Rational Architec [4], Eclipse Luna [10], ArgoUML [5] , Vaadin Flow [9], y otros, generan código incompleto [3] y solo sirven para diseñar o generar código sin el formalismo de metodología de desarrollo de software para aplicaciones web.

Sin embargo, las herramientas CASE han sido usadas por los investigadores para poner en práctica sus trabajos de generación código, como en el estudio de [14] , 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 [18] 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 trabajo de investigación: primero, el mayor esfuerzo en el desarrollo de aplicaciones web no es el diseño, sino 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.

3. 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

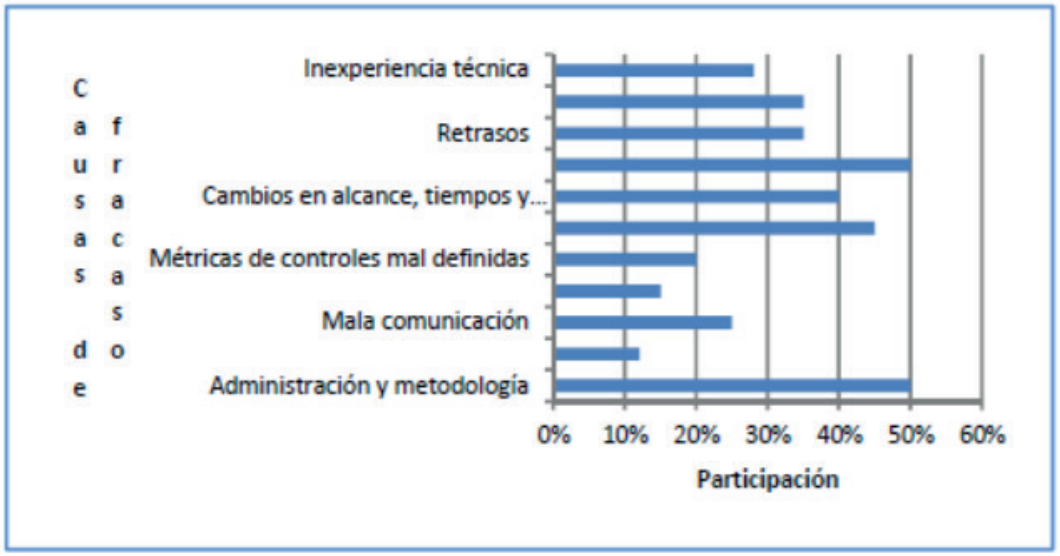


Figura 1. Causas de fracasos en Proyectos de Software **[2]**

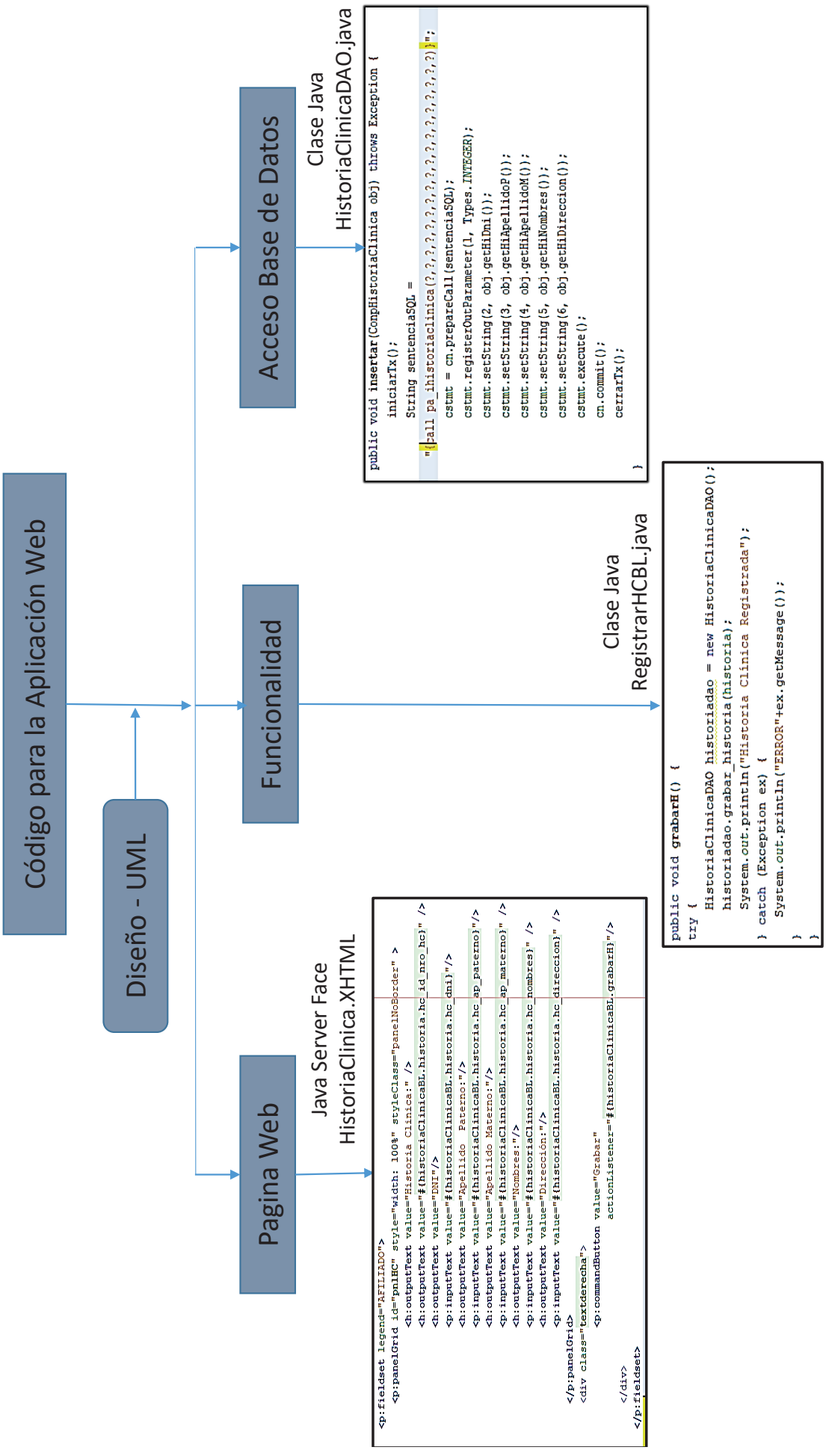


Figura 2. Código para la Pagina Web, Funcionalidad y Acceso de la Base de Datos

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 **[19]**, 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 **[20]** **[21]** en el desarrollo de aplicaciones web, la figura 3 muestra MDC y la figura 4 muestra MVC.

La programación web basada en el modelo de desarrollo por capas (PWMDC) es un método que permite programar en capas **[24]** 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 **[25]**, 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 describen 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, JavaScript 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 **[26]** 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) **[27]** 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

Figura 3. Modelo de desarrollo por Capa **[22]**

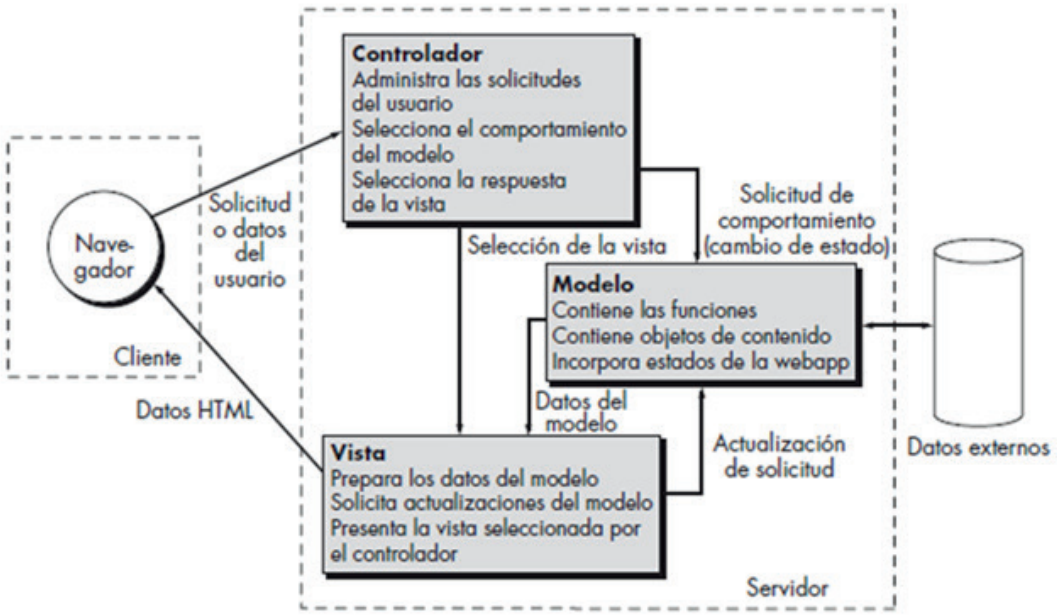


Figura 4. Modelo Vista Controlador **[23]**

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 5 se muestra la estructura de la PWMDC con sus respectivas capas.

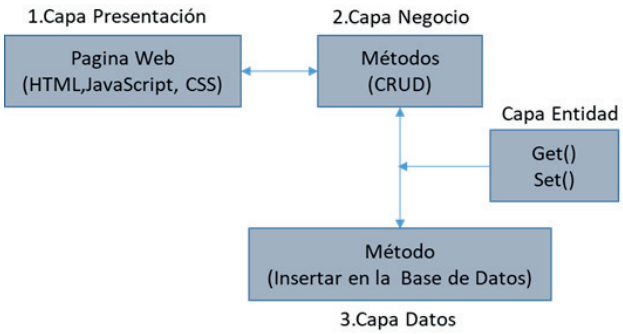


Figura 5. Estructura de la PWMDC

4. Propuesta

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 6 muestra la visión general del enfoque.

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) [20].

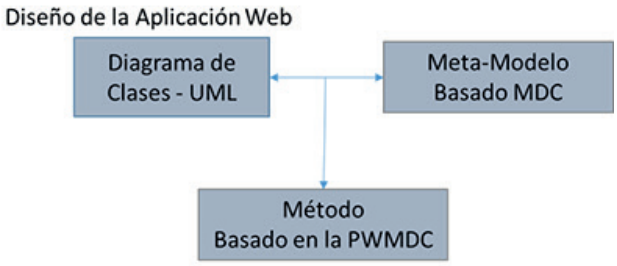


Figura 6. Visión General de la generación de códigos

Con base en el trabajo de investigación de Manoli et al. [14], sobre método basado en reglas de diseño, se han elaborado reglas de diseño para el diagrama de clases que cuentan 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 2 muestra las reglas de diseño para la capa de negocio, por ejemplo, una de las reglas para el diagrama de clases 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 3 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 4 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 **[18]**, se ha creado el meta-modelo para luego ser comparado con el diagrama de clases. La figura 7 muestra el meta-modelo elaborado con los stereotypes basados MDC, el cual puede ser descargado **[28]**.

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 **[29]**, posteriormente, es comparada con un meta-modelo, para que finalmente se genere códigos basado en la PWMDC.

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	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	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

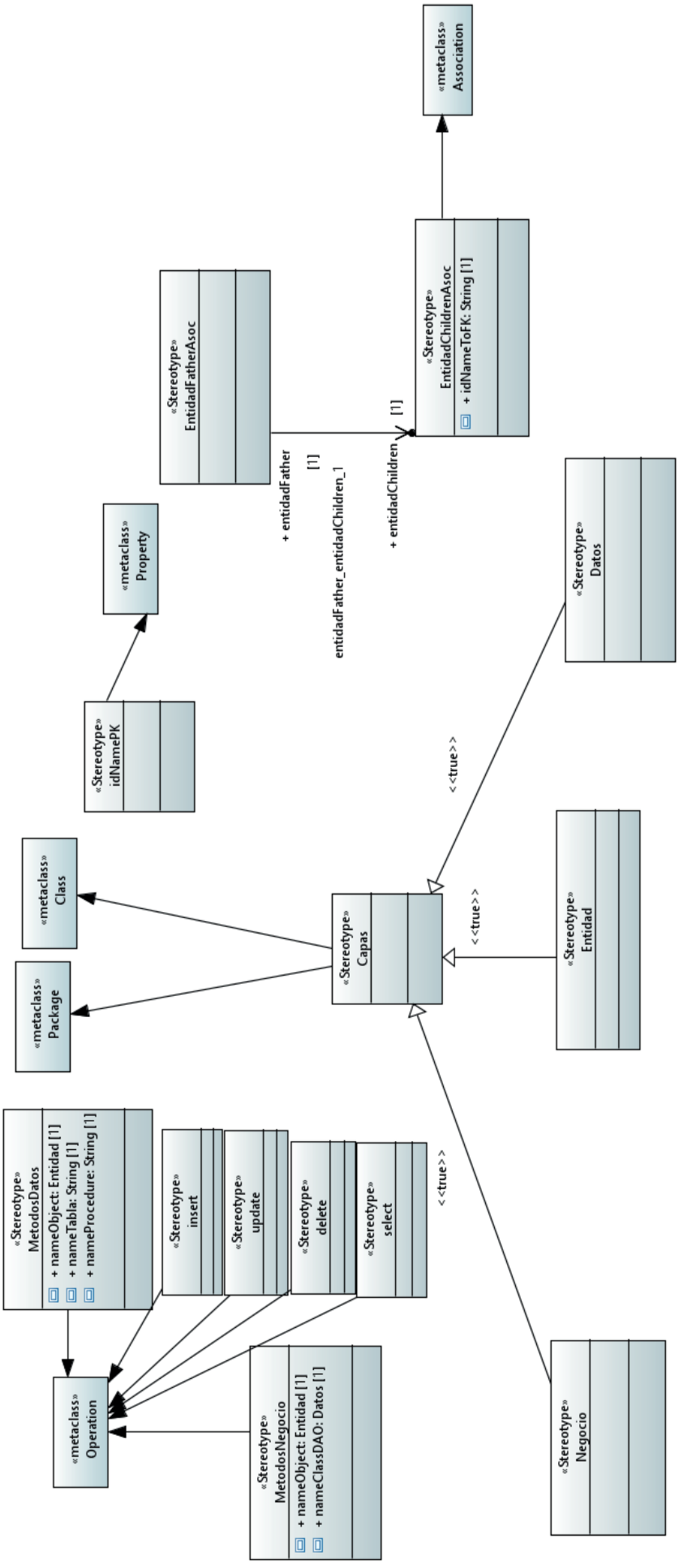


Figura 7. Meta-modelo basado en la MDC

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 8 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 8. 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 9 muestra las fases para método basado en la programación por capas para generar código automático desde el diagrama de clases.

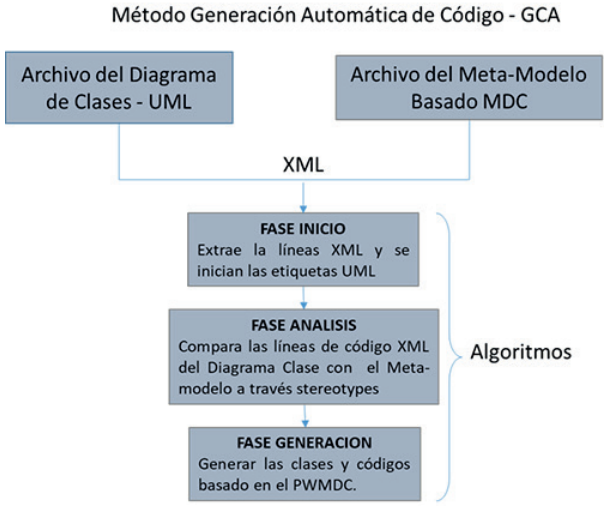


Figura 9. Método de generación basada en la PWMDC

Tabla 5. *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"

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 relacionadas con las etiquetas definidas en la tabla 5, en el caso que cumplan, pasan a la siguiente fase, el resto son rechazadas e informadas. La figura 10 muestra la clase de nombre "iniciarEtiquetas" encargada de iniciar las etiquetas, por su parte, la figura 11 muestra la clase "readStereotype" que contiene el algoritmo.

```
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 10.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 11. 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 7 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 12 muestra el 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 13 muestra el algoritmo de la fase de generación.

5. Caso De Uso

En esta sección se tomó una aplicación web terminada para aplicar nuestra propuesta, donde se utilizó como caso de uso la aplicación de nombre SIS_HC (usada para los servicios de salud de los centros médicos ocupacionales en el Perú), desarrollada con la arquitectura por capas y diseñada con los diagramas del UML, asimismo, se hizo uso de la tecnología Java Server Face

```
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 12. Algoritmo de la fase de análisis

```
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 13. Algoritmo de la fase generación

para las páginas web y el lenguaje de Programación Java para el código. Como ejemplo se utilizó una funcionalidad "registro de empresa" del SIS_HC para aplicar nuestra propuesta. Con ayuda del CASE Eclipse, se explica los pasos a seguir:

Primer Paso: Consiste en crear un proyecto nuevo de nombre SIS_HC (ver figura 14 (a)), se selecciona el tipo de lenguaje UML y los diagramas UML (ver figuras 14 (b) y 14 (c)), y, por último, se aplica el meta-modelo (definido en el sección 4) al proyecto SIS_HC. La figura 15 muestra la aplicación del meta-modelo, donde se selecciona el archivo profile que tiene el meta-modelo y se agrega al proyecto.

Segundo Paso: elaboración el diagrama de clases para la funcionalidad registro de empresa, 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 16(a) muestra el diagrama de clases para la capa de entidad con sus respectivos stereotypes, La figura 16(b) muestra el diagrama de clase de la capa negocio y la figura 16(c) muestra el diagrama de clase de la capa datos. Los diagramas de clases se han elaborado siguiendo las reglas de diseños descritas en las tablas 2,3 y 4.

Tercer Paso: En esta parte se ha elaborado una aplicación de nombre "Generación de Código Automático" (GCA) para 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 17 muestra la aplicación, donde se selecciona el diagrama clases de la aplicación web SIS_HC y se generan los códigos basados en PWMDC. La figura 18(a) presenta el código generado para la clase de nombre "EmpresaBL" del paquete

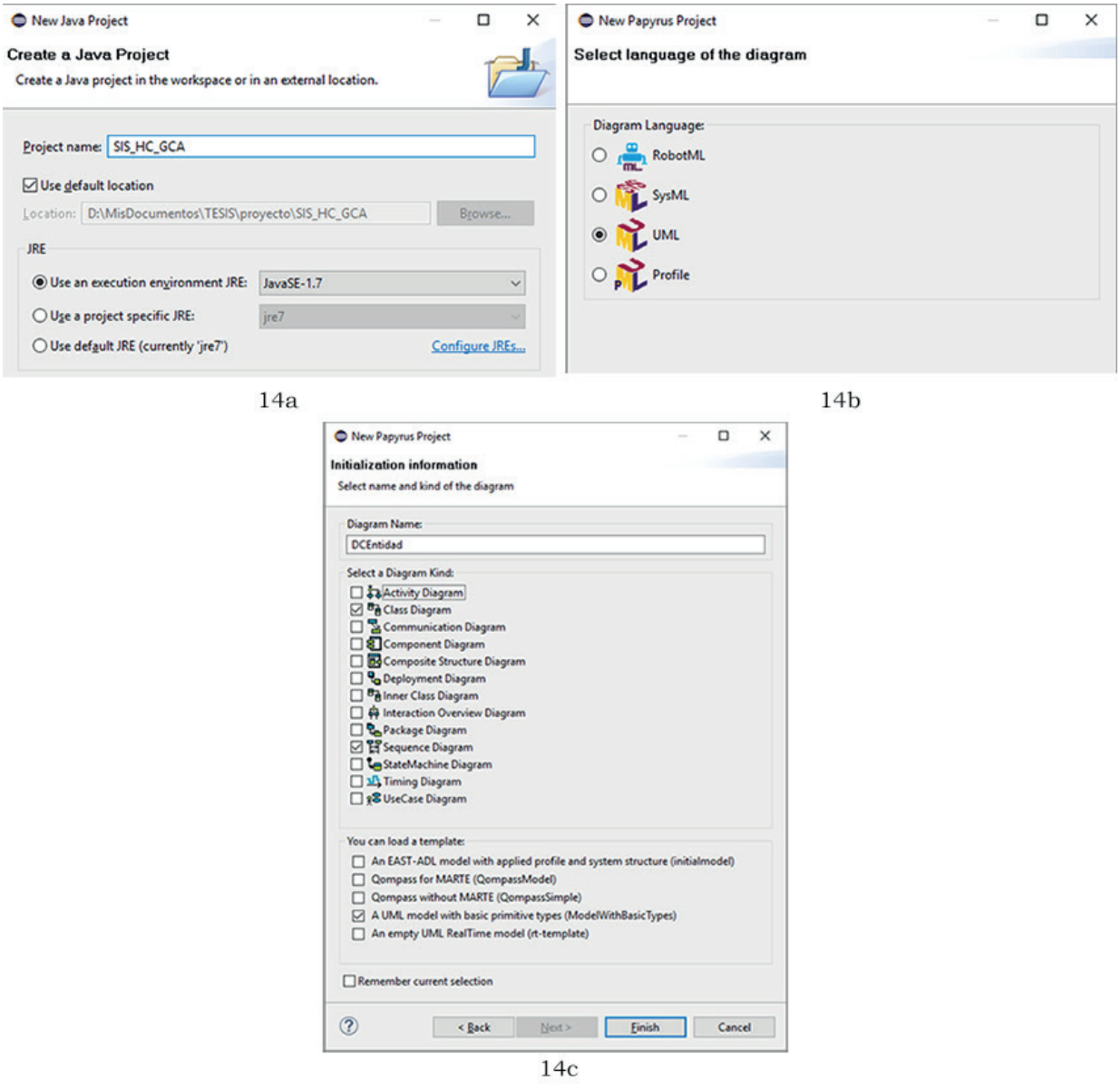


Figura 14. Pasos con el Eclipse: 14a) creación del nuevo proyecto, 14b) seleccionar el lenguaje diagrama, 14c) seleccionar el diagrama

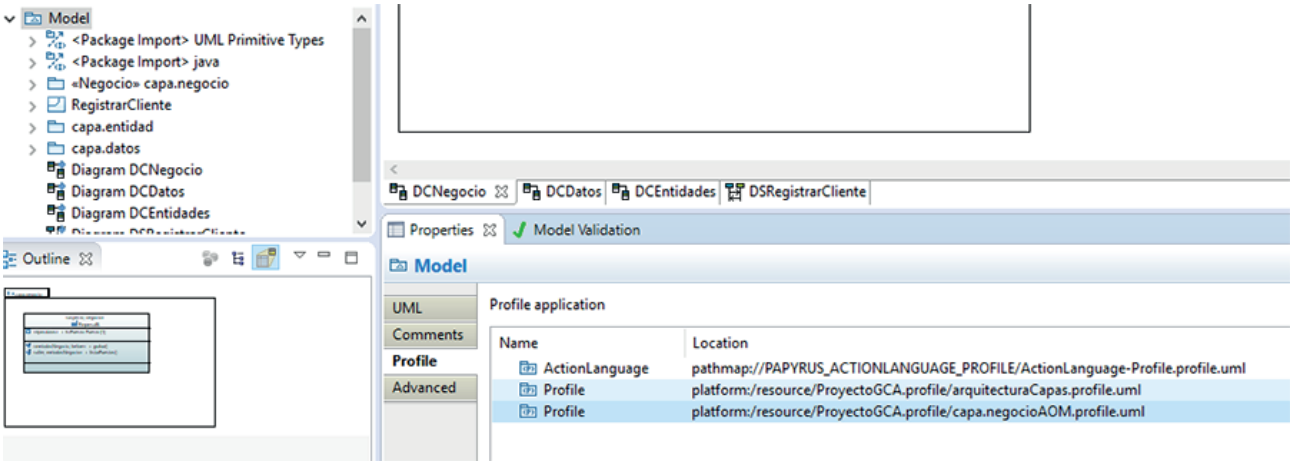
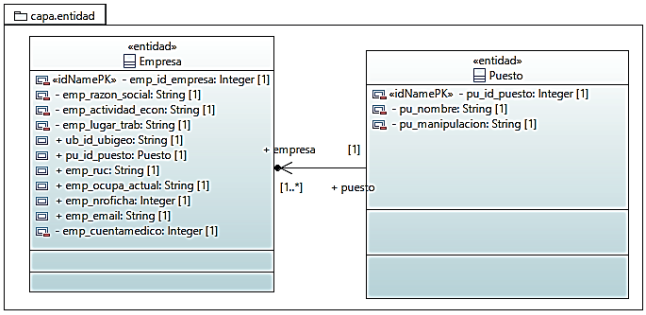
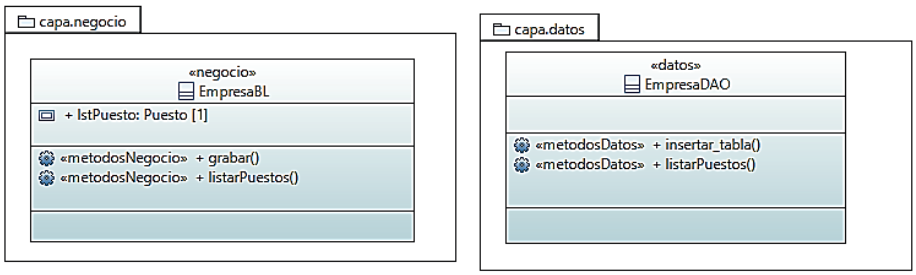


Figura 15. Aplicación del meta-modelo con el Eclipse



16 a



16 b 16 c

Figura 16. Diagramas de clases (DC): 16a) DC de la Capa de Entidad, 16b) DC de la Capa de Negocio, 16c) DC de la capa de datos

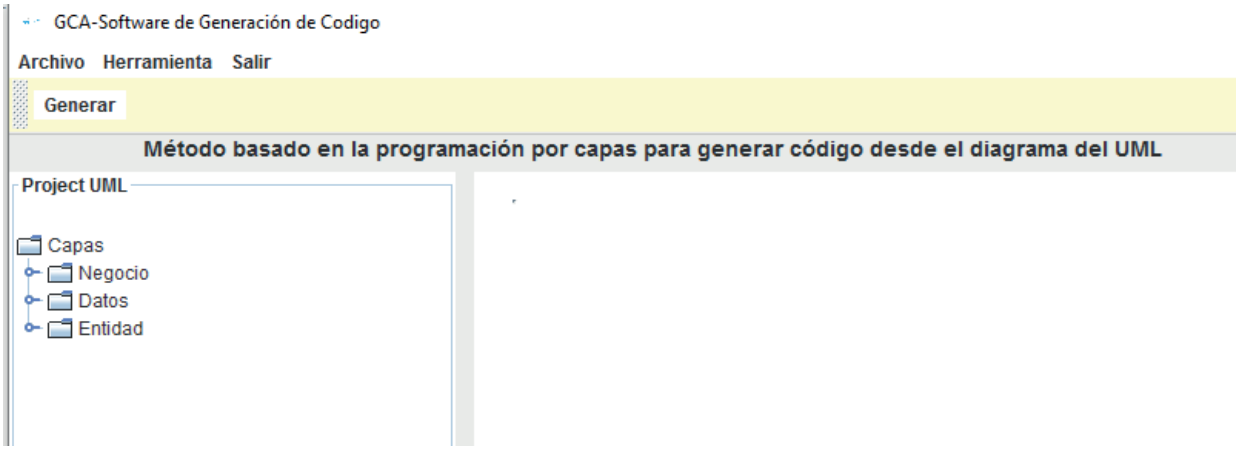


Figura 17. Aplicación Generación de Código Automático con el Método Propuesto

de negocio, esta clase representa la funcionalidad (registro de empresa) de la aplicación web, por otro lado, la figura 18(b) hace notar el código de la clase entidad de nombre "Empresa" representa la tabla empresa de la base de datos, por último, la figura 18(c) muestra el código para la clase de datos de nombre "EmpresaDAO" representa la subclase creada para insertar los datos a la tabla empresa.

Cuarto Paso: las clases son incluidas en el proyecto de la aplicación web SIS_HC. La figura 19 presenta la interface gráfica (página web) terminada, donde se usa las clases generadas en el proyecto que utilice el Java Server Face para el diseño de páginas, el lenguaje de programación Java para la funcionalidad y acceso a la base de datos.

6. Pruebas

En esta sección se probará la propuesta, así, se ha considerado un indicador: *tiempo de generación de código* [30], porque permite medir el tiempo que demora la generación códigos después de haber concluido con el diagrama de clases.

```java
package capa.negocio;

import java.util.ArrayList;
import java.util.List;
import capa.datos.EmpresaDAO;
import capa.entidad.Empresa;
import capa.entidad.Puesto;

 public class EmpresaBL {

        private Empresa empresa = new Empresa();
        private List<Puesto> lst_puesto = new ArrayList<Puesto>();
        private EmpresaDAO empresadao= new EmpresaDAO();

        public Empresa getEmpresa() {
            return empresa;
        }
        public void setEmpresa(Empresa empresa) {
            this.empresa = empresa;
        }
        public List<Puesto> getLst_puesto() {
            return lst_puesto;
        }
        public void setLst_puesto(List<Puesto> lst_puesto) {
            this.lst_puesto = lst_puesto;
        }



        public void grabar(){
              empresadao.grabar(empresa);
        }

```

18a

```java
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;
    }

```

18b

```java
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 rs1=null;
        static  CallableStatement cstmt = null;
        static  CallableStatement cstmt1 = 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();
    }
```

18c

Figura 18. Códigos generados: 18a) Clase negocio Registrar Empresa, 18b) Clase Entidad Empresa, 18c) Clase Datos EmpresaDAO.

Figura 19. Proyecto de la aplicación web SIS_HC, funcionalidad registro de empresa

Primero, las pruebas consistieron en seleccionar funcionalidades de una aplicación web terminada "SIS_HC" para que después las mismas funcionalidades sean desarrolladas con nuestra propuesta y finalmente sean comparadas, seleccionamos una funcionalidad por cada módulo de la aplicación SIS_HC, siendo un total de 10 funcionales para medir el tiempo que demora la generación de los códigos. Luego, se asignaron dos funcionalidades a cinco ingenieros de sistemas de la especialidad de desarrollo de software para crear el CRUD a cada funcionalidad. La tabla 6 muestra la relación de las funcionalidades asignadas a los ingenieros. Por último, 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 7 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 8 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 9 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.

La figura 20 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.

7. Conclusiones

En este trabajo se ha presentado un método basado en la programación por capas para generar código automático desde el diagrama clase, 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 tomó una aplicación web terminada para aplicar nuestra propuesta y se utilizó como caso de uso la aplicación de nombre SIS_HC (usada para los servicios de salud de los centros médicos ocupacionales en el Perú), desarrollada con la arquitectura por capas y diseñada con los diagramas del UML, así mismo, se hizo uso de la tecnología Java Server Face para las páginas web y lenguaje de programación java. Como 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.

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.

Finalmente, el presente trabajo de investigación beneficiará a los profesionales de ingeniería de software, quienes contarán con un generador automático de código para reducir el tiempo de desarrollo de una aplicación web.

También beneficiará a las empresas por que contarán con un generador de código que facilitará la construcción o elaboración de las aplicaciones web en menos tiempo o permitirá estandarizar y mejorar el criterio lógico de programación.

Tabla 6. *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 7. *Tiempo de la generación de código sin el enfoque*

ítem	Funcionalidades del SIS_HC	Tiempo generación de código sin el enfoque											[1]TTs
		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 8. *Tiempo de la generación de código con el enfoque*

ítem	Funcionalidades del SIS_HC	Tiempo de generación de código con el enfoque									*TTs
		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 9. *Tiempo Promedio de generación de código*

ítem	Funcionalidades del SIS_HC	SIN ENFOQUE	CON ENFOQUE	*TPr(s)
		TPs (s)	TPc (s)	
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

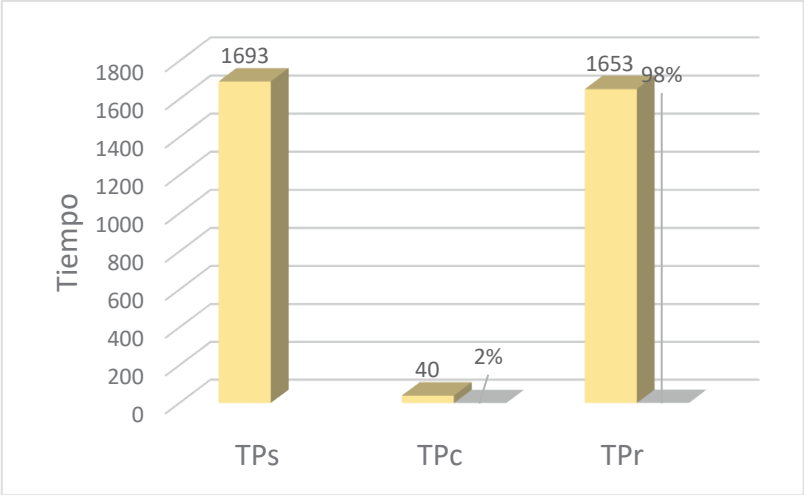


Figura 20. Tiempo Promedio de la generación de código.

TPs = Tiempo promedio sin el método de generación de código
TPc = Tiempo promedio con el método de generación de código
TPr = Tiempo promedio reducido

8. Referencias

[1] E. F. Mejía Peñafiel, «La ingeniería de requisitos una base fundamental para el desarrollo de proyectos de ti en la web,» *Revista Científica y Tecnológica UPSE,* vol. III, nº 1, pp. 71-78, 2015.

[2] R. G. C. a. V. H. M. G. Piraquive, «Analysis and Improvement of the Management,» *IEEE Latin America Transactions,* 2015.

[3] B. P. L. A. a. M. Z. Domínguez, «A systematic review of code generation proposals from state machine specifications,» *Information and Software Technology 54,* p. 1045–1066, 2012.

[4] IBM, «IBM Rational Rose,» 20 Agosto 2018. [En línea]. Available: http://www-01.ibm.com/software/awdtools/ developer/rose/.

[5] Tigris.org, «Open Source Software Engineering Tools,» 25 Setiembre 2018. [En línea]. Available: http://argouml.tigris.org.

[6] J. Resig, «https://jquery.com,» 29 05 2019. [En línea].

[7] PrimeFaces, «https://www.primefaces.org/,» 23 09 2019. [En línea].

[8] JBOSS, «http://richfaces.jboss.org,» 23 09 2019. [En línea].

[9] W. Michael, «Vaadin,» 05 11 2019. [En línea]. Available: https://vaadin.com/. [Último acceso: 2019 12 23].

[10] «Eclipse Luna,» 21 Abril 2018. [En línea]. Available: www.eclipse.org/downloads/packages/eclipse-modeling-tools/lunasr2.

[11] A. S. Foundation, Oracle Corporation y Sun Microsystems, «NetBeans,» 1 12 2019. [En línea]. Available: https://netbeans.org/. [Último acceso: 23 12 2019].

[12] MKLab, «http://staruml.io/,» 29 05 2019. [En línea].

[13] C. P. J. G. C. F. P. L. J. M. F. M. J. C. M. Alejandro, Fundamentos de programación, Madrid: Thomson, 2007.

[14] A. Manoli, «Generating operation specifications from UML class diagrams: A model transformation approach,» *Data & Knowledge Engineering, 70,* p. 365–389, 2011.

[15] J. Bennett, «Aspect-oriented model-driven skeleton code generation: A graph-based transformation approach,» *Science of Computer Programming 75,* p. 689_725, 2010.

[16] M. Pinto, «Deriving detailed design models from an aspect-oriented ADL using MDD",» *The Journal of Systems and Software, 85,* p. 525– 545, 2012.

[17] M. Rincón, «Generación Automática de Código a Partir de Máquinas de Estado Finito,» *Computación y Sistemas Vol. 14 No. 4,* pp. 405-421, 2011.

[18] «Papyrus,» 30 Octubre 2018. [En línea]. Available: http://www.papyrusuml.org/.

[19] P. Clemente, « Managing crosscutting concerns in component based systems using a model driven development approach,,» *The Journal of Systems and Software 84 ,* p. 1032–1053, 2011.

[20] F. P. Basso, «Automated design of multi-layered web information systems,» *Journal of Systems and Software, vol. 117,* pp. 612-637, 2016.

[21] I. Garrigós, «Specification of personalization in web application design,» *nformation and Software Technology,* p. 991–1010, 2010.

[22] F. A. Vasquez, «Programación por Capas,» *Revista del Departamento Académica Universidad Ricado Palma,* vol. 1, pp. 13-30, 2010.

[23] R. S. Pressman, Ingenierıa de software: un enfoque practico. 7ta edicion, ISBN 970–10–5473–3, Mexico: INTERAMERICANA, 2010.

[24] Santiago Domingo Moquillaza Henríquez, Hugo Vega Huerta, «Programación en N capas,» *Revista de Investigación de Sistemas e Informática,* vol. 7, nº 2, pp. 57-67, 2010.

[25] R. Hat, «http://www.hibernate.org/,» 30 09 2019. [En línea].

[26] S. Microsystems, «JSF,» 04 Febrero 2016. [En línea]. Available: https://javaee.github.io/javaserverfaces-spec/.

[27] «PHP,» 30 06 2018. [En línea]. Available: www.php.net.

[28] A. Vega Fajardo, «Generación Codigo,» 30 Octubre 2018. [En línea]. Available: www.speeddatasoftware.com/speed/gca.html.

[29] E. Mamas, «Towards PortableSource Code Representation Using XML,» *7th WCRE'2000,* 2000.

[30] T. Thuma, «FeatureIDE: An Extensible Framework for,» *Science of Computer Programming,* 2012.