ESCUELA SUPERIOR POLITÉCNICA DE CHIMBORAZO

FACULTAD DE INFORMÁTICA Y ELECTRÓNICA

ESCUELA DE INGENIERÍA EN SISTEMAS

TEMA:

"ANÁLISIS DEL PATRÓN MODELO VISTA CONTROLADOR IMPLEMENTADO EN LENGUAJES DE SOFTWARE LIBRE PARA EL DESARROLLO DE APLICACIONES WEB. CASO PRÁCTICO: LICEO DE TALENTOS STEPHEN HAWKING"

TESIS DE GRADO

Previa obtención del título de:

INGENIERO EN SISTEMAS INFORMÁTICOS

PRESENTADO POR:

Víctor Miguel Fuertes Ortega

Juan Carlos Guevara Morocho

RIOBAMBA – ECUADOR

2010

DEDICATORIA

A Dios, a mis padres Mónica y Marcelino, a mis hermanas Pamela y Valeria, y a quienes pusieron su confianza, esperanza y corazón para ver en un momento alcanzar un anhelo que solo su paciencia dulce e interminable puede esperar.

A las personas que me incentivaron a realizar este tema de investigación y me facilitaron los recursos necesarios para culminarlo en una forma adecuada.

Miguel

A Dios, a mis papis Rosa y Carlos, a mis hermanas Diana, Andrea y Galilea por su apoyo constante y cariño que sólo ellos me pueden otorgar, a mis compañeros con los que he compartido momentos de alegría y preocupación. Y a todas las personas que de una u otra manera han permitido que este trabajo se cristalice.

Juan Carlos

AGRADECIMIENTO

Formulamos un agradecimiento muy sincero al Liceo de Talentos “Stephen Hawking” y su directiva por la colaboración prestada para la realización de este trabajo, además a nuestra directora de tesis y asesor por brindarnos su ayuda en todo este proceso.

FIRMAS RESPONSABLES Y NOTA

NOMBRE	FIRMA	FECHA
Ing. Iván Menes **DECANO FACULTAD INFORMÁTICA Y ELECTRÓNICA**	……	……
Ing. Raúl Rosero **DIRECTOR ESCUELA INGENIERÍA EN SISTEMAS**	……	……
Ing. Lorena Aguirre **DIRECTOR DE TESIS**	……	……
Ing. Raúl Rosero **MIEMBRO DEL TRIBUNAL**	……	……
Tlgo. Carlos Rodríguez **DIRECTOR CENTRO DE DOCUMENTACIÓN**	……	……
NOTA DE LA TESIS:	……	

RESPONSABILIDAD DE LOS AUTORES

"Nosotros Víctor Miguel Fuertes Ortega y Juan Carlos Guevara Morocho, somos responsables de las ideas, doctrinas y resultados expuestos en esta Tesis de Grado, y el patrimonio intelectual de la misma pertenece a la Escuela Superior Politécnica de Chimborazo"

FIRMAS:

---------- ----------

Víctor Miguel Fuertes Ortega Juan Carlos Guevara Morocho

CAPÍTULO I

MARCO DE REFERENCIA

1.1. Título de la Investigación

Análisis del patrón Modelo Vista Controlador implementado en lenguajes de software libre para el desarrollo de aplicaciones web. Caso práctico: Liceo de Talentos Stephen Hawking.

1.2. Problema de la Investigación

En la actualidad, el desarrollo de sistemas web es un proceso que implica mucho esfuerzo por parte de los programadores debido a que cada vez que se va a implementar un nuevo proyecto se debe se debe crear una estructura a seguir, la cual muchas de las veces no está bien definida, y seguir un formato que no es flexible. Esto implica un diseño con errores y que requiere un mayor tiempo de desarrollo. El patrón de diseño Modelo Vista Controlador sugiere un nuevo enfoque para la creación de sistemas web, basados en la reutilización de código,

flexibilidad en la gestión con bases de datos y ordenamiento de la estructura del los procesos.

1.2.1. Análisis

Para la creación ágil y ordenada de un portal web se realizará un análisis profundo del patrón MVC, para ello se estudiarán sus diferentes partes como:

- El Modelo y la gestión del mismo con los diferentes lenguajes de software libre.
- Vistas y su aplicación.
- El Controlador y su administración con las herramientas libres.

Además, se estudiará las herramientas que ofrece Ruby on Rails (RoR) y Symfony de PHP, dando a conocer su arquitectura, funciones innovadoras, uso, ventajas y desventajas siendo éste un análisis real ya que se pondrá en práctica en el desarrollo de un de aplicación web para el cobro de rubros para el Liceo de Talentos Stephen Hawking.

Las ventajas más sobresalientes de usar frameworks basados en el paradigma de diseño MVC son:

- *Menos software* quiere decir que se escriben menos líneas de código para implementar la aplicación. Si el código es pequeño quiere decir que el desarrollo es más rápido y con menos errores, lo que hará que el código sea fácil de entender, mantener y mejorar.

1.2.2. Delimitación

Lugar de Aplicación

Liceo de Talentos Stephen Hawking

Alcance

Este tema de tesis se enmarca en el análisis del patrón de diseño MVC para el desarrollo de aplicaciones web enfocado en lenguajes de software libre, su estructura, funcionalidades, facilidades que brinda y el uso de éste para desarrollar un ***sistema de cobro de rubros***. Se realizará un estudio para determinar la productividad en el desarrollo de aplicaciones web de dos lenguajes de software libre mediante la creación de una aplicación web basada en el patrón de diseño MVC.

Limitación

La falta de experiencia en lenguajes de software libre puede generar un desfase de las actividades planificadas. El personal del Liceo de Talentos Stephen Hawking deberá recibir capacitación sobre el manejo del software que se creará, ya que actualmente no cuentan con un sistema informático que les ayude en sus funciones.

1.3. Objetivos

1.3.1. Objetivo General

- Analizar el patrón de diseño Modelo Vista Controlador implementado en lenguajes de software libre para el desarrollo de aplicaciones web.

1.3.2. Objetivos Específicos

- Investigar y estudiar el patrón de diseño Modelo Vista Controlador.
- Estudiar y determinar los parámetros de comparación de la productividad en el desarrollo de software.

- Analizar la productividad en el desarrollo de aplicaciones web comparando lenguajes de software libre (Utilizando un framework basado en MVC y sin utilizarlo).
- Implementar un sistema de cobro de rubros en el Liceo de Talentos Stephen Hawking basado en el patrón de diseño Modelo Vista Controlador con un lenguaje de software libre.

1.4. Justificación de la Investigación

1.4.1. Justificación Teórica

En la actualidad un gran porcentaje de instituciones educativas intentan migrar todos sus procesos hacia la web, ya que esto implica un sin número de beneficios como mayor organización de la información, mayor rapidez y reducción de costes y esfuerzos.

Las nuevas tecnologías que actualmente inundan nuestro mercado con variadas y sofisticadas herramientas de desarrollo de software merecen una especial atención debido a que nos ofrecen innovadoras soluciones y un sin número de ventajas, ejemplo de éstas son: Ruby on Rails y symfony de PHP que son frameworks que facilita el desarrollo de aplicaciones web ya que reduce las líneas de código y la cantidad de configuraciones, de la misma manera facilita la reutilización de código, el mantenimiento de la aplicación y las conexiones con los diferentes motores de base de datos.

Un gran número de problemas pueden presentarse cuando las aplicaciones mezclan el código de acceso a datos, el código de la lógica de negocios y el código de presentación. Tales aplicaciones son difíciles de mantener, en razón de que las interdependencias entre todos los componentes causan efectos colaterales cada vez que un cambio se realiza en algún lado. La mezcla de código en proporciones elevadas dificulta o imposibilita la reutilización por causa de la dependencia sobre muchas otras clases. Agregar nuevas vistas de datos frecuente requiere re-implementación o cortar y pegar código de la lógica de negocios, conforme sea necesario en diferentes lugares de la aplicación.

El Patrón de Diseño Modelo-Vista-Controlador resuelve este problema a través de la separación del acceso a datos, la lógica de negocios, la presentación de datos y la interacción del usuario.

Se amerita el estudio del patrón MVC ya que este representa un mecanismo de mejora de procesos de desarrollo de software, fácil de comprender y aplicar, que propende al trabajo en equipo donde se requieren diferentes conjuntos de habilidades para diferentes responsabilidades, además y permite el diseño de aplicaciones de mayor grado de complejidad y facilita el soporte a nuevos tipos de clientes.

1.4.2. Justificación Práctica

El Liceo de talentos Stephen Hawking actualmente efectúa el proceso de cobro de rubros manualmente, esto implica un proceso tedioso y dificultades para gestionar los diferentes pagos como pensiones, transporte, alimentación, clases extras, etc., por lo que la implementación de un sistema que automatice mencionados procesos ayudaría a la mejor administración de los ingresos económicos con los que cuenta la institución.

La implementación de la aplicación web de cobro de rubros permitirá llevar un control más organizado de los diferentes pagos cancelados por los estudiantes, así también generar reportes mensuales, la creación de rubros para la planilla de pago personalizada de cada estudiante según los servicios de los cuáles se desea beneficiar.

1.5. Hipótesis

La implementación de una aplicación web bajo el patrón de diseño Modelo Vista Controlador con un lenguaje de software libre seleccionado mejorará la productividad de desarrollo.

CAPÍTULO II

MARCO TEÓRICO

2.1. Patrones de Diseño

Normalmente cuando se codifica hay pequeños trozos de código que se crean, a veces sin pensar en ellos porque resultan evidentes. Si ese código sólo sirve para ese programa, todo va más o menos bien.

Sin embargo, es normal que luego se tenga que hacer otro programa similar. Es bastante normal también que se copie el primer programa o trozos del mismo en otro sitio y se empiece a modificar, porque el código que se debe hacer se parece al que ya se tenía, pero no es exactamente igual.

Cuando un programador se ve obligado a hacer eso una y otra vez, a copiar y pegar el mismo código y modificarlo una y otra vez, empieza a pensar en la forma de hacer ese trozo de código de forma que la tarea de llevárselo a otro sitio sea

más cómoda para él. El programador empieza a inventar diferentes formas de hacer ese trozo de código de forma que sea lo suficientemente versátil y configurable como para no tener que volver a tocarlo nunca más. Idealmente, le debería bastar con llevarse la librería ya compilada de un programa a otro.

Ha habido en la historia miles de programadores con este problema y miles de ellos han llegado a las mismas soluciones o formas de hacer esos trozos de código de forma que sean completamente reutilizables. Esas soluciones son los "patrones de diseño".

Una de las cosas que los programadores y desarrolladores han aprendido en base a la experiencia es que no deben resolver los problemas como si fuese la primera vez que se presentan. Para lograrlo, reutilizan el código que desarrollaron en el pasado, es decir que, cuando encuentran una buena solución a un problema concreto, recurren a ella una y otra vez. Los patrones de diseño suelen resolver problemas específicos del diseño y hacen que éste sea más flexible, elegante y reutilizable.

2.1.1. Historia

La idea de los patrones de diseño de software nació originalmente del campo de arquitectura. Christopher Alexander, un arquitecto, escribió dos libros revolucionarios que describen los patrones de arquitectura y de planificación urbana: A Pattern Language: pueblos, edificios, construcción (1977) y The Timeless Way of Building (1979). Las ideas presentadas en esos libros son aplicables a varios campos fuera de arquitectura, incluso el desarrollo del software.

En 1987, Ward Cunningham y Kent Beck usaron algunas de las ideas de Alexander para desarrollar cinco modelos para el diseño de interfaces de usuario. Ellos publicaron un estudio en OOPSLA-87 titulado “Using Pattern Languages for Object-Oriented Programs.”

En los años noventa, Erich Gamma, Richard Helm, John Vlissides, and Ralph Johnson empezaron a trabajar en uno de los libros de computación más influyente de la última década: Design Patterns. Publicado en 1994 y a menudo llamado la “Gang of Four” (Banda de los Cuatro) o GoF, este popularizó la idea de patrones de diseño.

En los últimos años los patrones han ido ganando aceptación, y se fueron extendiendo a otras áreas dentro del desarrollo y mantenimiento de software.

2.1.2. Definición

Un Patrón de Diseño es una solución repetible a un problema recurrente en el diseño de software. Esta solución no es un diseño terminado que puede traducirse directamente a código, sino más bien una descripción sobre cómo resolver el problema, la cual puede ser utilizada en diversas situaciones. Los patrones de diseño reflejan todo el rediseño y re-modificación que los desarrolladores han ido haciendo a medida que intentaban conseguir mayor reutilización y flexibilidad en su software.

Los patrones documentan y explican problemas de diseño, y luego discuten una buena solución a dicho problema. Con el tiempo, los patrones comienzan a incorporarse al conocimiento y experiencia colectiva de la industria del software, lo que demuestra que el origen de los mismos radica en la práctica misma más que en la teoría.

Christopher Alexander el arquitecto a quién se le atribuye el inicio de los patrones de diseño, los describe como: “cada patrón describe un problema que ocurre una y otra vez en nuestro entorno, así como la solución a ese problema, de tal modo que se pueda aplicar esta solución un millón de veces, sin hacer lo mismo dos veces”. Si bien ésta definición es sobre patrones de ciudades y edificios la idea es

aplicable a la industria del software: encontrar una solución a un problema dentro de un contexto. Un patrón de diseño nomina, abstrae e identifica los aspectos clave de una estructura de diseño común, lo que los hace útiles para crear un diseño orientado a objetos reutilizable. El patrón de diseño identifica las clases e instancias participantes, sus roles y colaboraciones, y la distribución de responsabilidades. Cada patrón de diseño se centra en un problema concreto, describiendo cuándo aplicarlo y si tiene sentido hacerlo teniendo en cuenta otras restricciones de diseño, así como las consecuencias, ventajas e inconvenientes de su uso.

2.1.3. Elementos de los Patrones de Diseño

En general, un patrón tiene cuatro elementos esenciales:

a. El **nombre del patrón** permite describir, en una o dos palabras, un problema de diseño junto con sus soluciones y consecuencias. Al dar nombre a un patrón inmediatamente estamos incrementado nuestro vocabulario de diseño, lo que nos permite diseñar a un mayor nivel de abstracción. Tener un vocabulario de patrones nos facilita pensar en nuestros diseños y transmitirlos a otros, junto con sus ventajas e inconvenientes. Encontrar buenos nombres ha sido una de las partes más difíciles al desarrollar nuestro catálogo.

b. El **problema** describe cuándo aplicar el patrón. Explica el problema y su contexto. Puede describir problemas concretos de diseño (por ejemplo, cómo representar algoritmos como objetos), así como las estructuras de clases u objetos que son sintomáticas de un diseño inflexible. A veces, el problema incluye una serie de condiciones que deben darse para que tenga sentido aplicar el patrón.

c. La **solución** describe los elementos que constituyen el diseño, sus relaciones, responsabilidades y colaboraciones. La solución no describe

un diseño o una implementación en concreto, sino que un patrón es más bien como una plantilla que puede aplicarse en muchas situaciones diferentes. El patrón proporciona una descripción abstracta de un problema de diseño y cómo una lo resuelve una disposición general de elementos (en nuestro caso, clases y objetos).

d. Las **consecuencias** son los resultados así como las ventajas e inconvenientes de aplicar el patrón. Aunque cuando se describen decisiones de diseño muchas veces no se reflejan sus consecuencias, éstas son críticas para evaluar las alternativas de diseño y comprender los costes y beneficios de aplicar el patrón.

Las consecuencias en el software suelen referirse al equilibrio entre espacio y tiempo. También pueden tratar cuestiones de lenguaje e implementación. Por otro lado, puesto que la reutilización suele ser uno de los factores de los diseños orientados a objetos, las consecuencias de un patrón incluyen su impacto sobre la flexibilidad, extensibilidad y portabilidad de un sistema. Incluir estas consecuencias de un modo.

2.1.4. Describir Patrones de Diseño

Para reutilizar el diseño, se debe hacer constar las decisiones, alternativas, ventajas e inconvenientes que dieron lugar a él. También son importantes los ejemplos concretos, porque ayudan a ver el diseño en acción. Es así que el libro “Design Patterns: Elements of Reusable Object-Oriented Software” describe una plantilla la cual se encuentra dividida en secciones con el objetivo de facilitar el aprendizaje, comprensión y uso de los patrones de diseño.

La plantilla propuesta para la descripción de patrones de diseño es la siguiente:

Nombre del Patrón y Clasificación

El nombre del patrón transmite brevemente su esencia. Un buen nombre es vital, porque pasará a formar parte del vocabulario de diseño. La clasificación del patrón refleja el esquema que se presenta en la sección

Propósito

Una frase breve que responde a las siguientes cuestiones: ¿Qué hace este patrón de diseño? ¿En qué se basa? ¿Cuál es el problema concreto de diseño que resuelve?

También Conocido Como

Otros nombres, si existen, por los que se conoce al patrón.

Motivación

Un escenario que ilustra un problema de diseño y cómo las estructuras de clases y objetos del patrón resuelven el problema. El escenario ayudará a entender la descripción que sigue.

Aplicabilidad

¿En qué situaciones se puede aplicar el patrón de diseño? ¿Qué ejemplos hay de malos diseños que el patrón puede resolver? ¿Cómo se puede reconocer dichas situaciones?

Estructura

Una representación gráfica de las clases del patrón usando una notación basada en la Técnica de Modelado de Objetos. También se hace uso de diagramas de interacción para mostrar secuencias de peticiones y colaboraciones entre objetos.

Participantes

Las clases y objetos participantes en el patrón de diseño, junto con sus responsabilidades.

Colaboraciones

Cómo colaboran los participantes para llevar a cabo sus responsabilidades.

Consecuencias

¿Cómo consigue el patrón sus objetivos? ¿Cuáles son las ventajas e inconvenientes y los resultados de usar el patrón? ¿Qué aspectos de la estructura del sistema se pueden modificar de forma independiente?

Implementación

¿Cuáles son las dificultades, trucos o técnicas que se deberían tener en cuenta a la hora de aplicar el patrón? ¿Hay cuestiones específicas del lenguaje?

Código de ejemplo

Fragmentos de código que muestran cómo se puede implementar el patrón.

Usos conocidos

Ejemplos del patrón en sistemas reales.

Patrones relacionados

¿Qué patrones de diseño están estrechamente relacionados con éste? ¿Cuáles son las principales diferencias? ¿Con qué otros patrones deberían usarse?

2.1.5. Tipos de Patrones de Diseño

Los patrones generalmente se clasifican en:

- De Creación
- Estructurales
- De Comportamiento

2.1.5.1. De Creación: Abstraen el proceso de creación de instancias de objetos. Ayudan a hacer a un sistema independiente de cómo se crean, se componen y se representan sus objetos.

- Abstract Factory (Fábrica abstracta): Permite trabajar con objetos de distintas familias de manera que las familias no se mezclen entre sí y haciendo transparente el tipo de familia concreta que se esté usando.
- Builder (Constructor virtual): Abstrae el proceso de creación de un objeto complejo, centralizando dicho proceso en un único punto.
- Factory Method (Método de fabricación): Centraliza en una clase constructora la creación de objetos de un subtipo de un tipo determinado, ocultando al usuario la casuística para elegir el subtipo que crear.
- Prototype (Prototipo): Crea nuevos objetos clonándolos de una instancia ya existente.
- Singleton (Instancia única): Garantiza la existencia de una única instancia para una clase y la creación de un mecanismo de acceso global a dicha instancia.

2.1.5.2. Estructurales: Tratan con la composición de clases u objetos. Se ocupan de cómo las clases y objetos son utilizados para componer estructuras de mayor tamaño.

- Adapter (Adaptador): Adapta una interfaz para que pueda ser utilizada por una clase que de otro modo no podría utilizarla.
- Bridge (Puente): Desacopla una abstracción de su implementación.
- Composite (Objeto compuesto): Permite tratar objetos compuestos como si se tratase de uno simple.
- Decorator (Envoltorio): Añade funcionalidad a una clase dinámicamente.

- Facade (Fachada): Provee de una interfaz unificada simple para acceder a una interfaz o grupo de interfaces de un subsistema.
- Flyweight (Peso ligero): Reduce la redundancia cuando gran cantidad de objetos poseen idéntica información.
- Proxy: Mantiene un representante de un objeto

2.1.5.3. De Comportamiento: Caracterizan el modo en que las clases y objetos interactúan y se reparten la responsabilidad. Atañen a los algoritmos y a la asignación de responsabilidades entre objetos.

- Chain of Responsibility (Cadena de responsabilidad): Permite establecer la línea que deben llevar los mensajes para que los objetos realicen la tarea indicada.
- Command (Orden): Encapsula una operación en un objeto, permitiendo ejecutar dicha operación sin necesidad de conocer el contenido de la misma.
- Interpreter (Intérprete): Dado un lenguaje, define una gramática para dicho lenguaje, así como las herramientas necesarias para interpretarlo.
- Iterator (Iterador): Permite realizar recorridos sobre objetos compuestos independientemente de la implementación de estos.
- Mediator (Mediador): Define un objeto que coordine la comunicación entre objetos de distintas clases, pero que funcionan como un conjunto.
- Memento (Recuerdo): Permite volver a estados anteriores del sistema.
- Observer (Observador): Define una dependencia de uno-a-muchos entre objetos, de forma que cuando un objeto cambie de estado se notifique y actualicen automáticamente todos los objetos que dependen de él.
- State (Estado): Permite que un objeto modifique su comportamiento cada vez que cambie su estado interno.

- Strategy (Estrategia): Permite disponer de varios métodos para resolver un problema y elegir cuál utilizar en tiempo de ejecución.
- Template Method (Método plantilla): Define en una operación el esqueleto de un algoritmo, delegando en las subclases algunos de sus pasos, esto permite que las subclases redefinan ciertos pasos de un algoritmo sin cambiar su estructura.
- Visitor (Visitante): Permite definir nuevas operaciones sobre una jerarquía de clases sin modificar las clases sobre las que opera.

2.2. Patrón Modelo Vista Controlador

2.2.1. Descripción

Un problema muy común para los programadores es la reutilización del código que ya tienen hecho. A veces hay que resolver un problema parecido a algo que ya tenemos hecho, mejorar el aspecto de un programa o mejorar su algoritmo. Esta tarea se facilita mucho si a la hora de programar tenemos la precaución de separar el código en varias partes que sean susceptibles de ser reutilizadas sin modificaciones.

La arquitectura MVC separa la lógica de negocio (el modelo) y la presentación (la vista) por lo que se consigue un mantenimiento más sencillo de las aplicaciones. Si por ejemplo una misma aplicación debe ejecutarse tanto en un navegador estándar como en un navegador de un dispositivo móvil, solamente es necesario crear una vista nueva para cada dispositivo; manteniendo el controlador y el modelo original.

MVC es usado principalmente en aplicaciones que manejan gran cantidad de datos y transacciones complejas donde se requiere una mejor separación de conceptos para que el desarrollo esté estructurado de una mejor manera, facilitando la programación en diferentes capas de manera paralela e independiente.

El controlador se encarga de aislar al modelo y a la vista de los detalles del protocolo utilizado para las peticiones (HTTP, consola de comandos, email, etc.). El modelo se encarga de la abstracción de la lógica relacionada con los datos, haciendo que la vista y las acciones sean independientes de, por ejemplo, el tipo de gestor de bases de datos utilizado por la aplicación.

Una vista debe asegurarse de que su apariencia refleja el estado del modelo. Cada vez que cambian los datos del modelo, éste se encarga de avisar a las vistas que dependen de él. Como respuesta a dicha notificación, cada vista tiene la oportunidad de actualizarse a sí misma. Este enfoque permite asignar varias vistas a un modelo para ofrecer diferentes presentaciones. También se pueden crear nuevas vistas de un modelo sin necesidad de volver a escribir éste.

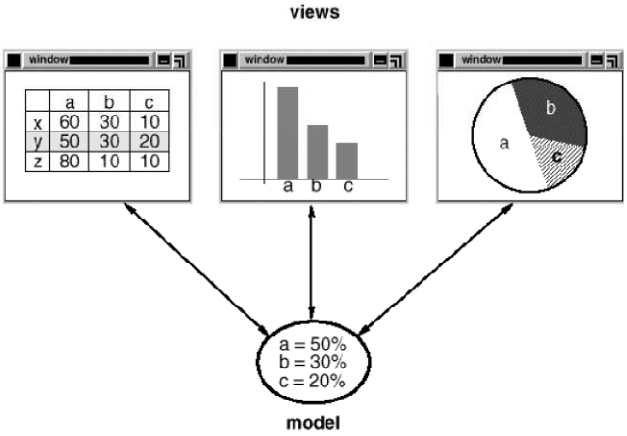


Ilustración II.01: Interacción simplificada entre modelo y vista.

2.2.2. Historia de MVC

La arquitectura MVC (Model/View/Controller) fue introducida como parte de la versión Smalltalk-80 del lenguaje de programación Smalltalk. Fue diseñada para reducir el esfuerzo de programación necesario en la implementación de sistemas múltiples y sincronizados de los mismos datos. Inicialmente diseñado para Aplicaciones de escritorio cuya idea principal era separar la presentación del modelo de dominio.

En aquel entonces la vista se encargaba del dibujado de mapas de bits, el controlador atendía acciones del mouse y teclado, y el modelo guardaba el estado de la ventana y la información mostrada en ella. Cada vista estaba asociada con un controlador y un modelo en su creación. Cualquier modelo podía ser asociado débilmente a cualquier vista ya que un modelo podía relacionarse con varias vistas. El controlador no se encargaba de manejar la lógica del negocio en la aplicación, su propósito era manejar la interacción del mouse y el teclado, y actualizar la vista acorde, lo cual no era una tarea trivial.

2.2.3. Modelo

El modelo suele ser independiente de cómo se desea que el programa recoja los resultados o cómo los presente. Por ejemplo, si se quiere hacer un juego de ajedrez, todas las reglas del ajedrez son totalmente independientes de cómo se va a dibujar el tablero en 3D o plano, con figuras blancas y negras tradicionales o cualquier otro tipo de figura. Este código constituiría el **modelo**. En el juego del ajedrez el modelo podría ser una clase (o conjunto de clases) que mantengan un registro piezas, que permita mover dichas piezas verificando que los movimientos son legales, que detecte los jaques, jaque mate, tablas, etc. De hecho, las metodologías orientadas a objeto nos introducen en este tipo de clases, a las que llaman **clases del negocio**.

Las principales características del Modelo son:

- Consiste en un conjunto de Clases y Objetos correspondientes al *Modelo del Negocio* para nuestra aplicación (estados y funcionalidad)
- Tiene un bajo acoplamiento con Vistas y Controladores

- En él definen métodos para realizar consultas (informar el estado), comandos (modificar el estado) y mecanismos de notificación (para informar a los *observadores* / vistas).

2.2.4. Vista

Es la representación del modelo en forma gráfica disponible para la interacción con el usuario. En el caso de una aplicación Web, la Vista es una página HTML con contenido dinámico sobre el cual el usuario puede realizar operaciones. En el ejemplo del ajedrez serían las posibles interfaces gráficas (3D, plano). Esta parte del código es la **vista**. Consiste en la ventana que dibuja el tablero con las figuras de las piezas, que permiten arrastrar con el ratón una pieza para moverla, botones, lista visual con los movimientos realizados, etc.

Las principales características de las Vistas son:

- Administra la visualización y presentación de la información
- Observa al Modelo para actualizar los cambios
- Al definirse en el modelo una interfaz clara y estable, es fácil implementar múltiples Vistas para un mismo modelo
- Altamente dependiente del dispositivo y tecnología de visualización
- Muy dependiente del Modelo (debe conocerlo).

2.2.5. Controlador

Es la capa encargada de manejar y responder las solicitudes del usuario, procesando la información necesaria y modificando el Modelo en caso de ser necesario. Es código que no tiene que ver con las ventanas visuales ni con las reglas de nuestro modelo. En el ejemplo formarían parte de esto el algoritmo para pensar las jugadas (el más complejo de todo el juego).

Las principales características del Controlador son:

- Responsable de definir el comportamiento de la aplicación
- Recibe los eventos del usuario y decide qué es lo que se debe hacer, mapeándolos en comandos (mensajes) hacia el Modelo
- Altamente dependiente de los dispositivos y mecanismos de interacción del usuario
- Muy dependiente del Modelo (debe conocerlo).

2.2.6. Frameworks de MVC

2.2.6.1. Ruby on Rails (Ruby)

Ruby on Rails, también conocido como RoR o Rails es un framework de aplicaciones web de código abierto escrito en el lenguaje de programación Ruby, siguiendo el paradigma de la arquitectura Modelo Vista Controlador (MVC). Trata de combinar la simplicidad con la posibilidad de desarrollar aplicaciones del mundo real escribiendo menos código que con otros frameworks y con un mínimo de configuración. El lenguaje de programación Ruby permite la metaprogramación, de la cual Rails hace uso, lo que resulta en una sintaxis que muchos de sus usuarios encuentran muy legible. Rails se distribuye a través de RubyGems, que es el formato oficial de paquete y canal de distribución de bibliotecas y aplicaciones Ruby.

2.2.6.2. Struts (Java / J2ee)

Struts es una herramienta de soporte para el desarrollo de aplicaciones Web bajo el patrón MVC bajo la plataforma J2EE (Java 2, Enterprise Edition). Struts se desarrollaba como parte del proyecto Jakarta de la Apache Software Foundation, pero actualmente es un proyecto independiente conocido como Apache Struts.

Permite reducir el tiempo de desarrollo. Su carácter de "software libre" y su compatibilidad con todas las plataformas en las que Java Entreprise esté disponible lo convierten en una herramienta altamente disponible.

Se basa en el Framework del Modelo-Vista-Controlador (MVC) el cual se utiliza ampliamente y es considerado de gran solidez. De acuerdo con este Framework, el procesamiento se separa en tres secciones diferenciadas llamadas el modelo, las vistas y el controlador.

2.2.6.3. Catalyst (Perl)

Catalyst es una estructura de código libre para aplicaciones web escrito en Perl. Soporta la arquitectura MVC, así como soporta algunos patrones web experimentales. Está altamente inspirado en Ruby on Rails, Maypole y Spring.
Catalyst promueve el re-utilizamiento de los módulos de Perl que ya soportan bien lo que requieren las páginas Web.

La forma en que Catalyst soporta la arquitectura MVC es la siguiente:

- La parte de Modelo (Model) es manejada por medio de DBIx:Class, Plucene, Net::LDAP u otras clases modelo.
- La parte de Vista (View) es usualmente manejada por Template Toolkit, Mason o HTML::Template.
- La parte de Control (Controller) es escrita por el autor, por supuesto. Grandes pedazos de funcionalidad usualmente se pueden conseguir con los plugins de Catalyst (ejemplo: Catalyst::Plugin::FormValidator, Catalyst::Plugin::Prototype, Catalyst::Plugin::Account::AutoDiscovery, etc.).

Catalyst provee ayudas para simplificar el control de flujo y mapeo de URLs para los métodos de Control.

2.2.6.4. Symfony (PHP)

Symfony es un completo framework diseñado para optimizar el desarrollo de las aplicaciones web mediante algunas de sus principales características. Para empezar, separa la lógica de negocio, la lógica de servidor y la presentación de la aplicación web. Proporciona varias herramientas y clases encaminadas a reducir el tiempo de desarrollo de una aplicación web compleja. Además, automatiza las tareas más comunes, permitiendo al desarrollador dedicarse por completo a los aspectos específicos de cada aplicación. El resultado de todas estas ventajas es que no se debe reinventar la rueda cada vez que se crea una nueva aplicación web.

Symfony está desarrollado completamente con PHP 5. Ha sido probado en numerosos proyectos reales y se utiliza en sitios web de comercio electrónico de primer nivel. Symfony es compatible con la mayoría de gestores de bases de datos, como MySQL, PostgreSQL, Oracle y Microsoft SQL Server. Se puede ejecutar tanto en plataformas *nix (Unix, Linux, etc.) como en plataformas Windows.

2.2.7. Ventajas

Las principales ventajas de hacer uso del patrón MVC son:

- La separación del Modelo de la Vista, es decir, separar los datos de la representación visual de los mismos.
- Es mucho más sencillo agregar múltiples representaciones de los mismos datos o información.

- Facilita agregar nuevos tipos de datos según sea requerido por la aplicación ya que son independientes del funcionamiento de las otras capas.
- Crea independencia de funcionamiento.
- Facilita el mantenimiento en caso de errores.
- Ofrece maneras más sencillas para probar el correcto funcionamiento del sistema.
- Permite el escalamiento de la aplicación en caso de ser requerido.

2.2.8. Desventajas

Las desventajas de seguir el planteamiento de MVC son:

- La separación de conceptos en capas agrega complejidad al sistema.
- La cantidad de archivos a mantener y desarrollar se incrementa considerablemente.
- La curva de aprendizaje del patrón de diseño es más alta que usando otros modelos más sencillos.

CAPÍTULO III

Análisis entre los frameworks Rails y Symfony para determinar la mejor aplicabilidad del patrón de diseño MVC

3.1. Generalidades de Ruby on Rails

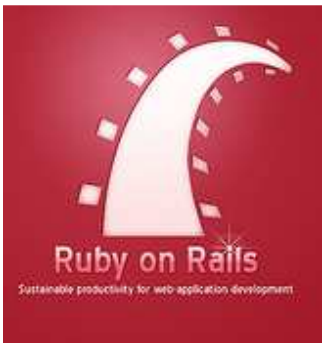


Ilustración III.02: Logotipo de Ruby on Rails

Ruby on Rails es un framework que hace que sea más fácil desarrollar, implementar y realizar mantenimientos a aplicaciones web. Durante los meses que

siguieron a su lanzamiento inicial, Rails pasó de ser un juguete desconocido a ser un fenómeno mundial. Rails ha ganado premios, y, más importante, se ha convertido en el framework elegido para la implementación de una amplia gama de las llamadas aplicaciones Web 2.0.

Las aplicaciones Rails están escritas en Ruby un lenguaje de scripts moderno, orientado a objetos. Ruby es conciso se puede expresar ideas de forma natural y limpiamente en su código. Esto lleva a los programas que son sencillos de escribir (no menos importante) a que meses después sean fáciles de leer y entender.

Una de las características más interesantes de Rails es que impone algunas restricciones sobre cómo estructurar sus aplicaciones web.
Sorprendentemente, estas restricciones hacen más fácil crear aplicaciones web.

Características e Historia de Ruby

Ruby es un lenguaje de scripts, multiplataforma, netamente orientado a objetos es software libre, fue creado por Yukihiro Matsumoto conocido como Matz. La primera versión fue liberada en 1995, hereda varias características de lenguajes como: Perl, Smalltalk, Eiffel, Ada y Lisp. Como lo indica su propio autor, es un lenguaje “aparentemente sencillo pero internamente complejo”.

Ruby fue diseñado para un desarrollo rápido y sencillo. Cada día este lenguaje va ganando más adeptos, tanto así que la empresa Sun Microsystems, está apoyando un proyecto llamado Jruby que es un intérprete de Ruby escrito 100% en Java.

Entre las características del lenguaje se encuentran:

- Posibilidad de hacer llamadas directamente al sistema operativo.
- Muy potente para el manejo de cadenas y expresiones regulares.
- No se necesita declarar las variables.
- La sintaxis es simple y consistente.

- Gestión de memoria automática.
- Todo es un objeto.
- Métodos Singleton.

Características e historia de Rails

Rails es un framework para el desarrollo de aplicaciones web, software libre por naturaleza, está basado en el patrón de diseño Modelo Vista Controlador (MVC). Fue creado por David Heinemeier Hansson, empleado de la empresa 37signals.

Fue liberado por primera vez al público en julio del 2004, y lo implementó en una aplicación orientada a la administración de proyectos llamada Basecamp. Actualmente se unieron más personas al desarrollo. Rails está basado en estos principios de desarrollo:

- Don't Repeat Yourself
- Convention Over Configuration

Primer principio:

La primera regla significa "No lo vuelvas a repetir", es una de las cosas más novedosas que se ha podido encontrar en este framework. Ejemplo: Se puede tener un formulario, y llamarlo las veces que se quiera y desde donde quiera, simplemente con una línea código.

Cuando se tiene una tabla en una base de datos, se puede manipular a los registros como un objeto y a sus campos como un atributo, sin necesidad de declarar nada, son sólo algunas aplicaciones de este principio de desarrollo.

El segundo principio:

"Convención antes que Configuración", con esto el framework nos dice que si usamos algo con frecuencia de una forma en particular y siempre se tiene que configurar, el lenguaje lo configura. Lo que significa de ahorro de tiempo para el

desarrollador. Pero si el desarrollador irrespeta esa configuración, el lenguaje no crea ningún conflicto.

Implementación

Ruby on Rails es a menudo utiliza RubyGems, que es un gestor de paquetes que se incluye con Ruby. Muchas distribuciones de Linux también soporte para el montaje de los carriles y sus dependencias a través de su nativa sistema de gestión de paquetes .

Ruby on Rails es normalmente integrado con un servidor de base de datos como MySQL y un servidor web como Apache.

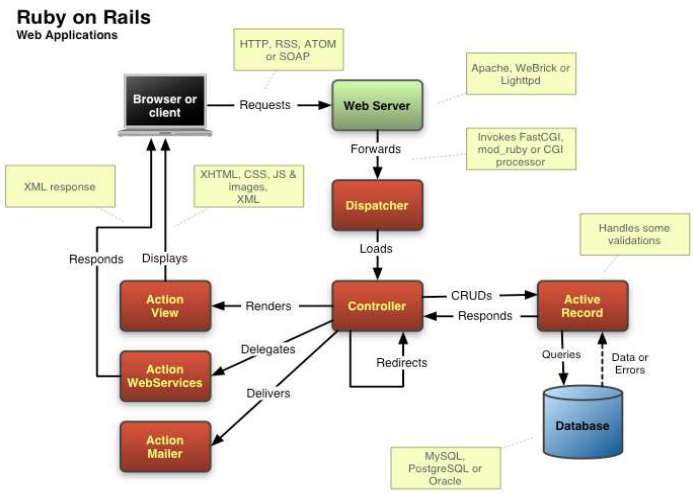


Ilustración III.03: Procedimiento de una petición de Ruby on Rails en la web

Depurador

Ruby posee herramientas de depuración, se pueden encontrar en el archivo de la biblioteca debug.rb. El depurador en Ruby le permitirá ejecutar el programa una instrucción a la vez con pausas entre ellos. En cada pausa se puede inspeccionar las variables o valores de su código, localizar su posición en una serie de comandos anidados y después de todas las inspecciones se puede reanudar la ejecución. Los *puntos de interrupción* (Breakpoints) son otra característica del

depurador de Ruby. La facilidad del punto de interrupción es que se puede detener el depurador cuando se desee.

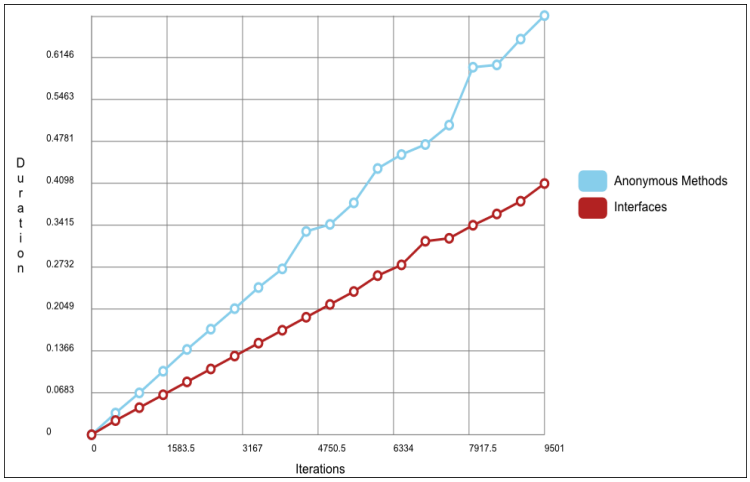


Ilustración III.04: Gráfica Iteraciones vs Duración.

Perfiles

Algunos de los perfiladores para Ruby incluyen:

Modos: Ruby puede medir una serie de parámetros que incluye el uso de memoria, los tiempos de llamada, y la asignación de objeto.

Informes: puede generar informes HTML con referencia cruzada y texto.

Llamadas recursivas: soporta nuevos métodos de llamadas recursivas

Temas: Puede soportar perfiles de varios hilos simultáneamente

El uso de perfiles también identifica las áreas en su programa que utiliza muchos recursos del sistema. Se puede conocer las áreas que están utilizando muchos recursos y tener la posibilidad de volver a escribir el código, ya sea, una parte o por completo y hacer que el programa se ejecute de manera eficiente.

3.2. Arquitectura de Aplicaciones en Rails

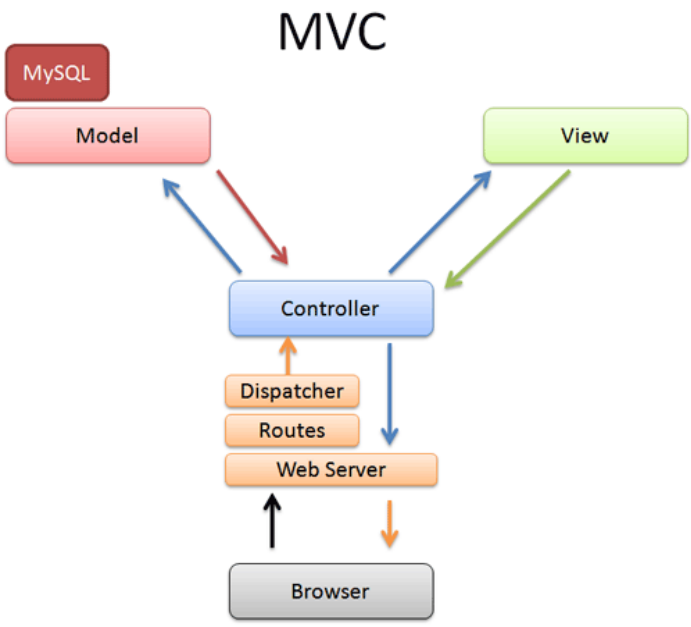


Ilustración III.05: Interacción Modelo Vista Controlador

3.2.1. Active Record: El Modelo

En las aplicaciones web orientadas a objetos sobre bases de datos, el *Modelo* consiste en las clases que representan a las tablas de la base de datos.

En Ruby on Rails, las clases del Modelo son gestionadas por ActiveRecord. Por lo general, lo único que tiene que hacer el programador es heredar de la clase *ActiveRecord::Base*, y el programa averiguará automáticamente qué tabla usar y qué columnas tiene.

Las definiciones de las clases también detallan las relaciones entre clases con sentencias de mapeo objeto relacional. Por ejemplo, si la clase *Imagen* tiene una definición *has_many:comentarios*, y existe una instancia de *Imagen* llamada a, entonces *a.comentarios* devolverá un array con todos los objetos Comentario cuya columna *imagen_id* (en la tabla comentarios) sea igual a *a.id.*

El modelo representa:

- Las Tablas de la Base de Datos.
- Migraciones (Expresan Cambios en las BD)
- Observadores

3.2.2. Vistas y su aplicación en archivos .rhtml (Ruby embebido)

En MVC, *Vista* es la lógica de visualización, o cómo se muestran los datos de las clases del *Controlador*. Con frecuencia en las aplicaciones web la vista consiste en una cantidad mínima de código incluido en HTML.

Existen en la actualidad muchas maneras de gestionar las vistas. El método que se emplea en Rails por defecto es usar Ruby Embebido (archivos.rhtml, desde la versión 2.x en adelante de RoR archivos.html.erb), que son básicamente fragmentos de código HTML con algo de código en Ruby, siguiendo una sintaxis similar a JSP. También pueden construirse vistas en HTML y XML con Builder o usando el sistema de plantillas Liquid.

Es necesario escribir un pequeño fragmento de código en HTML para cada método del controlador que necesita mostrar información al usuario. El "maquetado" o distribución de los elementos de la página se describe separadamente de la acción del controlador y los fragmentos pueden invocarse unos a otros.

3.2.3. Action Pack: La Vista y el Controlador

En MVC, las clases del *Controlador* responden a la interacción del usuario e invocan a la lógica de la aplicación, que a su vez manipula los datos de las clases del *Modelo* y muestra los resultados usando las *Vistas*. En las aplicaciones web basadas en MVC, los métodos del *controlador* son invocados por el usuario usando el navegador web.

La implementación del *Controlador* es manejada por el ActionPack de Rails, que contiene la clase *ApplicationController*. Una aplicación Rails simplemente hereda de esta clase y define las acciones necesarias como métodos, que pueden ser invocados desde la web, por lo general en la forma *http://aplicacion/ejemplo/metodo*, que invoca a *EjemploController#metodo*, y presenta los datos usando el archivo de plantilla */app/views/ejemplo/metodo.html.erb*, a no ser que el método redirija a algún otro lugar.

Rails también proporciona *andamiaje*, que puede construir rápidamente la mayor parte de la lógica y vistas necesarias para realizar las operaciones más frecuentes.

3.3. Generalidades de PHP y Simfony

3.3.1. Descripción

Symfony es un completo framework diseñado para optimizar, gracias a sus características, el desarrollo de las aplicaciones web. Para empezar, separa la lógica de negocio, la lógica de servidor y la presentación de la aplicación web. Proporciona varias herramientas y clases encaminadas a reducir el tiempo de desarrollo de una aplicación web compleja.

Además, automatiza las tareas más comunes, permitiendo al desarrollador dedicarse por completo a los aspectos específicos de cada aplicación. El resultado de todas estas ventajas es que no se debe reinventar la rueda cada vez que se crea una nueva aplicación web. Symfony está desarrollado completamente con PHP 5. Ha sido probado en numerosos proyectos reales y se utiliza en sitios web de comercio electrónico de primer nivel. Symfony es compatible con la mayoría de gestores de bases de datos, como MySQL, PostgreSQL, Oracle y SQL Server de Microsoft. Se puede ejecutar tanto en plataformas *nix (Unix, Linux, etc.) como en plataformas Windows.

3.3.2. Características

Symfony se diseñó para que se ajustara a los siguientes requisitos:

- Fácil de instalar y configurar en la mayoría de plataformas (y con la garantía de que funciona correctamente en los sistemas Windows y *nix estándares)Independiente del sistema gestor de bases de datos
- Sencillo de usar en la mayoría de casos, pero lo suficientemente flexible como para adaptarse a los casos más complejos
- Basado en la premisa de *"convenir en vez de configurar"*, en la que el desarrollador solo debe configurar aquello que no es convencional
- Sigue la mayoría de *mejores prácticas* y patrones de diseño para la web
- Preparado para aplicaciones empresariales y adaptable a las políticas y arquitecturas propias de cada empresa, además de ser lo suficientemente estable como para desarrollar aplicaciones a largo plazo
- Código fácil de leer que incluye comentarios de phpDocumentor y que permite un mantenimiento muy sencillo
- Fácil de extender, lo que permite su integración con librerías desarrolladas por terceros.

3.3.3. Historia

La primera versión de Symfony fue publicada en Octubre de 2005 por Fabien Potencier presidente de Sensio, una empresa francesa de desarrollo de aplicaciones web conocida por sus innovaciones en este campo.

En el año 2003, Fabien realizó una investigación sobre las herramientas de software libre disponibles para el desarrollo de aplicaciones web con PHP. Llegando a la conclusión de que no existía ninguna herramienta con esas características. Después del lanzamiento de la versión 5 de PHP, decidió que las herramientas disponibles habían alcanzado un grado de madurez suficiente como para integrarlas en un framework completo. Fabien empleó un año e entero para desarrollar el núcleo de Symfony, basando su trabajo en el framework Mojavi (que también era un framework que seguía el funcionamiento MVC), en la herramienta

Propel para el mapeo de objetos a bases de datos (conocido como ORM, de "object-relational mapping") y en los helpers empleados por Ruby on Rails en sus plantillas.

Symfony se desarrollo para utilizarlo en los proyectos de Sensio, ya que disponer de un framework efectivo es la mejor ayuda para el desarrollo eficiente y rápido de las aplicaciones. Además, el desarrollo web se hace más intuitivo y las aplicaciones resultantes son más robustas y más fáciles de mantener.

El framework se utilizó por primera vez en el desarrollo de un sitio de comercio electrónico para un vendedor de lencería y posteriormente se utilizó en otros proyectos.

Después de utilizar Symfony en algunos proyectos, Fabien decidió publicarlo bajo una licencia de software libre. Sus razones para liberar el proyecto fueron para donar su trabajo a la comunidad, aprovechar la respuesta de los usuarios, mostrar la experiencia de Sensio y porque consideraba que era divertido hacerlo.

3.4. Estructura del proyecto en Symfony: Aplicaciones, Módulos y Acciones

3.4.1. El modelo, la vista y el controlador

El Modelo define la lógica de negocio. Symfony guarda todas las clases y archivos relacionados con el modelo en el directorio lib/model/, los mismos que son creadas con la ayuda de un ORM como Propel o Doctrine.

La Vista es con lo que el usuario interactúa. En Symfony, la vista es principalmente la capa de plantillas PHP. Estas son guardadas en el directorio /templates de los distintos módulos implementado. Las plantillas pueden utilizar helpers para tareas recurrentes como crear una URL o un enlace. Una plantilla puede ser decorada por un layout para abstraerse del encabezado y pie de las páginas.

El Controlador es la parte de código que llama al Modelo para obtener algunos datos que le pasa a la Vista para la presentación al cliente por medio de este se

verifica la integridad de las peticiones y preparan los datos requeridos por la capa de presentación.

3.5. Estudio comparativo entre los lenguajes y sus frameworks

Valoración de parámetros

Para evaluar y analizar las herramientas Ruby y Php con sus frameworks Rails y Symfony, respectivamente, se consideraron los siguientes valores:

Valor	Significado
0	No dispone
1	Regular
2	Bueno
3	Muy Bueno
4	Excelente

Tabla III.I: Valores para las comparaciones

El máximo valor es 4 lo que significa que el parámetro evaluado satisface completamente las expectativas. Para llegar a una conclusión general de la herramienta evaluada y conseguir resultados cuantitativos y cualitativos se realizará una sumatoria de los valores obtenidos, para lo cual se utilizará la siguiente fórmula:

$$\boldsymbol{Total} = \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \boldsymbol{VPi}$$

Donde:

VP: Valor Parámetro de evaluación

i: Parámetro específico

n: Total de parámetros evaluados

Total: Resultado de la sumatoria de todos los valores obtenidos

La misma fórmula se aplicará para alcanzar los resultados de la evaluación con las dos herramientas de forma individual.

3.5.1. Parámetros globales

Estos parámetros se refieren a las características que poseen las herramientas de forma general. Esto quiere decir que son cualidades que no se consideran dentro del modelo, la vista o el controlador. Para evaluar los parámetros globales se ha tomado en cuenta otros parámetros de manera más específica; cada uno de ellos tendrá su respectiva apreciación según los valores definidos anteriormente.

3.5.1.1. Análisis de Parámetros

- **Estructura del proyecto**

La estructura del proyecto se refiere al modo en que los frameworks actúan cuando se crea una nueva aplicación, y la manera como organiza los diferentes archivos vinculados a éste. La estructura está dada por el patrón MVC por lo que la valoración está dad por:

- ✓ Modelo
- ✓ Vista
- ✓ Controlador

Ruby on Rails

El framework utiliza una herramienta denominada *rails* que genera la estructura base del proyecto. Esta estructura es ordena en carpetas basadas en el patrón de diseño MVC. En la carpeta *app* se alojan los ficheros de código ruby organizados en las carpetas: *models, views, controllers* y una adicional llamada *helpers.* Adicional a ello se utiliza el comando *scaffold* que es el encargado de crear los ficheros para el modelo, la vista y el controlador.

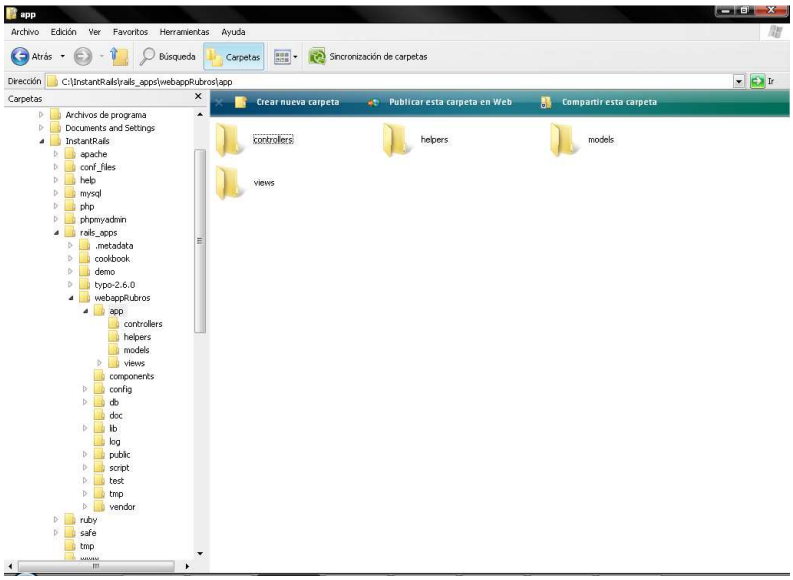


Ilustración III.06: Estructura de un proyecto en RoR.

Symfony

Un proyecto en symfony consiste en un conjunto de aplicaciones y estas a su vez se componen de módulos. Al momento de crear un nuevo proyecto se genera automáticamente toda una estructura de directorios que permite normalizar el desarrollo de aplicaciones facilitando la búsqueda de código, corrección de errores y adición de nuevas funciones. Los directorios que se crean son: bajo lib está el modelo que es general para todo el proyecto, bajo apps/'nombre de la aplicación' están la vista y el controlador que se crean uno por cada módulo implementado.

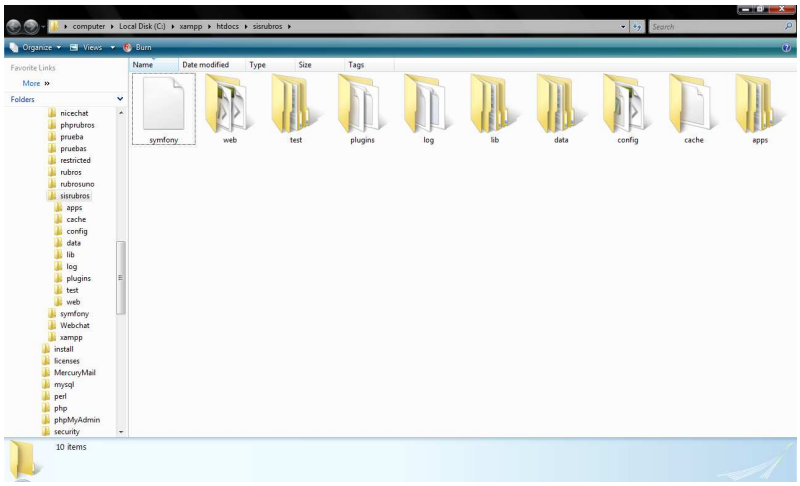


Ilustración III.07: Estructura del proyecto en Symfony.

- **Manejo de errores**

Dentro de este parámetro se van a evaluar la forma en que responden las herramientas frente a un error en tiempo de ejecución. Los parámetros a evaluar son:

- ✓ Control de errores de sintaxis
- ✓ Referencia
- ✓ Lógica de programación
- ✓ Respuesta de la aplicación

Ruby on Rails

RoR tiene la posibilidad de manejar excepciones para capturar con mayor detalle los errores. El método que se usa es el *rescue*, cuando encuentra un error se dirige a ese método y puede, incluso, dar un mensaje personalizado si se ha definido para un error específico. En el entorno de desarrollo cuando encuentra un error, se muestra una pantalla en el navegador especificando el error y muestra las líneas de código con sus números de línea. Esto realiza cuando hay un error de lógica o de sintaxis. Cuando se modifica el código sólo se requiere de actualizar el navegador más no de detener la aplicación y sus servicios.

Symfony

Este framework únicamente detecta los errores al momento de compilar la aplicación. Los errores que genera dependen del entorno en que se ejecute, por ejemplo dentro del entorno de desarrollo se generan excepciones las cuales muchas veces no cuentan con la información necesaria e indispensable para localizar el error por ejemplo cuando el error se origina en la parte del modelo. Otra de las herramientas que incluye este entorno es la barra de depuración la cual nos da un resumen de errores, configuración, consulta a la base de datos. Los errores que se controlan son los de sintaxis, lógica de programación,

referencia. En el entorno de producción la aplicación no se detiene sino que devuelve un valor nulo.

- **Estandarización de nombres**

Un estándar es una especificación que regula la realización de ciertos procesos en este caso los nombres con que se llaman a las variables, constantes, clases y funciones de un lenguaje de programación o palabras reservadas del mismo. Se evaluará:

- ✓ Facilidad de interpretación
- ✓ Intuitivo

Ruby on Rails

RoR tiene una sintaxis definida para el uso de variables, constantes, clases y demás. De las más importantes se tiene:

Las *variables de objeto* (o variables de instancia) sólo pueden ser leídas/escritas por el propio objeto. Ej. @nombre, @maximo, @hora_comer

Los *símbolos* son identificadores únicos que se encuentran en varios sitios. Ej. **:**nombre, :edad, :Clase. RoR define su sintaxis como un lenguaje natural de conversación como por ejemplo en algunos de sus métodos utiliza signos de interrogación y admiración.

Symfony

En el momento en que Symfony crea un nuevo proyecto al mismo tiempo genera varios archivos cuyos nombres manejan un estándar estricto y fácil de interpretar. Las diferentes clases que componen el modelo se identifican con el mismo nombre de la tabla de la que se originan, por lo que son fáciles de interpretar y las funciones encargadas de obtener los distintos valores, de la misma, se manejan con los tradicionales get y set. Las vistas se identifican con *nombrevista*Success y

están estrechamente relacionadas con el evento en el controlador llamado execute*Nombrevista* En definitiva Symfony cumple satisfactoriamente con los parámetros de evaluación inicialmente planteados en lo referente a la estandarización de nombres.

- **Entornos de desarrollo**

Este parámetro evalúa la posibilidad que tiene un framework para desarrollarse en diferentes ambientes. Realizar una aplicación en distintos ambientes brinda facilidades para gestionar errores y brindar mantenimiento mientras la aplicación se encuentra en uso. Los ambientes a evaluar son:

- ✓ Desarrollo
- ✓ Producción
- ✓ Pruebas
- ✓ Staging

Ruby on Rails

Rails formaliza la distinción informal que todo desarrollador de software hace entre el código que se ejecuta cuando se está desarrollando una aplicación y el que se ofrece al cliente o se sube a un servidor una vez la aplicación está finalizada.

En el entorno de desarrollo ***(development)*** el código de nuestra aplicación es cargado cada vez que se hace una petición al servidor. Esto baja el rendimiento de respuesta pero es perfecto para el desarrollo de la aplicación ya que no es necesario reiniciar el servidor cada vez que se hace un cambio en el código.

El entorno de producción ***(production) e***l código no es recargado en cada petición al servidor. Otra diferencia importante es que los reportes o avisos de error en el código no son mostrados en pantalla.

El entorno de prueba ***(test)*** se utiliza exclusivamente para ejecutar las pruebas y comprobaciones en la aplicación.

Symfony

Symfony trabaja con 4 entornos y estos son: el de desarrollo, que es el utilizado por el desarrollador para añadir nuevas funciones, corregir errores, etc. El de pruebas que es utilizado para probar la aplicación. El staging utilizado por el cliente para poner a prueba la aplicación e informar de errores o características faltantes. El de producción es con el que el usuario final interactúa.

3.5.1.2. Resultados

Frameworks / Parámetros	Ruby on Rails	Symfony
• **Estructura del proyecto**	4	4
• **Manejo de errores**	3	3
• **Estandarización de nombres**	4	4
• **Entornos de desarrollo**	3	4
Total	**14**	**15**

Tabla III.II: Resultados parámetros globales.

$$Total_{RoR} = \sum_{i=1}^{4} VPi = 4 + 3 + 4 + 3 = \mathbf{14}$$

$$Total_{Symfony} = \sum_{i=1}^{4} VPi = 4 + 3 + 4 + 4 = \mathbf{15}$$

Interpretación:

El framework Ruby on Rails presenta una gama de herramientas de calidad pero no tiene un control de errores de sintaxis en tiempo real de programación lo que no es de mucha ayuda cuando se escribe código nuevo en un lenguaje desconocido.

Symfony, por su parte, tampoco muestra un manejo de errores completamente satisfactorio ya que presenta deficiencias en tiempo de programación, pero posee entornos de desarrollo del proyecto mucho más completos que RoR; éstos permiten una mayor interacción con el usuario final y mejoran considerablemente la calidad del producto final.
Los dos frameworks poseen características similares en lo referente a estructuración, manejo de errores y estandarización de nombres, la única diferencia se produce en el uso de entornos de desarrollo lo que hace a symfony ligeramente superior.

3.5.2. Modelo

El modelo está vinculado con la parte del proyecto en donde se maneja la lógica del negocio, esto es la interacción con la base de datos. A continuación se realizará un análisis de los parámetros más sobresalientes que brindan los frameworks basados en MVC en lo referente al *Modelo*.

3.5.2.1. Análisis de Parámetros

- **Compatibilidad con bases de datos**

Un parámetro muy importante a evaluar es la compatibilidad con las distintas bases de datos, debido a la variedad de gestores de bases de datos que existen en el mercado. A medida que un framework es compatible con una mayor cantidad de bases de datos, sus proyectos son más flexibles en cuanto a disposición del programador con tendencia a cierta base de datos o a lo que el proyecto requiera y a una posible migración a futuro. La valoración del puntaje está dada por:

- ✓ Número de bases de datos compatibles
- ✓ Gestión con las bases de datos

Ruby on Rails

Rails soporta la biblioteca SQLite por defecto. El acceso a la base de datos es totalmente abstracto desde el punto de vista del programador y Rails gestiona los accesos a la base de datos automáticamente. Sin embargo, debido a la diferente naturaleza y prestaciones de los SGBDRs (Sistema de Gestión de Base de Datos Relacional) el framework no puede garantizar la compatibilidad completa. Ruby on Rails soporta diferentes SGBDRs, incluyendo DB2, Firebird, Frontbase, MySQL, Openbase, Oracle, Postgres, SQLite, SQL Server ySybase databases.

Symfony

Symfony permite utilizar bases de datos de tipo SQLite, MySQL, PostgreSQL, Oracle y SQL Server. Cambiar de base de datos sólo requiere modificar la opción dsn, ya que el resto de la aplicación funciona igual de bien con cualquier sistema gestor de base de datos.

- **Manejo de conexión a una base de datos**

Se refiere a la administración eficaz de vinculación a un motor de base de datos específico. Dentro de éste parámetro se tomará en cuenta la facilidad que tiene para asociarse a una base de datos única. Los parámetros a evaluar son:

- ✓ Configuración de conexión
- ✓ Migración

Ruby on Rails

Para configurar la conexión a la base de datos, rails dispone del archivo “database.yml” ubicado en *config -> database.yml.* Este archivo permite configurar la conexión a tres bases de datos.

La configuración de este archivo es muy sencilla. Basta con indicar, en cada sección el nombre de cada una de las bases de datos que hemos creado junto a la dirección del servidor (host) donde está ubicada, el username y password del

usuario con acceso a la misma. Rails considera, por defecto, que la base de datos a la que nos vamos a conectar es MySQL. Permite cambiar el SGDB cuando el proyecto lo amerite.

Ilustración III.08: Configuración archivo database.yml en ruby on rails.

Symfony

Symfony gestiona la conexión a una base de datos por medio del archivo databases.yml, el cual almacena la toda la información concerniente a la base de datos como es: motor de gestión, nombre, usuario y contraseña. Dicha información permite crear el esquema respectivo con solo escribir una línea de comandos. Al manejar de esta manera la información de conexión facilita varias acciones eventuales que se pueden presentar a lo largo del desarrollo como conectarse a otra base de datos o cambiar el gestor.

- **Mapeo de una base datos a un modelo de objetos**

En este parámetro se evalúa la propiedad de los frameworks para tomar tablas de una base de datos y transformarlas en clases para poder manipular y gestionar los datos ya sean éstos almacenados o por almacenar. Esta característica es una de

las más importantes de los frameworks basados en MVC ya que facilitan el trabajo y ahorran mucho tiempo. Los parámetros a evaluar son:

- Gestores de mapeo
- Mapeo del proyecto a la base de datos
- Mapeo de la base de datos al proyecto

Ruby on Rails

Rails permite tener las funcionalidades básicas de administración de datos de un modelo en un controlador. Estas funcionalidades son las denominadas, CRUD (Create, Retrieve, Update, Delete), típicas de cualquier sistema transaccional.
A éste método, se le denomina "scaffolding" y sirve para el mapeo desde una base de datos al proyecto.

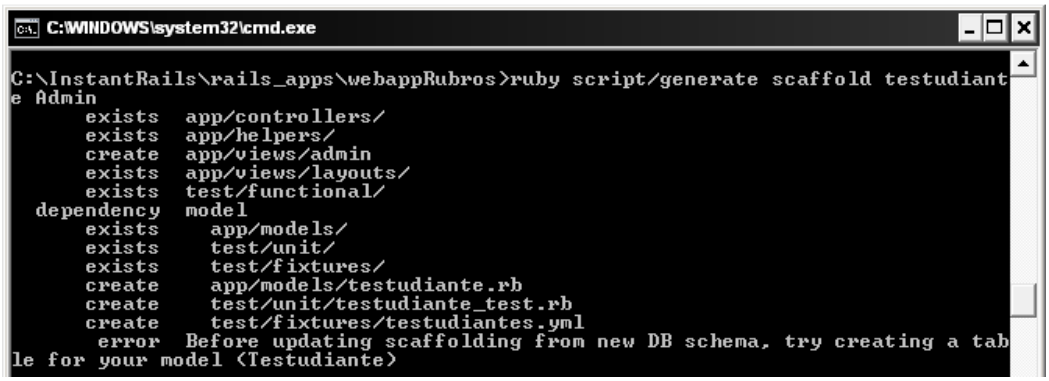


Ilustración III.09: Mapeo a objetos desde una BDD en RoR.

El *scaffold* toma las columnas de la tabla referenciada y crea los atributos para la clase que es el mismo nombre de la tabla en singular.
Rails También permite realizar el mapeo desde el proyecto a la base de datos.

Symfony

Para la realización de este mapeo Symfony cuenta con dos ORM que son Propel y Doctrine, los cuales facilitan todas las tareas que involucran el manejo de la base de datos como son: creación del esquema, el mismo que puede ser hecho a partir de una base de datos existente o de forma manual; construcción del modelo y generación de sql. Complementariamente puede generar formularios vinculados al

modelo y validadores. Además Propel y Doctrine permiten trabajar con una base de datos sin utilizar instrucciones SQL. Para crear, obtener, modificar o borrar datos de la base de datos, no es necesario escribir ni una sola sentencia SQL, ya que se puede trabajar directamente con objetos. El mapeo de una base de datos a objetos únicamente se realiza mediante una línea de código:

Symfony Doctrine: build –model

Symfony también puede realizar el mapeo desde el proyecto a una base de datos.

- **Entornos de desarrollo**

Los entornos de desarrollo para el modelo implican una base de datos para: desarrollo, pruebas denominada test y uso final de la aplicación como es la de producción. Estos entornos se han creado para mayor seguridad de los datos y la estructura de la base de datos, es una buena práctica cuando se llega a la etapa de mantenimiento. Los entornos a evaluar son:

- ✓ Desarrollo
- ✓ Pruebas
- ✓ Producción

Ruby on Rails

Ruby on Rails recomienda usar tres entornos separados para desarrollo, pruebas y producción.

- El entorno de desarrollo (*development*) optimiza la productividad del desarrollador. Las caches apenas operan, así que los cambios en el código de la aplicación se aprecian rápidamente, sin tener que recompilar o volver a desplegar nada. Sólo hay que recargar la página del navegador. Por esta razón Ruby on Rails es más rápido en el desarrollo que, por ejemplo, la plataforma Java 2.
- El entorno de pruebas (*test*) está optimizado para ejecutar pruebas unitarias, de integración y funcionales. Cada vez que se ejecuta una prueba, la base de datos se limpia de todos sus datos. Ruby on Rails se

encarga de poblar la base de datos con datos de prueba antes de cada test, a través de las *fixtures*.

- El entorno de producción (*production*) es donde se despliega la aplicación final. Este entorno está optimizado para rendimiento, lo que significa, por ejemplo, que las clases están en cache.

Symfony

Symfony trabaja únicamente con una base de datos para los diferentes entornos de la aplicación.

- **Interacción con la base de datos**

En esta parte se aprecia la gestión de la aplicación con la base de datos. Se evaluará:

- ✓ Bondades que presenta para realizar código *transact sql*
- ✓ Implementación de funciones de ingreso de datos en las tablas, modificación y eliminación de los mismos.
- ✓ Realización de consultas a la base de datos.

Ruby on Rails

En Ruby on Rails, las clases del Modelo son gestionadas por ActiveRecord. Por lo general, lo único que tiene que hacer el programador es heredar de la clase ActiveRecord::Base, y el programa averiguará automáticamente qué tabla usar y qué columnas tiene.

Los procesos SQL para inserción, modificación, eliminación y listado son prácticamente transparentes para el usuario ya que ruby on rails hace que las clases definidas en el modelo hereden de una clase base denominada *ActiveRecord.* Un nuevo proyecto en RoR tiene la posibilidad de alterar y sobrescribir los métodos de Active Record por lo que da una mayor flexibilidad.

Symfony

En lo que se refiere al ingreso, modificación, eliminación y listado, symfony genera automáticamente todas estas funciones para los diferentes módulos que el desarrollador desea crear. Es decir este código es totalmente transparente para el programador. Además genera querys para la realización de diferentes consultas mediante el uso de funciones y envío de parámetros ya que el ORM es el que se encarga de transformar estos datos en consultas SQL dependiendo de la base de datos con la que se trabaja.

3.5.2.2. Resultados

Frameworks / Parámetros	Ruby on Rails	Symfony
Compatibilidad con bases de datos	4	3
Manejo de conexión a una base de datos	4	4
Mapeo de una base datos a un modelo de objetos	3	4
Entornos de desarrollo	4	2
Interacción con la base de datos	4	4
Total	19	17

Tabla III.III: Resultados parámetros modelo.

$$Total_{RoR} = \sum_{i=1}^{4} VPi = 4 + 4 + 3 + 4 + 4 = \mathbf{19}$$

$$Total_{Symfony} = \sum_{i=1}^{4} VPi = 3 + 4 + 4 + 2 + 4 = \mathbf{17}$$

Interpretación

RoR posee en general una excelente compatibilidad con los distintos sistemas gestores de bases de datos pero por ser una herramienta de software libre tiene

una mayor apertura con bases de datos libres. Rails posee compatibilidad con mayor cantidad de gestores de base de datos incluido SQL Server. Symfony posee herramientas ORM como Propel y Doctrine que facilitan la interacción con las bases de datos y brinda al desarrollador la posibilidad de elegir la que se considere adecuada para su aplicación.

Las configuraciones que se deben realizar son sencillas e intuitivas y permite un manejo independiente y ordenado de toda la información concerniente a la base de datos.

En base a los resultados obtenidos en la valoración de los distintos parámetros se puede observar una ligera superioridad de rails en lo referente a la compatibilidad y posee un mejor manejo de entornos ya que tiene tres bases de datos: desarrollo, pruebas y producción. Controlando así la integridad de los datos de la aplicación. Symfony es superior en el mapeo de datos debido a las facilidades que ofrece sus ORMs.

3.5.3. Vista

La *vista* dentro del patrón de diseño MVC implica la parte visual del proyecto. Es lo que se va a mostrar al usuario como producto. La vista está encargada de interactuar con el usuario para obtener datos y solicitar resultados al controlador. Los frameworks basados en MVC presentan una característica especial, trabajan con plantillas para facilitar el trabajo.

3.5.3.1. Análisis de Parámetros

- **Generación de interfaces**

Este parámetro busca evaluar la capacidad que presentan los frameworks para:

- ✓ Generación de interfaces para las tareas más comunes de una aplicación web que trabaja con una base de datos (ingreso, modificación, eliminación y listado) de manera automática, mediante los datos obtenidos por el modelo.

Ruby on Rails

Las interfaces que RoR crea con el comando *scaffold* tienen un diseño simple. Utiliza *labels* con los nombres de las columnas de la tabla asociada. Se crean automáticamente las páginas para ingreso, modificación, eliminación, listado y sus respectivos enlaces para la navegabilidad.

Symfony

Ofrece la posibilidad de generar automáticamente las interfaces para el ingreso, modificación, eliminación y listado del módulo que el desarrollador desee, con una sola línea de comando, ahorrando de esta manera tiempo y esfuerzo. Siempre es necesario realizar pequeñas modificaciones en las interfaces generadas ya que estas son generadas con los nombres con el que se almacenan en la base de datos.

- **Manejo de estilos**

Los estilos se usan para personalizar varias páginas web sin tener que repetir el código para cada una de ellas. Además permite la administración independiente del formato de cada elemento de la página. Se evaluará la capacidad que tienen las herramientas para:

- ✓ Asociar páginas creadas con *estilos*.
- ✓ Flexibilidad

Ruby on Rails

Las hojas de estilo se guardan en public/stylesheet. Como el nombre de su directorio lo indica es de uso público para cualquier vista del proyecto. Se pueden definir más de un estilo y aplicarlo según sea seleccionado. Ruby tiene un estilo por defecto que puede ser personalizado o reemplazado. Las páginas de RoR tienen su propio código denominado Ruby Embebido que son básicamente fragmentos de código HTML con algo de código en Ruby.

Symfony

Este framewok cuenta con archivos independientes para el manejo de estilos, lo cual facilita la administración de estos. Al momento de crear el proyecto automáticamente se crea el directorio *web/css/* que va a ser el que aloje todas las hojas de estilos de la aplicación. Para relacionarlos con las diferentes html no es necesario hacerlos referencia en estas, ya que existe un archivo especializado para esta tarea, llamado view.yml que se maneja de manera independiente por los distintos módulos permitiendo aplicar un estilo diferente a cada uno de estos.

- **Uso de plantillas**

El uso de plantillas evaluará las bondades que prestan *Rails* y *Symfony* para la creación eficiente de un formato estándar utilizado en varias páginas de un sitio. Las características a calificar son:

- ✓ Uso de layouts
- ✓ Mantenimiento de la aplicación.

Ruby on Rails

Los formatos o *layouts* de Ruby on Rails sirven para rodear el contenido de las páginas con una cabecera y un pie de página. Se puede conseguir el mismo efecto insertando la cabecera y el pie en todas las páginas, pero esto atentaría contra el principio DRY de "No te repitas" (*Don't Repeat Yourself*), por el que hay que evitar la duplicación de código tanto como sea posible. Cuando se presenta un requerimiento diferente se puede realizarlo de manera ágil.

Symfony

Cuenta con un archivo llamado layout.php que se encarga de almacenar todas las plantillas globales para la aplicación con la finalidad de no repetir código. Puede almacenar los encabezados, pies de páginas, menús y lo único que cambiará dinámicamente será el contenido de la página. Existe un layout para cada módulo de la aplicación por lo que es muy fácil su configuración en caso de que no se

quieran utilizar las mimas plantillas globalmente. El mantenimiento de una aplicación es rápido y eficaz.

- **Tamaño de archivo para una misma función**

El tamaño de un archivo es muy importante en una aplicación web debido a que el proceso de transferencia de datos es complejo y mientras es menos extenso es mucho mejor y se transmite con menos errores.

- ✓ Espacio en disco
- ✓ Líneas de código

Ruby on Rails

El tamaño de archivo en rails es pequeño por lo general menos de 1.5 Kb. Los archivos de las vistas son de tamaño mínimo porque se ubican dentro de un *layout* en un espacio determinado por lo que no tiene que volver a escribir el código de encabezado, pie de página, enlaces de navegabilidad, etc.

Symfony

El tamaño en disco de los archivos en Symfony son relativamente pequeños debido a que utilizan Layouts para reutilizar el código creando un diseño global para todas las vistas que están referenciadas a él. También ofrece la opción de modificar el marco heredado.

- **Manejo de helpers**

Para evaluar el manejo de *helpers* se debe tener claro el concepto del mismo.
Un *helper* en desarrollo en general es un método de conveniencia que se crea básicamente para reducir la complejidad de un método más grande. Es una forma conveniente de separar un proceso grande en varios procesos más pequeños, que hacen el código más legible y que, puede ser que se reutilicen en otras partes. Los

helpers han sido creados con el objetivo de mejorar la productividad en el desarrollo. El aspecto a evaluar será:

- ✓ Manejo de helpers

Ruby on Rails

Un módulo helper es un módulo Ruby que contiene varios métodos de: validación, métodos que devuelven un resultado procesos, y además por defecto en Rails por cada controller se define un helper que se incluye automáticamente en todas las vistas de ese controlador.

Symfony

Las versiones actuales de Symfony han minimizado el uso de helpers sin embargo todavía son utilizadas para ciertas funciones con la finalidad de ahorrar código y esfuerzo al momento de programar. Un ejemplo helper es *include_stylesheets()*, utilizado para incluir estilos a una página HTLM ahorrando el trabajo repetitivo de referenciación.

3.5.3.2. Resultados

Frameworks / Parámetros	Ruby on Rails	Symfony
Generación de interfaces	3	3
Manejo de estilos	3	4
Uso de plantillas	4	4
Tamaño de archivo para una misma función	4	4
Manejo de helpers	4	4
Total	18	19

Tabla III.IV: Resultados parámetros vista

$$Total_{RoR} = \sum_{i=1}^{4} VPi = 3 + 3 + 4 + 4 + 4 = \mathbf{18}$$

$$Total_{Symfony} = \sum_{i=1}^{4} VPi = 3 + 4 + 4 + 4 + 4 = \mathbf{19}$$

Interpretación:

El framework rails tiene potentes herramientas para la creación automática de las vistas correspondientes a las operaciones básicas (CRUD) de un sistema que interactúa con una base de datos. Además posee código propio de ruby en sus interfaces como es el ruby embebido. Las interfaces tienen diseños simples y no muy elaborados.

Symfony al igual que Rails posee herramientas para la creación automática de vistas. Permite el manejo de estilos, plantillas y helpers indispensables para el refinamiento de las interfaces.

En base a la valoración de los parámetros se concluye que los dos frameworks poseen características similares para el manejo de vistas, plantillas y helpers. Symfony tiene una mayor flexibilidad con el manejo de estilos lo que lo hace tener mayor valoración que Rails.

3.5.4. Controlador

El *controlador* dentro del patrón de diseño MVC implica la gama de procesos que se deben ejecutar cuando el usuario realiza una acción desde la vista y solicita información que se puede obtener desde la base de datos gestionados por el modelo. Es decir, el *controlador* es el intermediario entre la *vista* y el *modelo*. También es conocido como enrutador.

Para evaluar las propiedades de un framework en la parte del controlador se han definido varios parámetros que a continuación se detallan.

3.5.4.1. Análisis de Parámetros

- **Administración de navegabilidad**

La navegabilidad abarca los diferentes métodos que los frameworks utilizan para manejar el enrutamiento entre páginas. En algunos casos el enrutamiento es un mecanismo de seguridad para el envío de parámetros en la url. Se analizará los siguientes parámetros:

- ✓ Enlaces entre páginas
- ✓ Seguridad en el envío de parámetros a través de la URL.

Ruby on Rails

En rails la expresión común para realizar un re direccionamiento es *redirect_to.* A partir de la versión 2.1.2 de Rails, se realiza un filtrado por defecto en la cabecera Location cuando se utiliza el método redirect_to. Brinda seguridad básica en la transferencia de parámetros en la dirección URL que evitan ataques como el physing.

Symfony

Maneja todo un framework especializado en el enrutamiento de URL permitiendo a las direcciones ser más amigables y que representen una descripción de la tarea que se encuentra realizando. Además brinda seguridad al no permitir mostrar parámetros que son enviados de una página a otra por medio de la url, y valida información de los formularios. El manejo del enrutamiento da mayor seguridad en los enlaces entre páginas.

- **Uso de parámetros**

El uso de parámetros en las declaraciones de métodos es una manera eficaz de enviar datos entre ellos para poder trabajar y devolver el resultado de un valor o

simplemente utilizar esos datos en determinados procesos. Se evaluará esta característica:

- ✓ Tipo de parámetros

Ruby on Rails

Ruby no tiene parámetros nombrados. Sin embargo, pueden ser emulados mediante el uso de symbols y hashes. Ruby on Rails, entre otros, usa esto a discreción.

Debido a la naturaleza simple y dinámica de Ruby, se puede pasar nombres de clases como parámetro, añadir métodos a cualquier clase cuando se necesite (dependiendo de cualquier condición o incluso dentro de bucles), hasta incluso a las clases básicas.

Symfony

Este framework debido a que se fundamenta en php no tiene tipos de datos definidos por lo que facilita el paso de parámetros ya que sólo tiene que nombrarlos más no especificar a qué tipo pertenecen.

- **Abstracción de código SQL**

La abstracción del código sql facilita el trabajo al desarrollador con la creación propia de código sql y su representación en simples métodos que se pueden utilizar cada vez que se los nombre. Los parámetros a evaluar son:

- ✓ Versatilidad de métodos
- ✓ Redefinir métodos

Ruby on Rails

Ruby on Rails tiene una clase llamada Active Record de donde hereda las funciones de interacción con la base de datos, abstrae el código sql y permite la creación de sql personalizado. Active record tiene ya especificados métodos muy útiles de búsqueda, creación, entre otros que facilitan la tarea del programador y

hacen menos tedioso su trabajo. Rails permite redefinir los métodos de una clase heredada.

Symfony

Gracias al archivo de descripción de la base de datos schema.yml, Symfony puede utilizar algunas de las tareas de Doctrine para generar los comandos SQL necesarios para crear tablas, realizar consultas y crear procesos para la base de datos. El ORM es el encargado de la abstracción de la base de datos y ayuda a la creación del modelo en general. Symfony es capaz de redefinir un método heredado añadiendo características personalizadas.

- **Funciones de validación de datos**

La validación de datos es una de las áreas más importantes en seguridad informática, especialmente en sistemas web. Los procesos que se realizan son: verificar, controlar y/o filtrar cada una de las entradas que provienen desde el exterior del sistema. Las características que se analizarán son:

- ✓ Autovalidación
- ✓ Mensajes de error de ingreso
- ✓ Redefinir tipo de validación

Ruby on Rails

Ruby on rails puede validar tipos de datos que vienen definidos desde el mapeo de la base de datos automáticamente. Además, se puede definir validaciones según la necesidad del proyecto. Rails tiene una manera particular de presentar sus mensajes de error, por defecto muestra un recuadro de color rojo alrededor de la casilla en donde se produjo el error y en la parte superior un cuadro especificando el motivo.

Ilustración III.010: Mensaje de error en validación Ruby on Rails.

Symfony

Symfony incluye un mecanismo específico de validación de formularios realizado mediante archivos YAML, en vez de utilizar código PHP en la acción. Como algunas de las reglas de validación son tan comunes que aparecen en todos los formularios, Symfony ha creado unos validadores que encapsulan todo el código PHP necesario para realizarlos. Un validador es una clase que proporciona un método llamado execute(). El método requiere de un parámetro que es el valor del campo de formulario y devuelve true si el valor es válido y false en otro caso. Symfony permite redefinir la validación del tipo de un tipo de dato al mismo tiempo que genera mensajes en caso de un ingreso erróneo, los mismos que pueden ser personalizados.

- **Manejo de sesiones**

Una aplicación web por mediana o pequeña que fuera necesita el manejo de sesiones. Las sesiones permiten almacenar y consultar información sobre un visitante sin necesidad de estar enviándola a través de formularios. Los parámetros a considerarse son:

- ✓ Autenticación
- ✓ Manejo de credenciales

Ruby on Rails

Rails utiliza una tabla hash (session) para mantener las sesiones de diferentes objetos, de ésta manera cualquier par de llaves y valores que se almacenen durante el procesamiento de una petición estarán disponibles durante las subsecuentes peticiones desde el mismo browser.

Symfony

Symfony maneja automáticamente las sesiones del usuario y es capaz de almacenar datos de forma persistente entre peticiones. Además Symfony proporciona todas las herramientas necesarias para el manejo de credenciales y la reutilización de plugins que facilitan el proceso de autenticación. Este framework divide el proyecto en aplicaciones con funciones administrativas y de usuario.

3.5.4.2. Resultados

Frameworks / Parámetros	Ruby on Rails	Symfony
Administración de navegabilidad	3	4
Uso de parámetros	4	4
Abstracción de código SQL	4	4
Funciones de validación de datos	4	4
Manejo de sesiones	3	4
Total	**18**	**20**

Tabla III.V: Resultados parámetros controlador

$$Total_{RoR} = \sum_{i=1}^{4} VPi = 3 + 4 + 4 + 4 + 3 = \mathbf{18}$$

$$Total_{Symfony} = \sum_{i=1}^{4} VPi = 4 + 4 + 4 + 4 + 4 = \mathbf{20}$$

Interpretación:

Symfony y Ruby on Rails presentan resultados aparentemente iguales. Symfony tiene sus métodos que brindan herramientas muy eficientes y que cumplen además con una buena funcionalidad; así como RoR maneja adecuadamente sus

métodos para ofrecer un catálogo muy amplio de funcionalidades las que pueden facilitar el desarrollo de una aplicación. Los dos frameworks tienen una buena representación de lo que corresponde al Controlador de MVC, administran correctamente las peticiones desde la vista y efectúan las mismas interactuando satisfactoriamente con el modelo. Symfony maneja de mejor manera la navegabilidad y las sesiones porque tiene mayor seguridad en sus procesos.

3.5.5. Resultado de los análisis efectuados

Después de haber comparado a Symfony y a Ruby on Rails se ha observado que ambos presentan características muy sobresalientes en base a MVC. Son considerados los mejores exponentes de este patrón de diseño y brindan facilidades muy destacables para la creación de proyectos web. Symfony tiene una ligera ventaja sobre RoR debido a que su lenguaje origen, php, es uno de los preferidos por los desarrolladores de páginas web y es cada vez mejorado. Por esta razón será elegido Symfony para realizar la evaluación en base a la productividad. Además, para realizar la evaluación se comparará con Php ya que será el mejor ejemplo del desarrollo de un módulo usando Php puro y Symfony que es un framework basado en Php y desarrollado en base a MVC.

CAPÍTULO IV

Parámetros de Evaluación de la Productividad en el desarrollo de software con el uso de lenguajes de programación

En el terreno de las metodologías de desarrollo de software, se aprecia una necesaria mejora en la puesta en práctica de dichas metodologías de desarrollo, así como la flexibilización de éstas para potenciar la productividad de las mismas sin renunciar a la calidad.

Por esta razón se hace cada vez más necesario disponer de herramientas efectivas para aumentar la productividad, no solo desde un punto de vista teórico sino especialmente en la puesta en práctica de dichas metodologías, consiguiendo que su despliegue impacte positivamente en el negocio de la empresa. La mejora de la efectividad y la productividad en el desarrollo de software está vinculada a la utilización de buenas prácticas de Ingeniería de Software. En la actualidad es indiscutible que el uso de una metodología apropiada es un factor clave para el éxito de cualquier esfuerzo de ingeniería y también debería serlo en la ingeniería del software. La ingeniería de software, por su relativa juventud como

disciplina y por la altísima variabilidad de los productos que gestiona, pocas organizaciones que desarrollen software utilizan metodologías de forma sistemática, aunque esta tendencia está cambiando día a día.

4.1. Productividad en el desarrollo de software

La ***productividad*** como concepto es concebida de una manera amplia, ésta es una definición de sistemas ya que puede ampliarse a diversas entidades, que varían desde un individuo o una máquina hasta una compañía, industria o una economía nacional. Así mismo, una mayor productividad proviene de tres fuentes primarias: tecnología, destreza administrativa y esfuerzo humano. Lo que significa la integración efectiva de la tecnología, la estructura, los procesos administrativos y el personal.
La productividad es el cociente entre los productos obtenidos y los recursos empleados. A simple vista sería como salidas/entradas y en cierta forma tiende a pensarse solamente en eficiencia de los recursos. Se considera que guarda relación con la efectividad debido a que los procesos deben generar productos parciales que cumplan con los objetivos trazados (efectividad del producto y proceso). Los recursos se deben emplear eficientemente a fin de cumplir con la planificación pautada para los procesos (eficiencia del producto y del proceso).

Los modelos y las metodologías basadas en modelos son la herramienta para abstraer de los detalles irrelevantes en un determinado contexto y poder razonar sobre el sistema a construir. Los modelos están demostrando ser una herramienta de productividad, acercando los modelos a los expertos del dominio de aplicación. Este enfoque permite separar los modelos que describen la solución al problema en términos de negocio, de los modelos que describen la implementación en términos de la plataforma software. Esta arquitectura de solución separa los aspectos del negocio de la tecnología de implementación facilitando que ambos evolucionen independientemente uno de otro y posibilitando verdaderas factorías

de software estructuradas por dominio de aplicación y por tecnología de implementación.

4.2. Evaluación de la productividad en el desarrollo de software

Para realizar el estudio de la productividad se ha tomado como referencia el *manual de FIM – productividad (Fondo para la Investigación y Mejoramiento de la productividad)* adaptado al desarrollo de software y los parámetros seleccionados de éste en la investigación **CALIDAD SISTÉMICA Y PRODUCTIVIDAD EN EL DESARROLLO DE SISTEMAS DE SOFTWARE** desarrollada por Edumilis M. MÉNDEZ, María A. PÉREZ, Anna C. GRIMÁN, Luis E. MENDOZA del Departamento de Procesos y Sistemas de la Universidad Simón Bolívar de Venezuela. Esta investigación se ha seleccionado como guía principal para analizar los parámetros de evaluación de la productividad. En aquella investigación se comprobó la efectividad de la adaptación del Manual de FIM - Productividad para el sector software ya que en primera instancia el manual fue aplicado para medir la productividad en la industria manufacturera.

Este instrumento fue de gran utilidad para esta investigación debido a que es un modelo probado, documentado y de fácil acceso; y además, se ha adaptado a los requerimientos que se han presentado.

Se analizará a la Productividad mediante tres indicadores: **Productividad Organizacional (**Perspectiva Financiera)**, Productividad del Proyecto** (Perspectiva Interna)**, y Productividad Individual** (Perspectiva de Aprendizaje y Crecimiento).

La **Productividad Organizacional** mejora la competitividad y el valor del accionista, lo cual le permite incrementar su inversión en la organización. La competitividad influye en la cuota del mercado y éste incide en el crecimiento, permitiendo de esta forma que el accionista aumente su inversión fija**,** y se

extienda el acceso a las tecnologías y el entrenamiento en prácticas base de la productividad al ofrecerle un nuevo entrenamiento al personal como “feedback” del proceso de aprendizaje. La inversión en nuevas tecnologías genera una mayor satisfacción de los empleados que incide en la calidad del proceso y en la **Productividad Individual**.

Area a evaluar	Sub-área
I. Gerencia	I.1. Gerencia y Entorno (Planificación Estratégica). I.2. Dirección y Control.
II. Organización, Información y Funciones	II.1 Estructura Funcional. II.2. Sistemas de Información. II.3. Normalización.
III. Recursos Humanos	III.1. Políticas. III.2. Sistemas de Administración del Personal. III.3. Políticas de Motivación.
IV. Planificación, Programación y Control de producción	IV.1. Planificación. IV.2. Programación. IV.3. Control.
V. Distribución en planta, Almacenamiento y Manejo de materiales	V.1. Distribución en planta. V.2. Almacenes. V.3. Manejo de Materiales.
VI. Suministros	VI.1. Política. VI.2. Planificación y Programación. VI.3. Control.
VII. Investigación y Desarrollo	VII.1. Diseño del Producto. VII.2. Diseño del Proceso. VII.3. Métodos de trabajo.
VIII. Mantenimiento	VIII.1. Políticas y Organización. VIII.2. Planificación y Programación. VIII.3. Control.
IX. Finanzas	IX.1. Política Financiera. IX.2. Presupuestos y Flujo de Caja. IX.3. Contabilidad de Costos y General.
X. Mercadeo	X.1. Políticas y Estrategias. X.2. Ejecución y Control.
XI. Ventas	XI.1. Políticas y Estrategias. XI.2. Ejecución y Control.
XII. Sistema de Control de Calidad	XII.1. Organización del Sistema. XII.2. Mediciones y Sistemas de Información. XII.3. Prevenciones y Correcciones.
XIII. Higiene y Seguridad Industrial	XIII.1. Política y Organización. XIII.2. Planificación y Programación. XIII.3. Control.

Tabla IV.VI: Áreas de evaluación en el manual FIM - productividad.

Para el estudio de la productividad en el desarrollo de software se ha tomado en cuenta únicamente del manual de FIM-Productivida las áreas de *Investigación y desarrollo* y la de *Sistema de control de calidad*. Para complementar la

investigación y evaluación de parámetros se tomó en cuenta varios indicadores y variables. Cabe recalcar que la evaluación para medir la productividad está en función de la *productividad individual*.

4.3. Parámetros para evaluar la productividad en el desarrollo de software

Para este estudio, no se toman en cuenta las características de la dimensión del producto debido a que éstas corresponden al sistema de software ya operativo y la investigación se hace a nivel del proceso de desarrollo del sistema.

4.3.1. Tiempo de desarrollo de una aplicación

4.3.1.1. Codificación

Existen estudios de productividad basados en las líneas de código. Para empezar, está todo lo referente a las diferencias entre los lenguajes, los distintos estilos de contar líneas, y las diferencias por las convenciones de codificación. Pero incluso si se va a usar un estándar consistente para contar líneas en los programas de un mismo lenguaje, que está todo auto-formateado a un único estilo, incluso entonces las líneas de código no miden la salida correctamente.

Se conoce que se puede codificar lo mismo con enormes variaciones en las líneas de código; de hecho un código que está bien diseñado y programado va a ser más corto porque elimina la duplicación. La programación basada en copiar-y-pegar lleva a enormes cantidades de Lineas De Código (LOC - Lines Of Code), y el diseño pobre genera duplicación.

Las LOC sirven muy bien para sugerir el tamaño de un sistema. Se puede asegurar que un sistema con 100 KLOC es más grande que un sistema con 10 KLOC. Pero si se escribe un sistema de 100 KLOC en un año y el mismo sistema en 10 KLOC en el mismo tiempo, esto no significa que el segundo desarrollador es

más productivo. Se debería concluir que la productividad fue aproximadamente la misma, y que el primer sistema tiene un diseño inferior.

Existen lenguajes que crean automáticamente los nombres de los objetos y tipos de datos que no están sujetos a estándares de desarrollo lo cual puede generar inconvenientes a la hora de hacer cambios o cuando la codificación pasa de un programador a otro. Un estándar de codificación es el establecimiento de un único estilo de codificación (convención) para todo el proyecto o la organización y los programadores de hacer estrictamente lo siguen. Esto también se conoce como estilo de programación o codificación de los convenios. Por lo general, los estilos de codificación son diferentes de un lenguaje a otro.

Un lenguaje puede ayudar a la productividad del desarrollo si el nombre de sus tipos de datos (diccionario de datos) es de fácil comprensión, lo que lo hace intuitivo para un desarrollador que está en proceso de aprendizaje de una herramienta. El nombre debe indicar claramente el propósito de esa clase, etc.

Cuando un lenguaje es sensible a mayúsculas y minúsculas se puede utilizar un estándar tomado de las mejores prácticas de Nagaraja (jefe de proyectos en una empresa de IT en la India), en donde todas las variables deben ser declaradas en letra minúscula y todas las constantes en letra mayúscula.

El uso de comentarios y la facilidad que una herramienta brinda para éstos al momento de crear una aplicación es muy importante ya que mientras más fácil de entender el código, es menos propenso a equivocaciones cuando se rota el personal en un proyecto o al mantenimiento de una aplicación.

- **Mejores prácticas para la codificación**

Fácil de entender el código y los tipos de datos de otros, debido a los diferentes estilos de programación del programador, sino que es a veces difícil de leer fácilmente la codificación realizada por otra persona. Si todos los programadores comienzan a usar la misma convención este problema se puede eliminar hasta cierta medida.

Fácil de utilizar y depurar el programa de codificación estándar, si se sigue, que será fácil para depurar el programa debido a dos factores, en primer lugar, algunos errores pueden eliminarse totalmente las cuales son causadas debido a la mala asignación de nombres y sangría de código, y en segundo lugar, ser fáciles de depurar programas de programadores compañeros. Es importante tener en cuenta la segunda razón, cuando se encuentra atascado en un punto, se debe intentar solucionar el problema en equipo. Una segunda persona puede tratar de entender su lógica y buscar más a fondo que usted.

La disminución de la complejidad del programa puede llegar a ser complejo, no sólo debido a la lógica del programa, sino también debido a la genialidad de codificación realizada por el promotor. Si la sangría y mejor convención de nomenclatura se sigue, este problema será menor.

Dotación de personal, gente nueva puede ser fácilmente añadida al proyecto porque no va a haber ningún problema para los nuevos para empezar a producir. La única necesidad es entender los estándares de codificación.

4.3.1.2. Manejo de errores

Debido a que el tiempo de programación siempre está dentro de una planificación, y por lo general existen desfases en sus fechas topes, la mayoría del tiempo se intenta acortar tiempos incrementando la presión en los desarrolladores, esto provoca una mayor densidad de errores en el producto, y eleva los tiempos de depuración y hace compleja la posibilidad de mantenimiento.
La búsqueda de atajos en los procesos de requisitos incrementará el coste y las agendas de desarrollo, y si anteponemos las agendas a la calidad no conseguiremos ni uno ni otro, etc.

Un lenguaje de desarrollo debe ofrecer como servicio integrado métodos para manejar correctamente los errores que se pueden producir cuando se realiza la codificación. En la industria del desarrollo de software se considera que lo normal es tener 15 errores de programación por cada mil líneas de programa. Según

estudios realizados en el Instituto Nacional de Estándares y Tecnología de los Estados Unidos estima que el empleo de software con errores en ese país supone un costo anual cercano a los 60 mil millones de dólares anuales lo que representa pérdidas extraordinarios de esfuerzo humano y tiempo en la corrección y depuración de errores por lo que un lenguaje preparado con herramientas para corregir errores es fundamental para mejorar la productividad en el desarrollo del software.

Un método muy útil, que poseen algunos IDEs, cuando se codifica es el subrayado cuando hay errores de sintaxis. Otro método usado frecuentemente es la depuración paso a paso, muchos errores se solucionan observando el comportamiento de cada procedimiento.

Si un lenguaje está basado en un patrón de diseño y da la posibilidad de crear una estructura que orienta al proyecto a seguir el patrón, podría reducir el número de errores según Sergio Lujan Mora de la Universidad de Alicante en el paper “Un enfoque para la enseñanza de la depuración de errores en las asignaturas de programación” publicado en el 2003.

- **Métodos de manejo y prevención de errores**

Retornar un valor neutral

A veces el regresar un valor inofensivo o inocuo es la mejor respuesta a los datos basura. De este modo se puede continuar operando sin problemas. Un cálculo numérico, por ejemplo, puede retornar cero, una operación de cadena podría retornar la cadena vacía, y una operación de punteros podría retornar un puntero vacío.

Sustituir el dato basura por el dato válido previo

Al procesar flujos de datos, como la lectura de registros de un fichero de base de datos o información proveniente de una línea de comunicación, y se encuentra un dato basura, puede optarse simplemente por ignorarlo y leer o esperar el siguiente registro o dato válido. Si por ejemplo se está tomando valores de un sensor a

razón de 100 veces por segundo y si en alguna no se obtiene una lectura válida, el software de lectura del sensor podría esperar una centésima de segundo y tomar la siguiente lectura.

Guarda un registro del error en un archivo

Esta técnica se utiliza en los ambientes que priorizan la robustez, permitiendo continuar la ejecución del programa aun cuando se reciban datos basura. Los datos basura se ignoran, pero se guarda un mensaje de error con información suficiente sobre el dato basura y el problema para ayudar en su diagnóstico y corrección. Este mecanismo puede usarse en combinación con cualquiera de las tres técnicas anteriores.

Regresa un código de error

Se puede decidir que sólo ciertos componentes de un programa manejarán lo errores detectados, y los demás sólo los reportarán. Estos últimos no están en capacidad de manejar los errores, pero confían que rutinas de niveles superiores de la jerarquía sí pueden hacerlo.

El modo de comunicar al resto del sistema que ha ocurrido un error puede ser uno de los siguientes:

- Establecer el valor de una variable de estado
- Retornar el estado como el valor de la función
- Lanzar una excepción mediante los mecanismos apropiados del lenguaje

Para que esta técnica funcione, todos los códigos de error devueltos deben ser chequeados siempre.

Se podría centralizar el manejo de errores en una única rutina que sea global a todo el programa, lo cual puede facilitar la depuración. Sin embargo, existe el problema de que todos los componentes deben saber de ella y estarán demasiado acoplados (conectados, articulados, enlazados) a la rutina de manejo de errores.

Por tanto, se dificultaría la reutilización de código en otros programas al tener que arrastrar también la rutina de manejo de errores junto con el código a reutilizar, debido a que probablemente el código la necesite.

Manejar el error de la forma que mejor funcione localmente

Este método es lo opuesto al anterior, y tiene como ventaja la reducción de la sobrecarga que implican las frecuentes llamadas a la rutina global de manejo de errores.

Esta aproximación brinda gran flexibilidad a los programadores de un proyecto de desarrollo, pero se deben especificar los mecanismos de manejo de errores a usar. De lo contrario, la variedad de métodos (tradicionales y particulares de cada programador) dificultará la depuración y el mantenimiento, sin mencionar las distintas formas y estilos de formato para comunicar los errores detectados: mensajes de error en ventanas emergentes, en la última línea u otras áreas específicas de la pantalla, en archivos bitácoras, etc.

Dicho de otro modo, la utilización de esta técnica atenta contra la consistencia de la interfaz del programa, lo cual es vital para la aceptación del software y su aprendizaje por parte del usuario. La seguridad también es puesta en riesgo al tener diferentes formas de redacción, lo cual puede exponer demasiada información sobre el software que puede ser utilizado para atacarlo. Por último, si se debe o se desea separar la interfaz de usuario del resto de código, este método de manejo de errores será un enorme obstáculo pues ambos componentes estarán muy mezclados.

4.3.1.3. Tiempo de respuesta

Es fundamental en el desarrollo de un proyecto escoger un lenguaje que se apegue a nuestras necesidades. Hay lenguajes de programación que funcionan solamente sobre ciertas plataformas y que en otras se ejecutan con un rendimiento bajo. Por lo que se debe analizar al inicio del proyecto todas o la

mayoría de las necesidades para elegir de manera apropiada la herramienta de desarrollo.
Según el tamaño de la empresa se debe escoger el lenguaje y la plataforma adecuada para que el producto final pueda ejecutarse de la mejor manera. Ruby on Rails está diseñado para dar soluciones a pequeñas y medianas empresas como es el caso de una entidad educativa.

4.3.1.4. Reutilización de líneas de código (Herencia)

Si el código no es complejo y basado en estándares, se puede fácilmente volver a utilizar en algún otro proyecto. Esta reutilización puede ser directa o con alguna modificación. En cualquiera de estos dos casos, las normas de codificación son realmente rentables e incrementan la productividad.

Hay lenguajes de desarrollo que brindan la posibilidad de manejar el código reutilizado y adaptarlo al nuevo proyecto en el que trabajemos. Por ejemplo cuando se cambia un nombre a una variable automáticamente se activa una herramienta para hacer una modificación en cascada de todas las variables que tenían ese nombre y así reducir: el tiempo para adecuar nuestro código reutilizado y el margen de errores.

Cuando se reutiliza el código que antes se desarrollo y no se toma el suficiente tiempo para estudiarlo completamente se corre el riesgo de duplicar líneas de código por lo que el desarrollador debe ser cuidadoso. Sería muy conveniente que en el lenguaje que usemos exista una herramienta de análisis de código que nos generara un error en eso casos.

4.3.2. Facilidades de un lenguaje

4.3.2.1. Herramientas

Una herramienta es cualquier dispositivo que, cuando se emplea en forma apropiada, mejora el desempeño de una tarea.

Pressman caracteriza la Ingeniería de Software como “una tecnología multicapa”, en la Ilustración IV.11.

Ilustración IV.11: Capas de la Ingeniería de Software.

Dichas capas se describen a continuación:

- Cualquier disciplina de ingeniería (incluida la ingeniería del software) debe descansar sobre un esfuerzo de organización de **calidad**. La gestión total de la calidad y las filosofías similares fomentan una cultura continua de mejoras de procesos que conduce al desarrollo de enfoques cada vez más robustos para la ingeniería del software.
- El fundamento de la ingeniería de software es la **capa proceso**. El proceso define un marco de trabajo para un conjunto de áreas clave, las cuales forman la base del control de gestión de proyectos de software y establecen el contexto en el cual: se aplican los métodos técnicos, se producen resultados de trabajo, se establecen hitos, se asegura la calidad y el cambio se gestiona adecuadamente.
- Los **métodos** de la ingeniería de software indican cómo construir técnicamente el software. Los métodos abarcan una gran gama de tareas que incluyen análisis de requisitos, diseño, construcción de programas, pruebas y mantenimiento. Estos métodos dependen de un conjunto de principios básicos que gobiernan cada área de la tecnología e incluyen actividades de modelado y otras técnicas descriptivas.
- Las **herramientas** de la ingeniería del software proporcionan un soporte automático o semi-automático para el proceso y los métodos, a estas herramientas se les llama herramientas CASE (*Computer-Aided Software Engineering*).

Las herramientas de desarrollo brindan una potente solución para la automatización de procesos. Por esta razón, mientras la calidad de las herramientas que empleemos en nuestro proyecto sea mejor, éstas aumentarán la productividad de nuestro software. El uso adecuado de las herramientas disminuye el número de errores y el tiempo de codificación.

Un lenguaje de desarrollo de software es más productivo cuando éste puede relacionarse con herramientas CASE y generar automáticamente en base a éstas clases, objetos, etc.

También es importante mencionar que hay herramientas que generan reportes en base a los comentarios utilizados por los desarrolladores.

- **Importancia de las herramientas en el desarrollo de sistemas**

  • Mejora la productividad del analista
  • Mejora la eficiencia
  • Mejora la calidad del sistema de información

4.3.2.2. Entorno

El entorno del desarrollo de software se refiere en sí al ambiente del lenguaje que vamos a utilizar. Si entorno del lenguaje es amigable, intuitivo y de fácil comprensión va a ser una herramienta más productiva para el desarrollador debido a la disposición de su manejo.
Debe tener las herramientas adecuadas para tener un ambiente productivo. Mientras más herramientas de calidad posean, va a ofrecer la posibilidad de crear soluciones mucho más robustas y de mayor alcance. Si el ambiente es fácil de entender y el entorno es fácil de manipular, será menor el tiempo de codificación.
El objetivo de un entorno amigable es que éste sea muy práctico al momento de desarrollar una solución empresarial.

4.3.2.3. Generación de código

La codificación es un proceso un poco tedioso cuando se vuelve a programar funciones ya definidas con anterioridad, por este motivo, hay herramientas CASE que generan código a partir de diagramas creados para el proyecto.
Hay que tener precaución al momento de generar códigos automáticamente ya que podemos caer en redundancias o dejar códigos basura que no desempeñan ninguna función en nuestros proyectos y que aumentan el tamaño de nuestra aplicación innecesariamente.

4.3.2.4. Estructura de un nuevo proyecto

El resultado de estudios de ingeniería de software ha dado impulso al desarrollo de sistemas. Esto es con la creación de arquitecturas y patrones de diseño los cuales se pueden adaptar según la necesidad de nuestros proyectos. De ahí la importancia de conocerlos y saber cual se debe aplicar.
En nuestro caso, los lenguajes se deben basar en el patrón de diseño Modelo Vista Controlador y deben tener herramientas para que el programador se acople a este tipo de patrón para crear soluciones empresariales. En éste tipo de lenguajes cuando se crea un proyecto nuevo, éstos automáticamente generan una estructura para basarse en el patrón MVC, ya sea en la distribución en carpetas de la codificación o en la misma codificación.
Estos lenguajes deben tener una solución para interactuar también con herramientas de modelado como son las herramientas CASE y ser flexibles en la generación de nuevas estructuras, claro tomando en cuenta el patrón de diseño en el que nos basemos.

4.3.3. Interacción con bases de datos

La mayoría de los sistemas de información ya sea, implantados en empresas grandes o pequeñas, utilizan una base de datos que pueden abarcar varias aplicaciones, por esta razón estos sistemas utilizan un administrador de base de

datos, en este caso el diseñador no construye la base de datos sino que consulta a su administrador para ponerse de acuerdo en el uso de esta en el sistema.

4.3.3.1. Integración de Bases de Datos en la Web

Para realizar una requisición de acceso desde el Web hasta una base de datos se necesita: un browser del Web, un Servidor Web y un software de procesamiento (aplicación CGI), el cual es el programa que es llamado directamente desde un documento HTML en el cliente. Dicho programa lee la entrada de datos desde que provienen del cliente y toma cierta información de variables de ambiente. El método usado para el paso de datos está determinado por la llamada CGI.

Una vez se reciben los datos de entrada (sentencias SQL o piezas de ellas), el software de procesamiento los prepara para enviarlos a la interfaz en forma de SQL, y luego ésta procesa los resultados que se extraen de la base de datos.

La interfaz contiene las especificaciones de la base de datos necesarias para traducir las solicitudes enviadas desde el cliente, a un formato que sea reconocido por dicha base. Además, contiene toda la información, estructuras, variables y llamadas a funciones, necesarias para comunicarse con la base de datos.

El software de acceso usualmente es el software distribuido con la base de datos, el cual permite el acceso a la misma, a través de solicitudes con formato. Luego, el software de acceso recibe los resultados de la base de datos, aún los mensajes de error, y los pasan hacia la interfaz, y ésta a su vez, los pasa hasta el software de procesamiento.

Cualquier otro software (servidor HTTP, software de redes, etc.) agrega enlaces adicionales a este proceso de extracción de la información, ya que el software de procesamiento pasa los resultados hacia el servidor Web, y éste hasta el browser del Web (ya sea directamente o a través de una red).

Los lenguajes de desarrollo web actuales están provistos de herramientas útiles para el enlace de una nueva solución con las bases de datos. Los lenguajes de

software libre apuntan a bases de datos no propietarias pero también ofrecen la posibilidad de interactuar con bases de datos propietarias. Las funciones son ejecutar sentencias desde la plataforma de desarrollo y la generación de código automático para los métodos comunes como son: Inserción, modificación y eliminación.

4.3.4. Manuales de ayuda

Un lenguaje bien documentado crea en el desarrollador una mayor confianza en el mismo. No se puede predecir que una plataforma jamás va a tener un problema y que es tan intuitiva que no necesita más allá de seguir un proceso simple para crear soluciones robustas. Por esta razón, las herramientas y la plataforma, en general, mientras mejor documentadas estén, van a ayudar a solucionar problemas frecuentes o enseñar a un nuevo usuario el uso correcto de sus herramientas y aprovechar la mayoría de los beneficios que éste propone.

Un manual de usuario ayuda al programador a auto-educarse y conocer a fondo la herramienta que está utilizando. El propósito general de un manual es proveer de información necesaria desde el inicio de un proyecto simple hasta la creación de soluciones de mayor extensión y que van a requerir funciones complejas. Los mejores manuales son los que tienen incorporados ejemplos prácticos en ellos y usan lenguaje técnico de fácil entendimiento. La idea de un manual es ayudar a resolver problemas de adecuación y uso correcto de una función en nuestro proyecto mediante información provista por los creadores de las herramientas.

4.4. Comparación entre Php y Symfony en la productividad del desarrollo mediante parámetros

El puntaje para los parámetros siguientes está dado por la experiencia obtenida en la creación de un módulo para el cobro de rubros; se desarrollo el mismo módulo primero en Php y luego en Symfony.

La calificación está dada por los valores entre el cero (0) que representa que no existe hasta cuatro (4) que simboliza que es excelente o fácil de aplicar.

Valor	Significado
0	No dispone
1	Regular
2	Bueno
3	Muy Bueno
4	Excelente

Tabla IV.VII: Valores para la evaluación de la productividad.

Entorno / Parámetro	Php	Symfony
LENGUAJE Y CODIFICACIÓN		
Diccionario de datos	0	0
Sintaxis	3	3
Uso de comentarios	2	3
Total	**6**	**6**
MANEJO DE ERRORES		
Paso de parámetros	3	4
Registro de errores	2	3
Reporte de errores	3	3
Métodos de excepción de errores	2	3
Manejo local de errores	2	3
Total	**12**	**16**
TIEMPO DE RESPUESTA		
Modificación de código	3	4
Compilación	3	3
Total	**6**	**7**
REUTILIZACIÓN DE CÓDIGO		
Herencia de clase base	0	4
Personalización de código	2	4

heredado		
Total	**2**	**8**
BONDADES DEL LENGUAJE		
Manejo de alguna metodología o patrón de diseño	0	4
Colaboración de herramientas CASE	0	0
Existe entorno de desarrollo integrado (IDE)	4	4
Generación de código	0	4
Estructura de un nuevo proyecto	0	4
Total	**4**	**16**
INTERACCION CON UNA BASE DE DATOS		
Conexión con la base de datos	3	4
Consultas a la base de datos	2	4
Total	**5**	**8**
AYUDA		
Manuales de ayuda	4	3
Foros	4	3
Total	**8**	**6**
CONTROL DE CAMBIOS		
Versionamiento	0	4
Respaldo de desarrollo	0	4
Total	**0**	8
TOTAL	**43**	**75**

Tabla IV.VIII: Evaluación de la productividad en el desarrollo Php vs Symfony

Interpretación de resultados: La diferencia entre Php y Symfony en el desarrollo de software tiene una considerable diferencia debido a que Symfony está basado en MVC que se traduce en reutilización de código y optimización de tiempos de implementación. El puntaje alcanzado por Php es de 43 mientras que el de Symfony es de 75. La generación de código con Symfony da una gran ventaja en las etapas de implementación del sistema web. El parámetro más sobresaliente es *Bondades del lenguaje,* pues es Symfony quien provee al desarrollador ventajas para una mejor y más rápida codificación.

4.5. Indicadores y variables

Algunas de las variables son más difíciles de medir que otras, puesto que presentan características poco cuantificables. Las variables no cuantificables y que no se mantienen en un registro histórico, son: habilidades administrativas del jefe de proyecto, metas adecuadas, capacidad individual de los programadores, entendimiento del problema, y nivel tecnológico. Este tipo de variables no se van a tomar en cuenta para el análisis de la hipótesis de este estudio.

Es importante destacar que el conjunto de variables que no va a ser incluido dentro del registro histórico debe ser considerado siempre antes de comenzar un proyecto. Es en la etapa de análisis dónde deben determinarse el papel que desempeña cada una ellas con el fin de mejorar sus valores.
Según Kepner las variables que son cuantificables se mantendrán en el registro histórico y se pueden categorizar en esenciales y opcionales. Una variable es esencial cuando su valor real es indispensable para la comparación con el valor registrado históricamente. Las variables consideradas como esenciales son: entrenamiento adecuado, uso de metodologías de diseño y programación, y lenguaje de programación.
Una variable es opcional cuando puede ser catalogada como útil, pero no indispensable para dicha comparación. Las variables consideradas opcionales son: control de cambios, comunicación del grupo, participación del usuario en la

definición de requerimientos, y cambios en el diseño del programa originados por el usuario.

La comparación de un conjunto de variables respecto del registro histórico se hace con el propósito de determinar si existe(n) algún(os) proyecto(s) que se asemeje(n) al actualmente en estudio. En este proceso todas las variables deberán ser consideradas por igual, es decir, no se harán diferencias entre esenciales y opcionales. En el caso que el número de proyectos seleccionados desde el registro histórico sea ínfimo o nulo, entonces se comenzará a descartar variables opcionales de acuerdo a algún criterio preestablecido para obtener un conjunto significativo de proyectos semejantes.

4.6. Determinación de Indicadores

Todo administrador de proyectos de desarrollo de software necesita saber cuál es el nivel de productividad que tiene su grupo de trabajo.
Los indicadores que se presentan a continuación se basan en forma importante en los datos del proyecto en estudio y en el registro histórico de proyectos pasados similares (Estudios de proyectos investigados y proyectos propios).

Es un factor muy importante el tamaño y la complejidad del proyecto para poder escoger la técnica de estimación adecuada. Para el módulo de evaluación se tiene una complejidad baja, y su tamaño es relativamente pequeño, razones por las cuales se ha escogido los métodos de PUNTOS DE FUNCIÓN Y COCOMO II, el primero ayudará a identificar todos los requerimientos con sus respectivas complejidades, y la herramienta COCOMO II permitirá encontrar el costo y la duración del proyecto.

4.6.1. COCOMO II

Para el módulo de comparación entre Php y Symfony se han considerado 129 puntos de función entre los cuales están 8 entradas como son: ingresos, modificaciones y eliminaciones de los datos para las tablas del estudiante y del rubro especial. En las entradas existen 6 de nivel bajo y 2 de nivel medio. Para las salidas se han considerado 2 de nivel bajo y 1 de nivel medio, entre ellas se tiene los listados de estudiantes, rubros especiales y cuentas por pagar. Hay 10 interfaces de las cuales 8 son de nivel bajo y 2 de nivel medio. Para la interaccion a la base ded datos se han considerado 11 puntos de función, entre los requerimientos están: ingresos, modificación, eliminación y listado. Total de líneas de código= 4128 líneas de código.

- **Definición de puntos de función**

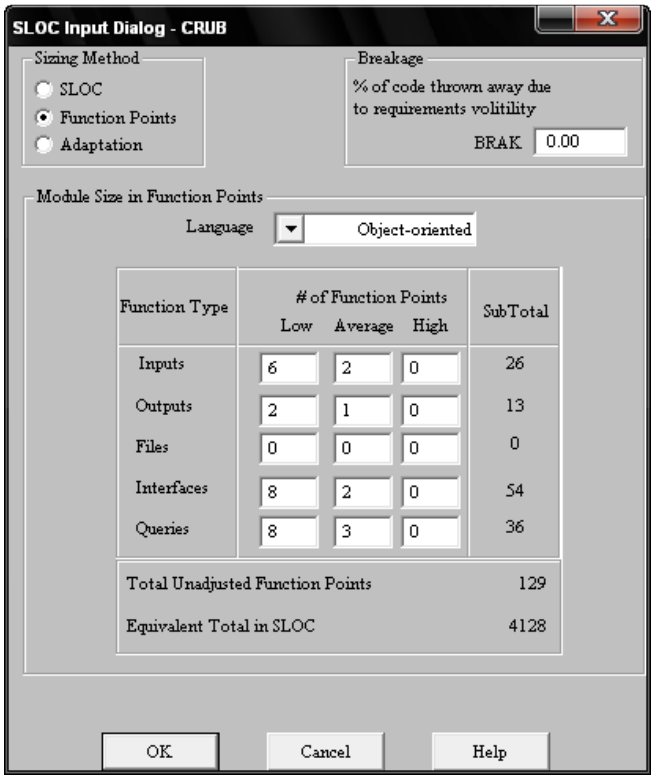


Ilustración IV.12: Puntos de función según Cocomo.

- **Total de persona mes a desarrollar un proyecto**

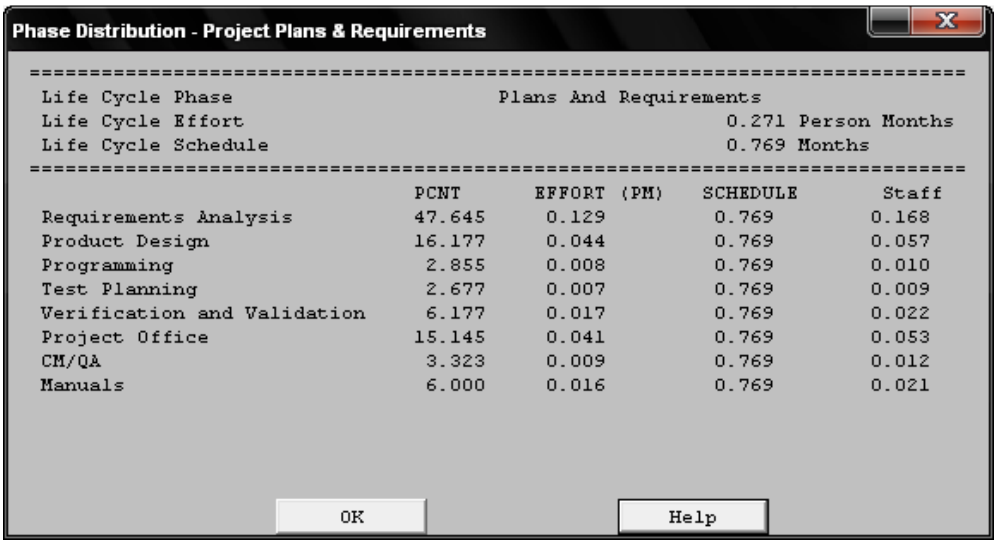


Ilustración IV.13: Planes y requerimientos según cocomo.

- **Tiempo a desarrollar y número de personas para la Fase de Diseño**

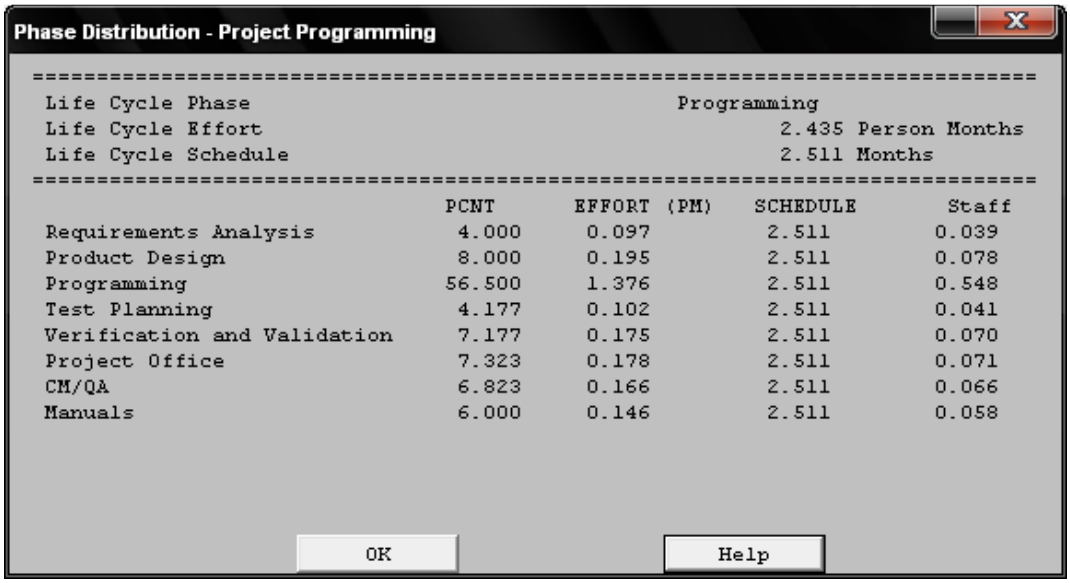


Ilustración IV.14: Programación del proyecto según cocomo.

- **Tiempo para el diseño del producto**

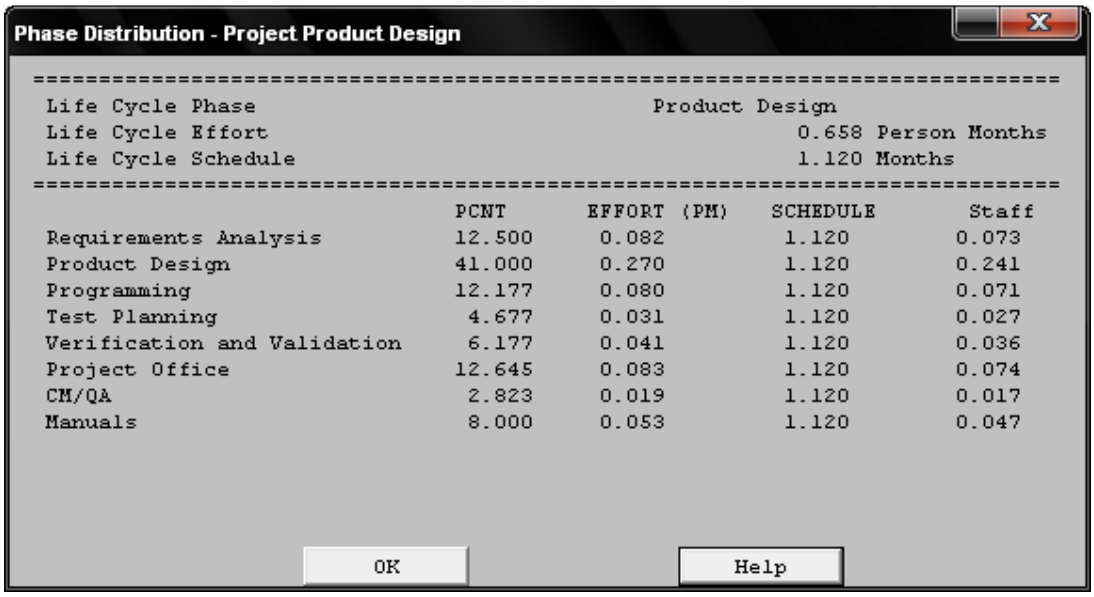


Ilustración IV.15: Diseño del producto según cocomo.

- **Tiempo y personas para la integración y pruebas**

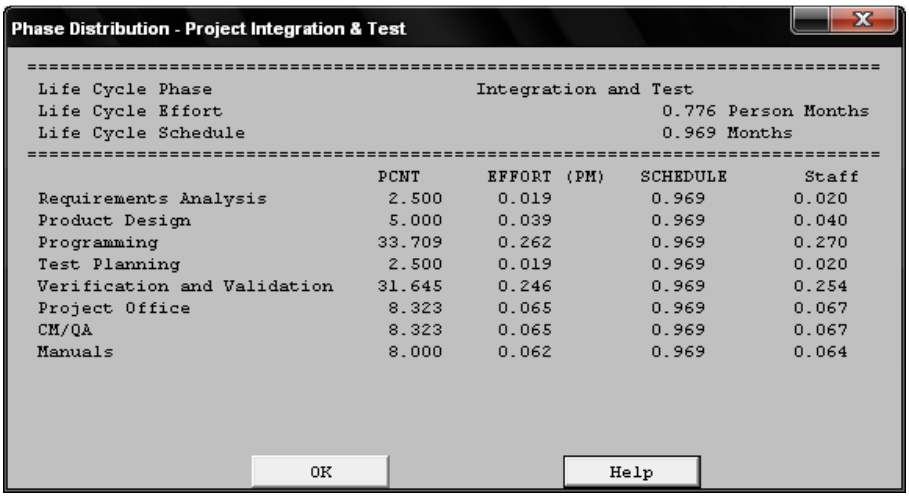


Ilustración IV.16: Integración y pruebas según cocomo

Total Costo de Desarrollo del Proyecto

Para determinar el costo del módulo que se desarrollo, se utilizó la herramienta Cocomo II y se obtuvo lo siguiente:

- Para un total de 4128 líneas de código y 129 puntos de función (no ajustados) el módulo de evaluación tendrá un costo de $1934.65 dólares y tendrá una duración de 4 meses y medio.

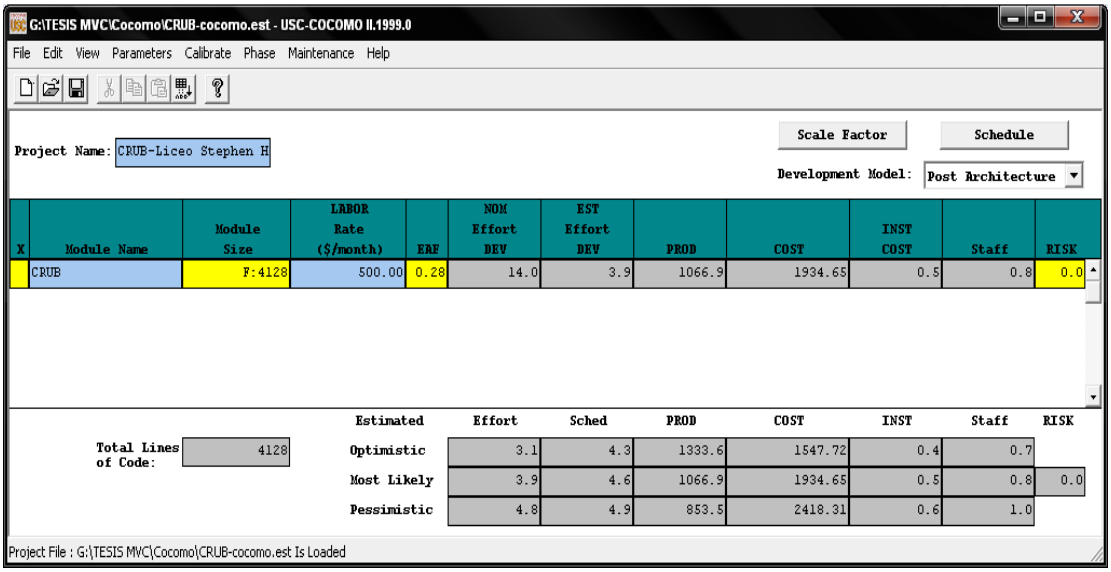


Ilustración IV.17: Datos Cocomo para el módulo de evaluación.

4.7. Comparación para medir la productividad entre Php y Symfony mediante indicadores

Para medir la productividad se realizó una comparación en el desarrollo de un módulo del sistema de cobro de rubros. El mismo módulo se desarrollo primero con el lenguaje de software libre Php y después empleando el framework Symfony basado con MVC. Los resultados obtenidos con Php se denominarán registro histórico. La mayoría de resultados se los hace midiendo el tiempo empleado en las actividades de desarrollo.

4.7.1. Índice de variación con respecto a los puntos de función

Este índice mide el tiempo de variación del desarrollo de los puntos de función bajos (poca dificultad) con respecto a los puntos de función promedio (mediana dificultad).

Se realizó la medición de tiempos en el desarrollo del ***módulo de rubros especiales*** primero con php y luego usando el framework symfony y se obtuvo los siguientes resultados:

Herramientas / Puntos de función	PHP	Symfony
Bajos	35 minutos aproximadamente	25 minutos aproximadamente
Promedio	45 minutos aproximadamente	39 minutos aproximadamente

Tabla IV.IX: Variación en el tiempo de desarrollo según puntos de función.

Interpretación de resultados: Se puede concluir que symfony al heredar de una clase base varios procedimientos para generar automáticamente las funciones básicas de interacción con las tablas de una base de datos, no se requiere de mucho esfuerzo y el tiempo en el que se soluciona varios puntos de función de nivel bajo es mínimo. En el nivel medio, el tiempo es optimizado con Symfony, debido a que en php se puede crear el tipo de código que se desee mientras que utilizando symfony, si es cierto que se puede crear cualquier tipo de procedimiento, se puede utilizar funciones ya establecidas y que se pueden personalizar para requerimientos propios, pero necesita de un amplio conocimiento en éstos.

4.7.2. Indicador con base en el estándar promedio al comienzo

Este indicador permite conocer si la productividad que se tuvo al término del desarrollo de un proyecto se asemeja a la estimada obtenida desde el registro histórico, tomando en consideración los datos generados al inicio del proyecto. La fórmula utilizada es:

$$Indice\ al\ comienzo = \frac{Est\ ndar\ real - Est\ ndar\ promedio}{Est\ ndar\ promedio}$$

Donde los estándares corresponden al cociente entre las Horas-Hombre y los puntos de función, ya sean reales como promedios (estos últimos se obtienen desde el registro histórico).

Según los datos obtenidos con el software COCOMO II se obtuvieron un total de 129 puntos de función. Para la parte inicial se consideran 89 puntos de función para los cuales empleando Php se desarrolló el módulo en un tiempo de 48 horas y con Symfony en 36 horas. Los datos calculados aplicando la fórmula serían:

$$Estándar\ promedio = \frac{48}{89} = 0.54$$

$$Estándar\ real = \frac{36}{89} = 0.40$$

$$Indice\ al\ comienzo = \frac{0.40 - 0.54}{0.54} = -0.26$$

Interpretación: El estándar promedio obtenido implica que para cada punto de función se emplea aproximadamente 32 minutos; mientras que el estándar real es alrededor de 24 minutos. El *índice al comienzo* es menor a cero lo que significa que el desarrollo de los primeros puntos de función con Symfony se elaboró en menos tiempo de lo esperado. Este tiempo de diferencia es muy significativo.

4.7.3. Indicador con base en el estándar promedio al término

Este indicador es similar al anterior y difiere sólo por el hecho que permite conocer si la productividad que se tuvo al término del desarrollo de un proyecto, se asemeja a la estimada obtenida desde el registro histórico - tomando en consideración los datos generados al finalizar el proyecto-. La fórmula correspondiente es:

$$Indice\ al\ tÇrmino = \frac{Est\ ndar\ real\ -\ Est\ ndar\ promedio}{Est\ ndar\ promedio}$$

Donde los estándares se definen análogamente al índice con base en el estándar promedio al comienzo.

Para este cálculo se obtuvo del registro histórico un tiempo de 28 horas para 40 puntos de función. Hay que tomar en cuenta que los últimos puntos de función son de mayor complejidad. El tiempo que tomó realizar éstos puntos de función con Symfony fue de 24 horas.

$$Estándar\ promedio = \frac{28}{40} = 0.7$$

$$Estándar\ real = \frac{24}{40} = 0.6$$

$$Indice\ al\ término = \frac{0.6 - 0.7}{0.7} = -0.14$$

Interpretación: El estándar promedio obtenido implica que para cada punto de función se emplea aproximadamente 42 minutos; mientras que el estándar real es alrededor de 36 minutos. El *índice al término* es menor a cero lo que significa que el desarrollo de los últimos puntos de función con Symfony se elaboró en menos tiempo de lo esperado. Este tiempo de diferencia no es muy notorio pocos puntos de función pero en un módulo de mayor extensión representaría una ventaja muy grande. Se puede decir que en el desarrollo de puntos de función muy complejos el tiempo de desarrollo es ligeramente más rápido empleando Symfony.

4.8. Interpretación de los indicadores

En la Ilustración IV.18 se puede observar claramente que el tiempo de desarrollo de Symfony es menor que el empleando por Php.

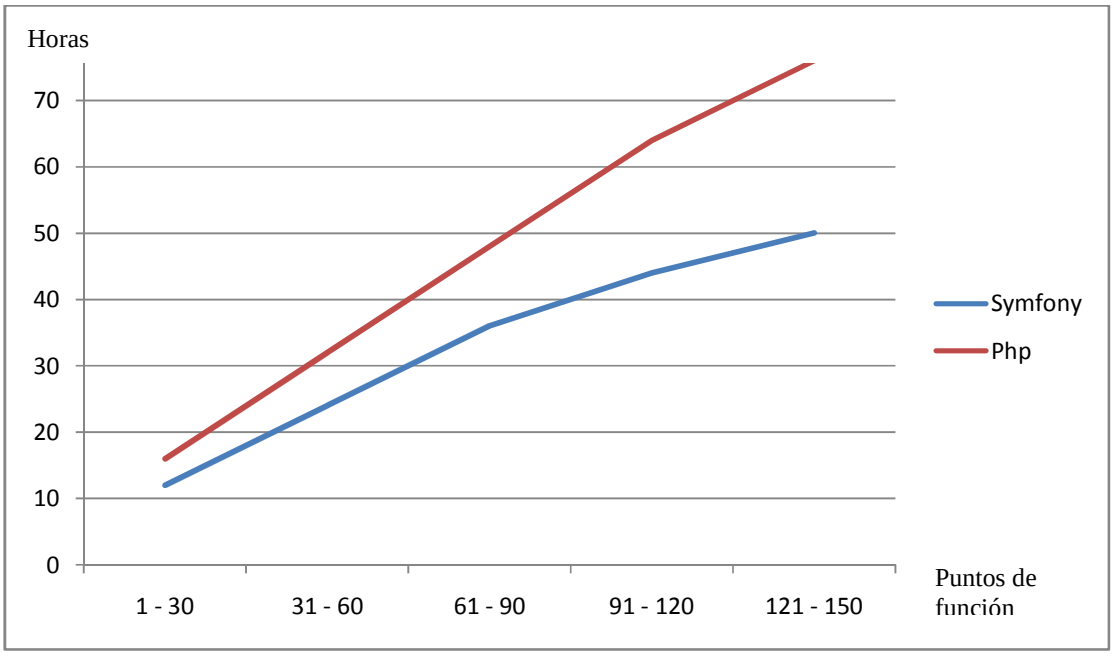


Ilustración IV.18: Desarrollo de puntos de función Php vs Symfony

El cálculo de los indicadores nos demuestra que que Symfony es más útil y práctico en puntos de función de nivel bajo. Se observa que Symfony optimiza los tiempos de desarrollo y que es justificado el incremento del tiempo para los últimos puntos de función debido a que muestran un mayor grado de dificultad.

4.9. Productividad individual

Este indicador permite conocer el rendimiento de una persona en todas aquellas actividades que ha realizado. La fórmula se define como:

Donde las *Horas-Hombre reales promedio* corresponden a todas las actividades concluidas por la persona, y los *PF reales promedio* se refiere a los puntos de función asignados a la persona con respecto al total de las actividades en que participó.

Con Php:

$$Productividad\ Individual\ Php = \frac{65\ PF\ asignados}{38\ horas\ de\ trabajo} = 1.71$$

Con Symfony:

$$Productividad\ Individual\ Symfony = \frac{32\ horas\ de\ trabajo}{65\ PF\ asignados} = 2.03$$

Productividad Individual Php < Productividad Individual Symfony

Interpretación: Podemos concluir que la productividad individual utilizando Symfony es mayor que la productividad individual utilizando Php ya que el tiempo empleado a desarrollar los puntos de función asignados a un programador se culminan en menor tiempo usando el framework.

CAPÍTULO V

Desarrollo e implementación de una Aplicación Web para el Cobro de Rubros

5.1. Planificación

5.1.1. Historia del Usuario

La secretaria realiza varias actividades en lo referentes a la venta o contratación de servicios y al cobro de los mismos, entre los más importantes tenemos:

- Vender y cobrar uniformes
- Vender y cobrar libros
- Vender y cobrar boletos para rifas, entradas para eventos, y otros rubros.
- Realizar matriculas
- Cobrar Pensiones
- Asignar y cobrar servicios de transporte, alimentación y tareas dirigidas a los estudiantes que lo requieran

El administrador, en este caso el director del Liceo de Talentos Stephen Hawking realiza las siguientes actividades:

- Ingresar y modificar datos del estudiante
- Asignar costos a las pensiones
- Administrar datos de uniformes (uniforme, talla,stock,valor).
- Administrar datos de libros (nombre, precio)
- Administrar datos de rubros especiales
- Administrar servicios de transporte y alimentación
- Administrar tareas dirigidas
- Administrar niveles y becas(crear, modificar,eliminar)
- Administrar períodos
- Manejar stock de uniformes
- Realizar búsquedas personalizadas

En resumen la secretaria es un tipo de usuario que manejará la aplicación web tanto para la venta como para el cobro de los diferentes servicios ofrecidos a los estudiantes de la institución educativa. El usuario encargado de la administración de la aplicación tendrá la posibilidad de crear diferentes tipos de rubros y asignarles un costo, manipular diferentes tipos de información como datos de estudiantes, niveles, periodos, y servicios en general.

5.1.2. Plan de publicaciones

Versión	Historias de Usuario	Fecha de Publicación
Siscrub 1.0	• Ingresar y modificar datos del Estudiante • Administrar niveles y becas • Administrar períodos	24/09/2010

Siscrub 1.1	• Administrar datos de uniformes • Manejar stock de uniformes • Administrar datos de libros • Administrar datos rubros especiales • Asignar costos a las pensiones	01/10/10
Siscrub 1.2	• Administrar servicios de alimentación y transporte • Administrar tareas dirigidas • Realizar búsquedas personalizadas	08/10/10
Siscrub 1.3	• Realizar matriculas • Vender y cobrar uniformes • Vender y cobrar libros • Vender y cobrar boletos para rifas, entradas para eventos y otros rubros	15/10/10
Siscrub 1.4	• Cobrar pensiones • Asignar y cobrar servicios de transporte, alimentación y tareas dirigidas	28/10/10

.**Tabla VI.X:** Plan de Publicaciones

5.1.3. Iteraciones

Iteraciones	Historias del Usuario
Iteracion1	• Ingresar y modificar datos del Estudiante • Administrar niveles y becas • Administrar períodos
Iteracion2	• Administrar datos de uniformes • Manejar stock de uniformes • Administrar datos de libros • Administrar datos rubros especiales • Asignar costos a las pensiones
Iteracion3	• Administrar servicios de alimentación y transporte • Administrar tareas dirigidas

	• Asignar tareas dirigidas • Realizar búsquedas personalizadas
Iteracion4	• Realizar matriculas • Vender y cobrar uniformes • Vender y cobrar libros • Vender y cobrar boletos para rifas, entradas para eventos y otros rubros • Realizar reportes
Iteracion5	• Cobrar pensiones • Asignar y cobrar servicios de transporte, alimentación y tareas dirigidas • Realizar Reportes

Tabla V.XI: Iteraciones

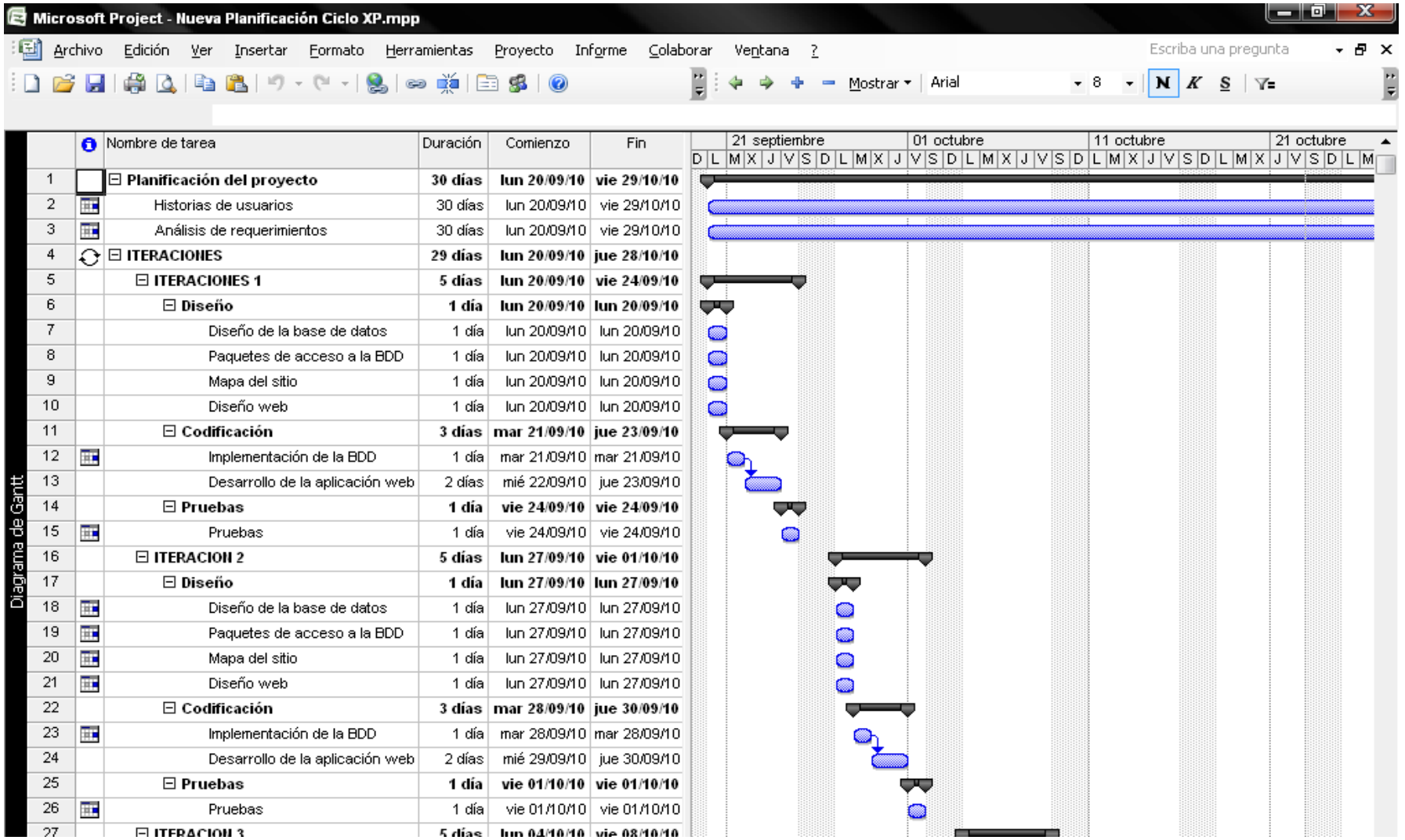
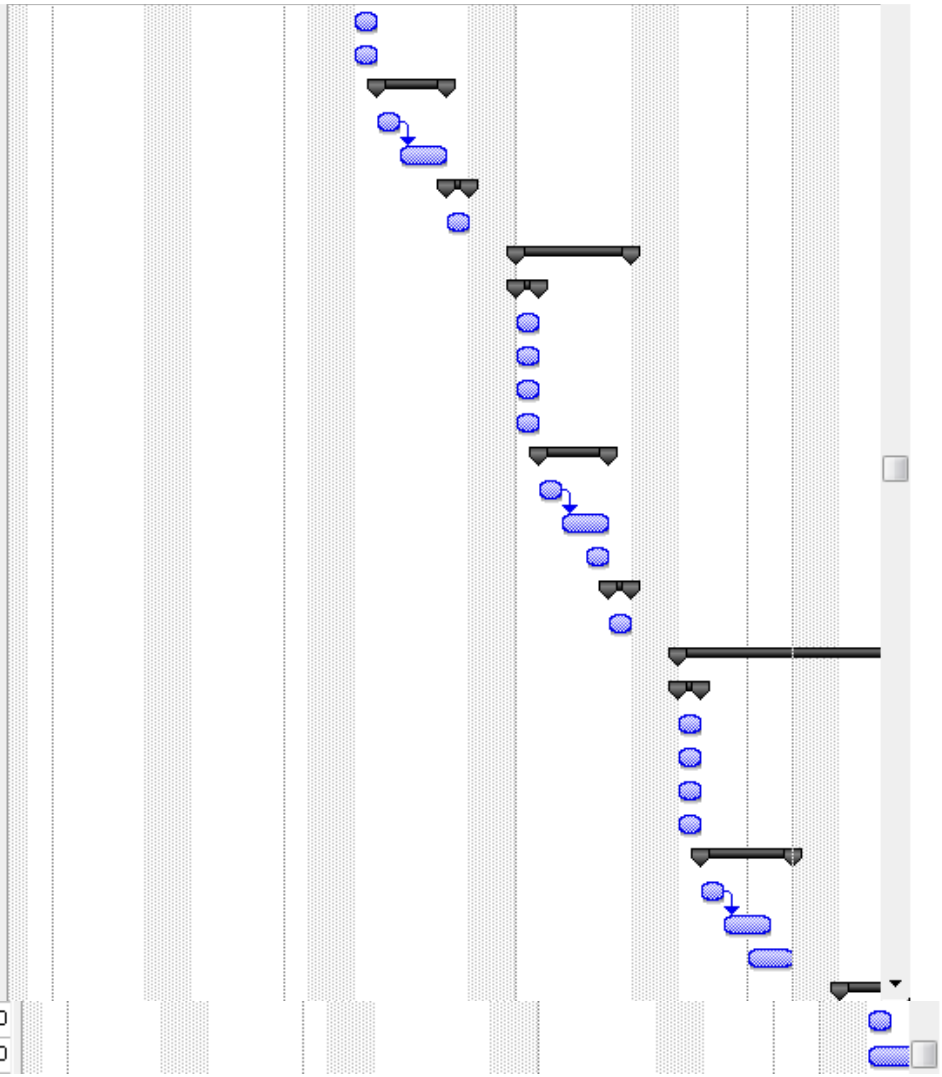

Microsoft Project - Nueva Planificación Ciclo XP.mpp
Archivo Edición Ver Insertar Formato Herramientas Proyecto Informe Colaborar Ventana ?
Escriba una pregunta
Mostrar Arial 8
Diagrama de Gantt
21 septiembre
01 octubre
11 octubre
21 octubre
D L M X J V S D L M X J V S D L M X J V S D L M X J V S D L M X J V S D L M X J V S D L M X J V S D L M
1 Planificación del proyecto 30 días lun 20/09/10 vie 29/10/10
2 Historias de usuarios 30 días lun 20/09/10 vie 29/10/10
3 Análisis de requerimientos 30 días lun 20/09/10 vie 29/10/10
4 ITERACIONES 29 días lun 20/09/10 jue 28/10/10
5 ITERACIONES 1 5 días lun 20/09/10 vie 24/09/10
6 Diseño 1 día lun 20/09/10 lun 20/09/10
7 Diseño de la base de datos 1 día lun 20/09/10 lun 20/09/10
8 Paquetes de acceso a la BDD 1 día lun 20/09/10 lun 20/09/10
9 Mapa del sitio 1 día lun 20/09/10 lun 20/09/10
10 Diseño web 1 día lun 20/09/10 lun 20/09/10
11 Codificación 3 días mar 21/09/10 jue 23/09/10
12 Implementación de la BDD 1 día mar 21/09/10 mar 21/09/10
13 Desarrollo de la aplicación web 2 días mié 22/09/10 jue 23/09/10
14 Pruebas 1 día vie 24/09/10 vie 24/09/10
15 Pruebas 1 día vie 24/09/10 vie 24/09/10
16 ITERACION 2 5 días lun 27/09/10 vie 01/10/10
17 Diseño 1 día lun 27/09/10 lun 27/09/10
18 Diseño de la base de datos 1 día lun 27/09/10 lun 27/09/10
19 Paquetes de acceso a la BDD 1 día lun 27/09/10 lun 27/09/10
20 Mapa del sitio 1 día lun 27/09/10 lun 27/09/10
21 Diseño web 1 día lun 27/09/10 lun 27/09/10
22 Codificación 3 días mar 28/09/10 jue 30/09/10
23 Implementación de la BDD 1 día mar 28/09/10 mar 28/09/10
24 Desarrollo de la aplicación web 2 días mié 29/09/10 jue 30/09/10
25 Pruebas 1 día vie 01/10/10 vie 01/10/10
26 Pruebas 1 día vie 01/10/10 vie 01/10/10
27 ITERACION 3 5 días lun 04/10/10 vie 08/10/10

Diagrama de Gantt

		Nombre de tarea	Duración	Comienzo	Fin
31		Mapa del sitio	1 día	lun 04/10/10	lun 04/10/10
32		Diseño web	1 día	lun 04/10/10	lun 04/10/10
33		**Codificación**	**3 días**	**mar 05/10/10**	**jue 07/10/10**
34		Implementación de la BDD	1 día	mar 05/10/10	mar 05/10/10
35		Desarrollo de la aplicación web	2 días	mié 06/10/10	jue 07/10/10
36		**Pruebas**	**1 día**	**vie 08/10/10**	**vie 08/10/10**
37		Pruebas	1 día	vie 08/10/10	vie 08/10/10
38		**ITERACION 4**	**5 días**	**lun 11/10/10**	**vie 15/10/10**
39		**Diseño**	**1 día**	**lun 11/10/10**	**lun 11/10/10**
40		Diseño de la base de datos	1 día	lun 11/10/10	lun 11/10/10
41		Paquetes de acceso a la BDD	1 día	lun 11/10/10	lun 11/10/10
42		Mapa del sitio	1 día	lun 11/10/10	lun 11/10/10
43		Diseño web	1 día	lun 11/10/10	lun 11/10/10
44		**Codificación**	**3 días**	**mar 12/10/10**	**jue 14/10/10**
45		Implementación de la BDD	1 día	mar 12/10/10	mar 12/10/10
46		Desarrollo de la aplicación web	2 días	mié 13/10/10	jue 14/10/10
47		Creación de Reportes	1 día	jue 14/10/10	jue 14/10/10
48		**Pruebas**	**1 día**	**vie 15/10/10**	**vie 15/10/10**
49		Pruebas	1 día	vie 15/10/10	vie 15/10/10
50		**ITERACION 5**	**9 días**	**lun 18/10/10**	**jue 28/10/10**
51		**Diseño**	**1 día**	**lun 18/10/10**	**lun 18/10/10**
52		Diseño de la base de datos	1 día	lun 18/10/10	lun 18/10/10
53		Paquetes de acceso a la BDD	1 día	lun 18/10/10	lun 18/10/10
54		Mapa del sitio	1 día	lun 18/10/10	lun 18/10/10
55		Diseño web	1 día	lun 18/10/10	lun 18/10/10
56		**Codificación**	**4 días**	**mar 19/10/10**	**vie 22/10/10**
57		Implementación de la BDD	1 día	mar 19/10/10	mar 19/10/10
58		Desarrollo de la aplicación web	2 días	mié 20/10/10	jue 21/10/10
59		Creación de Reportes	2 días	jue 21/10/10	vie 22/10/10
60		**Pruebas**	**4 días**	**lun 25/10/10**	**jue 28/10/10**
61		Instalación	1 día	lun 25/10/10	lun 25/10/10
62		Pruebas	4 días	lun 25/10/10	jue 28/10/10



5.1.4. Programación en pareja

La metodología X.P. aconseja la programación en parejas pues incrementa la productividad y la calidad del software desarrollado. Con esta finalidad los roles que desempeñarán los desarrolladores son los siguientes:

Personas	Roles	Funciones
Víctor Miguel Fuertes Ortega	Gestor	Es el vínculo entre programadores y cliente.
	Gestor Base de Datos	Diseñar e implementar base de datos y todas las funciones relacionadas a la misma.
	Tester	Realizar pruebas funcionales
Juan Carlos Guevara Morocho	Programador	Producir código del sistema
	Tracker	Realizar el seguimiento del progreso de cada iteración y evaluar cumplimiento de objetivos.
	Publicador	Dar de alta versiones y publicarlas en la web.

Tabla V.XII: Roles de desarrollo

Gracias a la asignación de estos roles se consigue que mientras uno codifica haciendo hincapié en la calidad de la función o método que está implementando, el otro analiza si ese método o función es adecuado y está bien diseñado. De esta forma se consigue un código y diseño con gran calidad.

5.1.5. Plan de reuniones

La comunicación entre las diferentes partes que interviene en un proyecto resulta fundamental para su desarrollo. Las reuniones van a ser frecuentes, de poca duración y de ser posible delante de la pantalla del ordenador. Serán fundamentales las reuniones al final de las iteraciones con el propósito de sacar a la luz los problemas, las soluciones y centrar el objetivo del equipo.

El usuario será parte fundamental del equipo de desarrollo del proyecto.

5.2. Diseño

5.2.1. Metáfora

Sistema para venta de servicios y cobro de los mismos, con gestión de estudiantes y rubros.

5.2.2. Glosario de términos

Término	Definición
SISCRUB	Sistema de Cobro de Rubros.
Banner	Es una imagen o gráfico que permite a una empresa anunciarse
BD	Base de datos
Navegador	Programa o aplicación que nos permite navegar por Internet
CSS	Formato de archivo que permite dar una presencia homogénea a varias páginas Web
HTML	Es el lenguaje con que se escriben las páginas Web
Interfaz	Punto de contacto entre el usuario, el ordenador y el programa
Código **Fuente**	Programa en su forma original, tal y como fue escrito por el programador
Compilador	Programa de computadora que produce un programa en lenguaje de maquina
Variable	Estructura que contiene datos y recibe un nombre único dado por el programador,
Usuario	Cualquier individuo que iteractúa con la computadora a nivel de aplicación

Campo	Espacio en la memoria que sirve para almacenar temporalmente un dato durante el proceso
Programador	Individuo que diseña la lógica y escribe las líneas de código de un programa de computadora.
Información	Es lo que se obtiene del procesamiento de datos

Tabla V.XIII: Glosario de Términos

5.2.3. Riesgos

En este punto vamos a desglosar y analizar los posibles riesgos que pueden presentarse durante el desarrollo y utilización del software y al mismo tiempo analizar el impacto que estos tendrían en caso de convertirse en problemas y la manera en que se podría gestionar una solución. Con este procedimiento podemos entender las fortalezas y debilidades que tendría el desarrollo del software y así solicitar ayuda técnica en caso de requerirla.

5.2.3.1. Identificación del Riesgo

Identificación	Descripción del riesgo	Categoría
R1	Las herramientas de desarrollo podrían sufrir algún daño (quemarse, perderse, etc.).	Riesgo de proyecto
R2	Las interfaces del sistemas pueden ser difíciles de utilizar	Riesgo de técnico
R3	El producto final podría no tener una seguridad adecuada	Riesgo técnico
R4	Mala estimación de recursos como tiempo, personal, dinero.	Riesgo del proyecto

R5	Podría producirse la deserción de miembros del personal de desarrollo.	Riesgo del proyecto
R6	El producto final no llena las expectativas de la empresa solicitante.	Riesgo del negocio
R7	Los programadores y diseñadores podrían no tener la experiencia necesaria.	Riesgo técnico
R8	Oferta de nuevos productos con funcionamiento similar y menor costo	Riesgo del negocio
R9	El usuario podría no revisar los avances en el proyecto	Riesgo proyectos
R10	Fallas en la base de datos o en las conexiones a la misma.	Riesgo técnico
R11	La aplicación final podría no ser compatible con cierto navegador	Riesgo técnico
R12	El personal de desarrollo puede entender mal los requerimientos	Riesgo del técnico.
R13	El personal de desarrollo podría detener temporalmente sus labores por feriados o vacaciones	Riesgo del proyecto
R14	El proyecto no cumpla con todos los requerimientos en el tiempo especificado	Riesgo técnico

Tabla V.XIV: Identificación de Riesgos

5.2.3.2. Análisis del riesgo

5.2.3.2.1. Determinación de la Probabilidad

Identificación	Probabilidad			Impacto		Exposición al riesgo	
	%	Valor	Probabil.	Valor	Impacto	Valor	Exposic.
R1	40%	2	Media	3	Alto	6	Alta
R2	60%	2	Media	4	Critico	8	Alta
R3	70%	3	Alta	3	Alto	9	Alta
R4	70%	3	Alta	3	Alto	9	Alta
R5	70%	3	Alta	2	Moderado	6	Alta
R6	30%	1	Baja	4	Critico	4	Media
R7	20%	1	Baja	2	Moderado	2	Baja
R8	40%	2	Media	3	Alto	6	Alta
R9	20%	1	Baja	3	Alto	3	Media
R10	10%	1	Baja	4	Critico	4	Media
R11	50%	2	Media	2	Moderado	4	Media
R12	70%	3	Alta	4	Critico	12	Alta
R13	40%	2	Media	3	Alto	6	Alta
R14	30%	1	Baja	3	Alto	3	Media

Tabla V.XV: Determinación de Probabilidades

Rango de probabilidad	Descripción	Valor
1% - 33%	Bajo	3
34% - 67%	Media	2
68% - 99%	Alta	1

Tabla V.XVI: Valores Probabilidad

5.2.3.2.2. Determinación del Impacto

Impacto	Retraso	Impacto Técnico	Costo	Valor
Bajo	10 días	Ligero efecto	<1%	1
Moderado	20 días	Moderado efecto	<5%	2
Alto	1 mes	Severo efecto	<10%	3
Critico	>1 mes	Proyecto no puede	>10%	4

Tabla V.XVII: Determinación de Impacto

5.2.3.2.3. Determinación de la Exposición al riesgo

Exposición al riesgo	Valor	Color
Baja	1 o 2	Verde
Media	3 o 4	Amarilla
Alta	>6	Roja

Tabla V.XVIII: Valores Exposición al riesgo

Impacto / Probabil.	Baja 1	Moderada 2	Alto 3	Critico 4
Alto 3	3	6	9	12
Medio 2	2	4	6	8
Bajo 1	1	2	3	4

Tabla V.XIX: Exposición a Riesgo

5.2.3.2.4. Determinación de la Prioridad del riesgo

Identificación	Descripción	Exposición	Prioridad
R12	El personal de desarrollo puede entender mal los requerimientos	12	1
R3	El producto final podría no tener una	9	2

	seguridad adecuada		
R4	Mala estimación de recursos como tiempo, personal, dinero.	9	2
R2	Las interfaces del sistemas pueden ser difíciles de utilizar	8	3
R1	Las herramientas de desarrollo podrían sufrir algún daño (quemarse, perderse, etc.).	6	4
R5	Podría producirse la deserción de miembros del personal de desarrollo	6	4
R8	Oferta de nuevos productos con funcionamiento similar y menor costo	6	4
R13	El personal de desarrollo podría detener temporalmente sus labores por feriados o vacaciones	6	4
R6	El producto final no llena las expectativas de la empresa solicitante.	4	5
R10	Fallas en la base de datos o en las conexiones a la misma.	4	5
R11	La aplicación final podría no ser compatible con cierto sistema operativo	4	5
R9	El usuario podría no revisar los avances en el proyecto	3	6
R14	El proyecto no cumpla con todos los requerimientos en el tiempo especificado	3	6
R7	Los programadores y diseñadores podrían no tener la experiencia necesaria	2	7

Tabla V.XX: Determinación Probabilidad Riesgo

5.2.3.3. Plan de Reducción, Supervisión y Gestión del Riesgo

<table>
<tr><td colspan="4">HOJA DE GESTIÓN DEL RIESGO</td></tr>
<tr><td colspan="2">ID. DEL RIESGO: R12</td><td colspan="2">FECHA: 22/septiembre/2010</td></tr>
<tr><td>Probabilidad:
Alta
Valor: 3</td><td>Impacto:
Crítico
Valor: 4</td><td>Exposición:
Alta
Valor: 12</td><td>Prioridad: 1</td></tr>
<tr><td colspan="4">DESCRIPCION.- El personal de desarrollo puede entender mal los requerimientos</td></tr>
<tr><td colspan="4">REFINAMIENTO:
<u>Causas</u>:
• Los desarrolladores pueden interpretar mal los requerimientos del cliente.
• No existe una comunicación fluida entre el cliente y los desarrolladores.
<u>Consecuencias</u>:
• Se entienden mal los requerimientos y por lo tanto se realizara mal el proyecto
• Retraso del proyecto</td></tr>
<tr><td colspan="4">REDUCCIÓN:
• Tener una buena comunicación con el cliente.
• Realizar entrevistas, cuestionarios, etc.</td></tr>
<tr><td colspan="4">SUPERVISION:
• Las relaciones humanas entre desarrolladores y cliente.
• La actitud de cooperación del cliente.</td></tr>
<tr><td colspan="4">GESTIÓN:
• Buscar una manera de minimizar el impacto con la finalidad de no volver a realizar el proyecto en su totalidad.</td></tr>
<tr><td colspan="4">ESTADO ACTUAL:
Fase de reducción iniciada X
Fase de Supervisión iniciada</td></tr>
</table>

Gestionando el riesgo:
RESPONSABLES: Miguel Fuertes, Juan Guevara

Tabla V.XXI: Gestión de Riesgo 12

HOJA DE GESTIÓN DEL RIESGO			
ID. DEL RIESGO: R3		**FECHA**: 22/septiembre/2010	
Probabilidad: Alta **Valor: 3**	**Impacto:** Alto **Valor: 3**	**Exposición:** Alta **Valor: 9**	**Prioridad: 2**
DESCRIPCION.- El producto final podría no tener una seguridad adecuada			
REFINAMIENTO: **<u>Causas</u>:** • Un mal entendimiento de los requerimientos • Poca experiencia de los desarrolladores en seguridad de sistemas **<u>Consecuencias</u>:** • Nuestro sistema no brindaría las seguridades necesarias • Personas ajenas al sistema podrían dañarlo.			
REDUCCIÓN: • Investigar acerca de las seguridades de los sistemas y de los posibles ataques de los que podría ser víctima nuestro sistema. • Hacer una buena planificación considerando la presencia de este riesgo.			
SUPERVISION: • La asignación diferenciada de permisos a cada usuario. • Las vulnerabilidades de nuestro sistema. • Laceración de cuentas para cada usuario.			

GESTIÓN: • Reforzar el proyecto en los puntos en los cuáles a mostrado debilidad.
ESTADO ACTUAL: Fase de reducción iniciada X Fase de Supervisión iniciada Gestionando el riesgo:
RESPONSABLES: Miguel Fuertes, Juan Guevara

Tabla V.XXII: Gestión de Riesgo 3

HOJA DE GESTIÓN DEL RIESGO			
ID. DEL RIESGO: R4		**FECHA**: 22/septiembre/2010	
Probabilidad: Alta **Valor: 3**	**Impacto:** Alta **Valor: 3**	**Exposición:** Alta **Valor: 9**	**Prioridad: 2**
DESCRIPCIÓN: Mala estimación de recursos como tiempo, personal, dinero.			
REFINAMIENTO: **<u>Causas</u>:** • Poca experiencia en estimaciones • Poca experiencia en definición de requerimientos **<u>Consecuencias</u>:** • Podríamos quedarnos sin recursos a medio desarrollo • Retraso del proyecto			
REDUCCIÓN: • Pedir ayuda o capacitarles a aquellas personas del grupo de desarrollo que no poseen estos conocimientos. • Trabajar siempre teniendo en cuenta una cantidad mayor de recursos porque es preferible que sobre antes que falte.			

SUPERVISION: • Los recursos con los que contamos para desarrollar el proyecto. • La utilización y deterioro de los recursos.
GESTIÓN: • Hacer gestiones para adquirir los recursos necesitados. • Hacer una nueva estimación de recursos y presentárselos al cliente para que él nos ayude económicamente.
ESTADO ACTUAL: Fase de reducción iniciada X Fase de Supervisión iniciada Gestionando el riesgo:
RESPONSABLES: Juan Guevara, Miguel Fuertes

Tabla V.XXIII: Gestión de Riesgo 4

HOJA DE GESTIÓN DEL RIESGO			
ID. DEL RIESGO: R2		**FECHA**: 29 septiembre 2010	
Probabilidad: Media **Valor: 2**	**Impacto:** Critico **Valor: 4**	**Exposición:** Media **Valor: 8**	**Prioridad: 3**
DESCRIPCIÓN: Las interfaces del sistemas pueden ser difíciles de utilizar			
REFINAMIENTO: **<u>Causas</u>:** • Poco conocimiento en el diseño de interfaces por parte del equipo técnico • Falta de tiempo para la realización de interfaces más amigables • No se domina el software de desarrollo **<u>Consecuencias</u>:** • El usuario tendrá muchos inconvenientes para utilizar el software • Las capacidades del software no serían utilizadas al máximo			

REDUCCIÓN: • Buscar ayuda en personas especializadas en interfaces. • Las interfaces deben ser desarrolladas con tiempo. • Las interfaces deben estar de acorde con el conocimiento académico del usuario.
SUPERVISION: • La calidad de las interfaces • La amigabilidad de las interfaces • El lenguaje utilizado en las interfaces debe ser claro
GESTIÓN: • Dar un curso de capacitación a los usuarios • Presentar un manual de uso con todas las funcionalidades que tiene el software. • Corregir las interfaces.
ESTADO ACTUAL: Fase de reducción iniciada Fase de Supervisión iniciada X Gestionando el riesgo:
RESPONSABLES: Miguel Fuertes, Juan Guevara

Tabla V.XXIV: Gestión de Riesgo 2

HOJA DE GESTIÓN DEL RIESGO			
ID. DEL RIESGO: R1		**FECHA**: 22/septiembre/2010	
Probabilidad: Media **Valor: 2**	**Impacto:** Alto **Valor: 3**	**Exposición:** Alta **Valor:6**	**Prioridad:** **4**
DESCRIPCIÓN: Las herramientas de desarrollo podrían sufrir algún daño (quemarse, perderse, etc.).			

REFINAMIENTO:

<u>Causas</u>:

- Podrían quemarse los equipos por un sobrevoltaje
- Podrían dañarse por algún golpe.

<u>Consecuencias</u>:

- Se perdería tiempo arreglando o comprando nuevos equipos
- Se podría perder la información
- Retraso del proyecto

REDUCCIÓN:

- Tener protegidos todos los equipos para evitar un sobrevoltaje o un golpe
- Tener backups de la información en otros equipos para no retrasar el proyecto.
- Firmar convenios con alguna empresa para arreglar los equipos en caso de daño.

SUPERVISION:

- El estado de los equipos
- El voltaje de trabajo de los equipos sea el adecuado
- El equipo técnico use correctamente las máquinas

GESTIÓN:

- Arreglar los equipos dañados y recuperar la información perdida.
- Asignar nuevas máquinas para que el equipo técnico siga con su trabajo

ESTADO ACTUAL:

Fase de reducción iniciada	X
Fase de Supervisión iniciada	
Gestionando el riesgo	

RESPONSABLES: Miguel Fuertes, Juan Guevara

Tabla V.XXV: Gestión de Riesgo 1

<table>
<tr><td colspan="4">HOJA DE GESTIÓN DEL RIESGO</td></tr>
<tr><td colspan="2">ID. DEL RIESGO: R5</td><td colspan="2">FECHA 30/septiembre/2010</td></tr>
<tr><td>Probabilidad:
Media
Valor: 2</td><td>Impacto: Alto
Valor: 3</td><td>Exposición:
Alta
Valor: 6</td><td>Prioridad: 4</td></tr>
<tr><td colspan="4">DESCRIPCIÓN: Podría producirse la deserción de miembros del personal de desarrollo</td></tr>
<tr><td colspan="4">REFINAMIENTO:
<u>Causas</u>:
• Problemas internos entre integrante del equipo técnico
• Salarios no se pagan puntualmente
• Problemas en la salud.
• <u>Consecuencias</u>:
• El proyecto podría aumentar en costo y esfuerzo del equipo técnico.
• Retraso del proyecto</td></tr>
<tr><td colspan="4">REDUCCIÓN:
• Cumplir con los salarios puntualmente.
• Tener personal capacitado para reemplazar a la persona que abandono el proyecto sin que este se vea afectado.</td></tr>
<tr><td colspan="4">SUPERVISION:
• Actitud de los miembros del proyecto
• Relaciones interpersonales</td></tr>
<tr><td colspan="4">GESTIÓN:
• Reemplazar a la persona que abandono el proyecto por otra con la misma o mayor capacidad.
• Redistribuir el trabajo a los integrantes del equipo técnico para su mayor optimización.</td></tr>
<tr><td colspan="4">ESTADO ACTUAL:
Fase de reducción iniciada</td></tr>
</table>

	Fase de Supervisión iniciada X Gestionando el riesgo:
RESPONSABLES: Miguel Fuertes, Juan Guevara	

Tabla V.XXVI: Gestión de Riesgo 5

HOJA DE GESTIÓN DEL RIESGO			
ID. DEL RIESGO: R8		**FECHA** : 4/octubre/2010	
Probabilidad: Media **Valor: 2**	**Impacto:** Alto **Valor: 3**	**Exposición:** Alta **Valor: 6**	**Prioridad: 4**
DESCRIPCIÓN: Oferta de nuevos productos con funcionamiento similar y menor costo			
REFINAMIENTO: **Causas:** • La gran cantidad de productos software que día a día aparecen en el mercado. **Consecuencias:** • El cliente podría preferir adquirir ese producto y cancelar el negocio. • Deberíamos aumentar considerablemente la calidad de nuestro producto o buscar nuevos clientes.			
REDUCCIÓN: • Firmar un contrato claro y específico con la finalidad que el cliente no pueda cancelar el negocio. • El precio del producto debe estar de acorde a la complejidad del mismo. • Mantener una buena relación con el cliente.			
SUPERVISION: • La calidad del software • La relación cliente – desarrollador			

GESTIÓN:
• Hacer respetar el contrato vigente para que no nos produzca pérdidas económicas. • Buscar otros clientes que necesiten del software desarrollado.
ESTADO ACTUAL: Fase de reducción iniciada Fase de Supervisión iniciada X Gestionando el riesgo:
RESPONSABLES: Miguel Fuertes, Juan Guevara

Tabla V.XXVII: Gestión de Riesgo 8

HOJA DE GESTIÓN DEL RIESGO			
ID. DEL RIESGO: R13		**FECHA**: 8/octubre/2010	
Probabilidad: Media **Valor: 2**	**Impacto:** Alto **Valor: 3**	**Exposición:** Alta **Valor: 6**	**Prioridad: 4**
DESCRIPCIÓN: El personal de desarrollo podría detener temporalmente sus labores por feriados o vacaciones			
REFINAMIENTO: **<u>Causas</u>:** • La gran cantidad de feriados que tiene nuestro país. • Los desarrolladores foráneos viajan a sus casas a reunirse con sus familias. **<u>Consecuencias</u>:** • El proyecto se retraza. • El trabajo se acumula y los desarrolladores se verían obligados a trabajar más rápido.			
REDUCCIÓN: • Utilizar los tiempos libres para avanzar en el desarrollo de la aplicación.			

• Tener un plan de actividades bien definido.
SUPERVISION: • El trabajo de los desarrolladores. • Cumplimiento del plan de trabajo.
GESTIÓN: • Distribuir el trabajo entre los distintos desarrolladores. • Recuperarlos días perdidos trabajando en jornadas más largas.
ESTADO ACTUAL: Fase de reducción iniciada X Fase de Supervisión iniciada Gestionando el riesgo:
RESPONSABLES: Miguel Fuertes, Juan Guevara

Tabla V.XXVIII: Gestión de Riesgo 13

HOJA DE GESTIÓN DEL RIESGO			
ID. DEL RIESGO: R6		**FECHA**: 10/octubre/2010	
Probabilidad: Baja **Valor: 1**	**Impacto:** Critico **Valor: 4**	**Exposición:** Media **Valor: 4**	**Prioridad: 5**
DESCRIPCIÓN: El producto final no llena las expectativas de la empresa solicitante.			
REFINAMIENTO: **<u>Causas</u>:** • Los requerimientos fueron mal entendidos. • El cliente no detallo correctamente todos los requerimientos. **<u>Consecuencias</u>:** • El cliente puede cancelar el negocio. • Se tiene que corregir el funcionamiento del software.			
REDUCCIÓN: • Detallar correctamente el SRS para que nos sirva como un respaldo			

legal por si el cliente presenta otras necesidades al final del proceso de desarrollo. • Presentar constantemente avances del proyecto para saber si el cliente esta satisfecho con e mismo.
SUPERVISION: • El cumplimiento de los requerimientos
GESTIÓN: • Corregir el funcionamiento del software.
ESTADO ACTUAL: Fase de reducción iniciada Fase de Supervisión iniciada X Gestionando el riesgo:
RESPONSABLES: Miguel Fuertes, Juan Carlos Guevara

Tabla V.XXIX: Gestión de Riesgo 6

HOJA DE GESTIÓN DEL RIESGO			
ID. DEL RIESGO: R10		**FECHA**: 2/octubre/2010	
Probabilidad: Baja **Valor: 1**	**Impacto:** Critico **Valor: 4**	**Exposición:** Media **Valor: 4**	**Prioridad: 5**
DESCRIPCIÓN: Fallas en la base de datos o en las conexiones a la misma			
REFINAMIENTO: **<u>Causas</u>:** • La falta de seguridad en la base de datos • Las conexiones entre la base de datos y la aplicación están mal diseñadas. **<u>Consecuencias</u>:** • La aplicación podría no trabajar correctamente. • El desarrollador debería corregir los errores implicando más			

trabajo.
REDUCCIÓN: • Al momento de diseñar la base de datos dar las seguridades necesarias a las mismas. • El tipo de conexión debe ser el más adecuado según las herramientas que se están utilizando y el tipo de aplicación que se esta desarrollando.
SUPERVISION: • El diseño de la base de datos • La seguridad en la base de datos. • Las conexiones entre la aplicación y la base de datos.
GESTIÓN: • Analizar las conexiones actuales y corregirlas o implementar nuevas conexiones. • Solicitar ayuda a personas expertas en diseño y seguridades de bases de datos
ESTADO ACTUAL: Fase de reducción iniciada Fase de Supervisión iniciada X Gestionando el riesgo:
RESPONSABLES: Miguel Fuertes, Juan Carlos Guevara

Tabla V.XXX: Gestión de Riesgo 10

HOJA DE GESTIÓN DEL RIESGO			
ID. DEL RIESGO: R11		**FECHA**: 20/octubre/2010	
Probabilidad: Media **Valor: 2**	**Impacto:** Moderado **Valor: 2**	**Exposición:** Media **Valor: 4**	**Prioridad: 5**

DESCRIPCIÓN: La aplicación final podría no ser compatible con algún navegador
REFINAMIENTO: **<u>Causas</u>:** • Es diseñada para trabajar con un solo navegador • La empresa solicitante decide utilizar otro navegador en sus máquinas. • Aparecen nuevas versiones de navegadores. **<u>Consecuencias</u>:** • Se deberá desarrollar nuevamente la aplicación. • Se extenderá el tiempo de entrega de la aplicación.
REDUCCIÓN: • Nuestra aplicación debe ser desarrollada con la finalidad que sea escalable y se adapte a cualquier tipo de entorno.
SUPERVISION: • La escalabilidad de la aplicación. • La programación orientada a objetos.
GESTIÓN: • Analizar los navegadores para los que la aplicación trabajará sin ningún problema. • Crear la aplicación basándose en estándares de diseño web.
ESTADO ACTUAL: Fase de reducción iniciada X Fase de Supervisión iniciada Gestionando el riesgo:
RESPONSABLES: Miguel Fuertes, Juan Guevara

Tabla V.XXXI: Gestión de Riesgo 11

5.2.4. Tarjetas CRC

Estudiante	
Responsabilidades	**Colaboradores**
Mostrar Información	Tnivel, Tbeca
Modificar Datos	Administrador
Cambiar Estado Estudiante	Administrador
Ver rubros por cobrar	Test_pension, Test_uniforme, Test_libro, Test_tarea,Test_rubroespecial

Tabla V.XXXII: CRC Estudiante

Pensión	
Responsabilidades	**Colaboradores**
Asignar valor a pensiones	Administrador
Asignar a cada periodo una pensión	Administrador
Listar pensiones de los diferentes periodos	Tperiodo_pension, Tperiodo, Tpension
Modificar aranceles a la pensión	Administrador

.**Tabla V.XXXIII:** CRC Pensión

Estudiante Pensión	
Responsabilidades	**Colaboradores**
Ver estudiantes deben pensión	Test_pension, Testudiante
Mostrar datos deuda	Test_pension,Testudiante,Tabono
Cobrar Pensión	Secretaria
Cobrar Abono	Secretaria
Mostrar pensiones canceladas	Test_pension, Testudiante

Tabla V.XXXIV: CRC Estudiante Pensión

Uniforme	
Responsabilidades	**Colaboradores**
Ingresar Uniforme, talla, valor y stock.	Administrador
Mostrar Uniformes en stock	Tuniformetalla

Tabla V.XXXV: CRC Uniforme

Estudiante Uniforme	
Responsabilidades	**Colaboradores**
Listar uniformes en Stock	Test_uniforme, Tuniforme_talla
Vender uniforme	Secretaria
Cobrar Abonos Uniforme	Secretaria
Mostrar uniformes cancelados y por cancelar	Test_uniforme,Tuniforme_talla, Tabono

Tabla V.XXXVI: CRC Estudiante Uniforme

Libro	
Responsabilidades	**Colaboradores**
Ingresar Libros	Administrador
Modificar Información Libros	Administrador
Listar Libros	Tlibros

Tabla V.XXXVII: CRC Libro

Estudiante Libro	
Responsabilidades	**Colaboradores**
Ver Libros disponibles	Tlibros
Vender Libro	Secretaria
Cobrar Abonos Libro	Secretaria
Mostrar libros cancelados y por cancelar	Tlibros_estudiante, Tlibro, Tabono

Tabla V. XXXVIII: CRC Estudiante Libro

Rubro Especial	
Responsabilidades	**Colaboradores**
Ingresar Rubro Especial	Administrador
Modificar Información Rubro Especial	Administrador
Listar Rubros Especiales	Trubro_especial

Tabla V.XXXIX: CRC Rubro Especial

Estudiante Rubro Especial	
Responsabilidades	**Colaboradores**
Ver lista de Rubros especiales	Trubro_especial
Asignar rubro especial	Secretaria
Cobrar Abonos Rubro Especial	Secretaria
Mostrar rubros cancelados y por cancelar	Test_rubroespecial, Trubro_especial

Tabla V.XL: CRC Estudiante Rubro Especial

Transporte y Alimentación	
Responsabilidades	**Colaboradores**
Crear Servicio	Administrador
Modificar Información Servicio	Administrador
Listar Servicios	Ttransporte_alimentacion

Tabla V.XLI: CRC Transporte y Alimentación

Estudiante Transporte Alimentación	
Responsabilidades	**Colaboradores**
Listar servicios proporcionados por la institución	Ttransporte_alimentacion
Asignar el servicio	Secretaria
Cobrar Abonos Servicio	Secretaria
Mostrar servicios cancelados y por cancelar	Test_alitrans, Ttransporte, Tabono

Tabla V.XLII: CRC Estudiante Transporte y Alimentación

Tarea dirigida	
Responsabilidades	**Colaboradores**
Crear tarea dirigida	Administrador
Modificar tareas dirigidas	Administrador
Listar tareas dirigidas	Ttarea_dirigida

Tabla V.XLIII: CRC Tarea dirigida

Estudiante tarea dirigida	
Responsabilidades	**Colaboradores**
Ver tareas dirigidas	Ttarea_dirigida
Asignar tarea dirigida	Secretaria
Cobrar abonos tarea dirigida	Secretaria
Mostrar tareas dirigidas cancelados y por cancelar	Ttarea_dirigida,Tabono

Tabla V.XLIV: CRC Estudiante tarea dirigida

5.3. Codificación

Después de un estudio minucioso y debido a las enormes ventajas que ofrece el framework Symfony se determinó utilizarlo para el desarrollo del Sistema de Cobro de Rubros (SISCRUB) del Liceo de Talentos Stephen Hawking, complementando el mismo con una base de datos desarrollada en MySql.

5.3.1. Diseño e implementación de la base de datos

En base a las reuniones sostenidas con el personal administrativo de la institución educativa se acordaron los requerimientos funcionales del sistema y fundamentado en los mismos se realizo el diseño de la base de datos, la misma consta de 27 tablas adecuadamente relacionas, las mismas fueran implementadas a lo largo del ciclo de vida del software.

Todo el proceso de creación realizo mediante líneas de comando y conto con la colaboración del cliente, obteniendo el siguiente diseño:

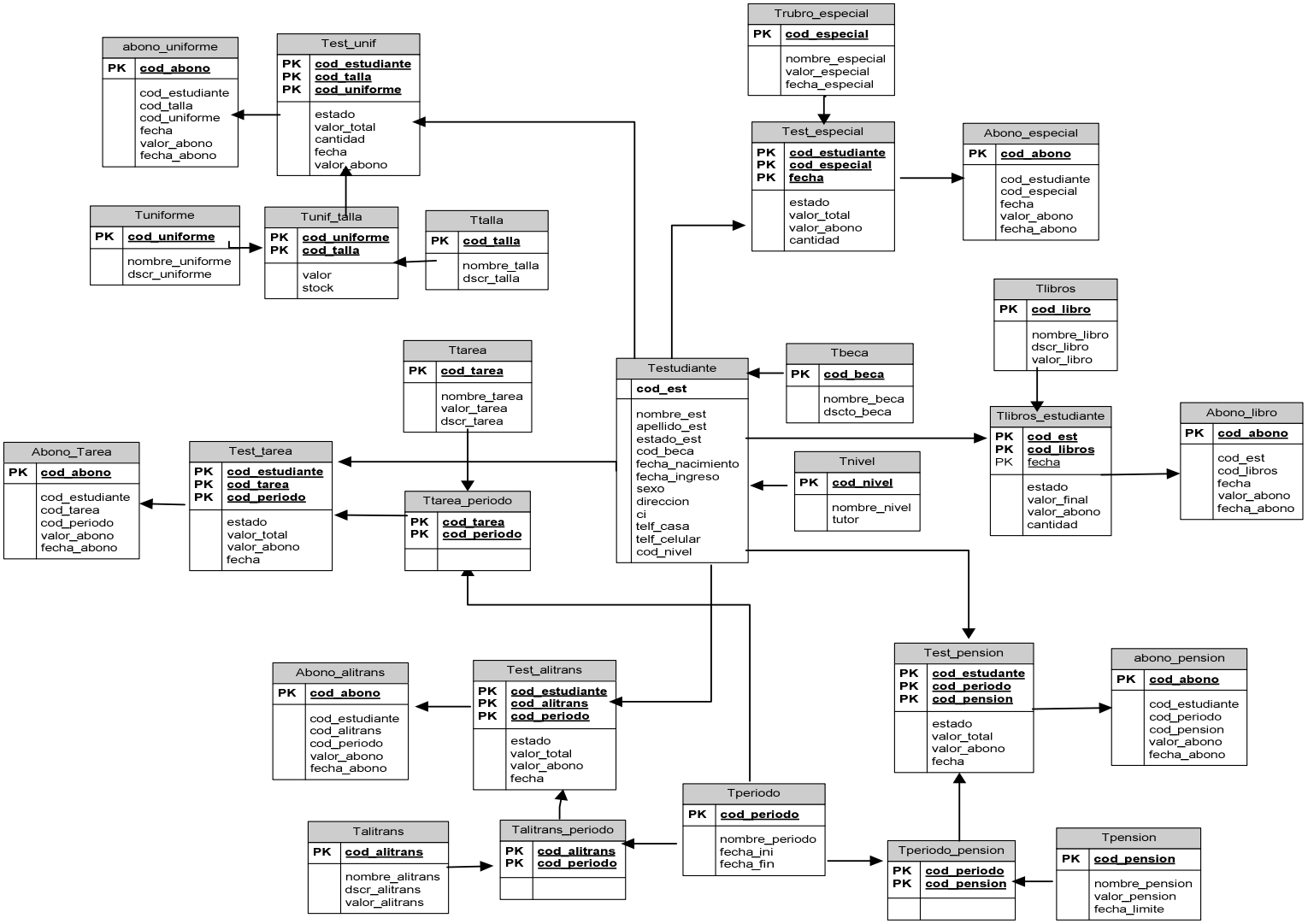


Ilustración V.19: Esquema de la base de datos Siscrub

5.3.2. Implementación de la Aplicación Web

El desarrollo de la aplicación web se encuentra divida en iteraciones y se fundamenta en el patrón de diseño MVC. Cada iteración se compone de tres fases: diseño, implementación y pruebas.

5.3.3. Iteración 1

5.3.3.1. Diseño

En base a las historias de usuarios asignadas a este módulo se determinaron las tablas a crear, estas son: Testudiante, Tnivel, Tbeca, Tperiódo.

Conjuntamente con el cliente se realizó un bosquejo de las interfaces de usuario a desarrollar, y sus respectivas funcionalidades entre las que destacamos:

El sistema debe permitir:

- Ingresar, modificar, eliminar y listar datos de usuario
- Crear, modificar, eliminar y listar niveles
- Crear, modificar, eliminar y listar becas
- Crear, modificar, eliminar y listar períodos
- Asignar una beca a un estudiante.
- Asignar estudiante a un nivel.

Un ejemplo del prototipo de interfaces diseñado en el siguiente, el mismo permite el ingreso de datos de un nuevo estudiante al mismo tiempo que asigna una beca y un nivel.

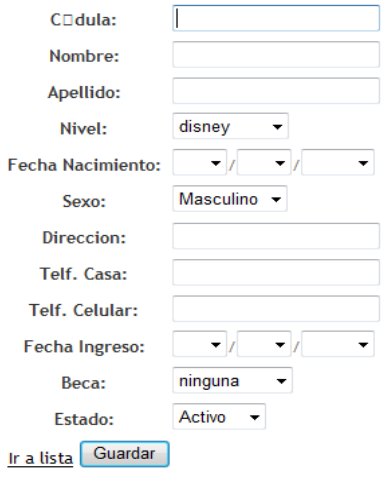


Ilustración V.20: Prototipo de interfaz el ingreso de un estudiante

5.3.3.2. Implementación

Se crearon las tablas definidas en la fase de diseño por medio de línea de comandos. La tabla Testudiante se creó de la siguiente manera:

```
CREATE TABLE IF NOT EXISTS `testudiante` (
`cod_est` int(11) NOT NULL auto_increment,
`nombre_est` varchar(50) NOT NULL,
`apellido_est` varchar(50) NOT NULL,
`estado_est` varchar(1) NOT NULL,
`cod_beca` int(11) NOT NULL default '0',
`fecha_nacimiento` date NOT NULL,
`fecha_ingreso` date NOT NULL,
`sexo` varchar(1) NOT NULL,
`direccion` varchar(100) NOT NULL,
`ci` varchar(10) NOT NULL,
`telf_casa` varchar(9) NOT NULL,
`telf_celular` varchar(9) NOT NULL,
`cod_nivel` int(11) NOT NULL,
  PRIMARY KEY  (`cod_est`)
)   ENGINE=InnoDB       DEFAULT   CHARSET=latin1
AUTO_INCREMENT=4 ;
```

El mismo proceso se realizó para todas las tablas involucradas en la iteración y sus respectivas relaciones.

Con la ayuda de Symfony se crearon los módulos necesarios para cumplir con las funcionalidades planteados en la fase de diseño. Cada módulo se fundamenta en el patrón Modelo Vista Controlador y su proceso de desarrollo es similar al descrito en el manual de Implementación con Symfony.

En esta iteración se inicio en el proceso de desarrollo con la metodología MVC por lo que tomo tiempo adaptarse a la nueva forma de programación y observar el verdadero potencial que posee la misma. Además se desarrollo el layout del sistema con la ayuda de estilos.

5.3.3.3. Pruebas

Para finalizar el desarrollo de la presente iteración se procedió a la realización de diferentes tipos de prueba como son: de integración, validación y sistema.
En lo referente a la integración se pudo observar que los diferentes módulos que compatibles con las distintas partes del sistema, al mismo tiempo se demostró el cumplimiento de los requisitos inicialmente planteado.
En las pruebas del sistema se pudo determinar que gracias al framework Symfony los diferentes módulos que componen a iteración cumplen con varias características importantes como son: rendimiento, robustez, seguridad, usabilidad.
No se presentaron mayores errores en esta iteración.

5.3.4. Iteración 2

5.3.4.1. Diseño

Tablas a implementar:

Tuniforme, Ttalla, Tunif_talla, Tlibro, Trubro_especial, Tperíodo_pension

Interfaces a Implementar:

- Ingresar, modificar y listar Uniformes con sus respectivas tallas, stock y precios.
- Ingresar, modificar y listar libros son sus respectivos precios
- Ingresar, modificar y listar rubros especiales
- Asignar pensiones a cada período

5.3.4.2. Implementación

Para la implementación de esta iteración se procedió a crear las tablas descritas en la fase de diseño, al mismo tiempo que se desarrollaron métodos en los respectivos objetos con el fin de proporcionar una funcionalidad adecuada a la aplicación. Entre los métodos más importantes que se implementaron se encuentra:

- reducirstock(), evento del objeto Test_unif utilizado para reducir el valor de stock al momento de vender un uniforme.
- getnombreperiodo(), evento del objeto Tperiodo_pension utilizado para obtener el nombre del periodo y poder generar una lista de los mismos.

- getnombrepension(), evento del objeto Tperiodo_pension utilizado para obtener el nombre de la pension.
- getpensionfinal(), evento del objeto Testudiante utilizado para calcular el valor final de las pensiones a cancelar por los estudiante que poseen becas.

De la misma manera se procedió a crear un método en la parte del controlador que consiste en que al momento que el administrador asigna una pensión a un periodo, se crea un rubro para todos los estudiantes con el valor de esta pensión.

5.3.4.3. Pruebas

Luego de realizar los diferentes tipos de pruebas en los módulos que componen esta iteración se pudo determinar algunas fortalezas como seguridad, robustez y usabilidad. Así mismo con la ayuda del cliente se determinaron varios problemas tanto al nivel de la aplicación como de la base de datos entre lo que destacamos:

- No existía un campo en la base de datos que almacenara el valor total a pagar y el valor abonado.
- La pensión se asignaba a todos los estudiantes, incluyendo estudiantes en estado inactivo.

Otro de los inconvenientes tuvo con ver con el diseno de interfaces, las mismas que se sometieron a un proceso de refinamiento para adaptarse a las necesidades del cliente.

La iteración tuvo que extenderse 1 día con la finalidad de solucionar estos inconvenientes.

5.3.5. Iteración 3

5.3.5.1. Diseño

Tablas a implementar

Talitrans, Ttarea, Test_tarea, Ttarea_pension, Tabono_tarea

Interfaces a implementar

- Ingresar, modificar y listar servicios de alimentación y transporte.
- Ingresar, modificar y listar tareas.

- Asignar servicios de alimentación y transporte a estudiantes.
- Mostrar información de deudas y abonos.
- Cobrar por servicios contratado.

5.3.5.2. Implementación

Lo más destacado en esta iteración es la implementación del primer módulo orientado a la venta o contratación de servicios y al cobro de los mismos. Las interfaces desarrolladas permiten un manejo ágil de transacciones, al mismo tiempo que facilitan el manejo detallado de todos los pagos realizados por un estudiante.

En la parte del modelo se implementaron varios eventos entre los que destacamos:

- getvalortotal(), evento del objeto Test_alitrans que permite obtener el valor del servicio contratado por el usuario y asignarlo a su información.
- getpagototal(), evento del objeto Tet_alitrans que realiza el pago total de un arancel y cambia el estado de la deuda a pagado.

Se implementaron dos vistas adicionales, una orientada a ver todas las deudas y otra para ver todos los rubros cancelados. En esta iteración se procedió a desarrollar un menú con la ayuda de la tecnología Jquery.

5.3.5.3. Pruebas

Conjuntamente con el cliente se desarrollaron todos los test necesarios para determinar la correcta funcionalidad del sistema en los ámbitos inicialmente acordados. Los errores encontrados en esta iteración son a nivel de la aplicación web, entre estos se encuentra:

- Solo se podía realizar un pago por el servicio contratado, no admitía pago de abonos.
- Al momento de visualizar la deuda, no se podía observar los abonos que se habían realizado hasta ese momento.

Para solucionar el primer inconveniente se construyo una tabla que almacene todo lo referentes abonos (valores y fechas), de esta manera se llevara un control optimo de los abonos realizados e una deuda. El siguiente

inconveniente fue resuelto con la ayuda de jquery, ya que mediante este podemos visualizar en la página de mostrar datos, una capa, la misma que almacena los datos de los abonos de una determinada deuda.

5.3.6. Iteración 4

5.3.6.1. Diseño

Tablas a implementar

Test_especial, Test_unif, Tabono_especial, Tabono_uniforme

Interfaces a implementar

- Vender servicios especiales a estudiante
- Cobrar deudas por servicios especiales vendidos
- Ver información de deudas y abonos de servicios especiales vendidos
- Vender uniformes
- Cobrar rubros por venta de uniformes
- Ver datos de deudas y abonos de uniformes

5.3.6.2. Implementación

Se implementaron los módulos para vender uniformes y rubros especiales como son: boletos, entradas, etc. En lo referente a la venta de uniforme se procedió a generar una tabla que detalla los productos existentes en stock, junto al formulario en el que se registra una venta con la finalidad que la secretaria no cometa errores al momento de realizar la transacción.

Las funciones más destacadas implementadas en lo referente al Modelo son:

- reducirstock(), evento del objeto Tunif_talla orientado a reducir el stock al momento de realizar una venta.
- getvalortotal(), evento del objeto Test_unif orientado a obtener el valor total a cancelar por el estudiante considerando varios aspectos como cantidad y costo del uniforme.

5.3.6.3. Pruebas

La realización de test involucro la implementación de varios escenarios que se podían presentar al momento de realizar la venta y cobro de servicios, no encontrando errores funcionales. Sin embargo durante esta etapa se

identificaron varios problemas en lo referente a la utilización del patrón MVC como:

- La reducción de los productos en stock se realizaba en el controlador, no en el modelo, por lo que se violaban los principios del MVC
- El valor abonado perteneciente a la tabla Abono_uniforme, se suma al abono total de la tabla Test_uniforme en la parte del controlador y no en la del modelo.

Se realizaron las correcciones necesarias implementando los eventos en las clases correspondiente mejorando de esta manera la legibilidad de código y la reutilización del mismo. Conjuntamente con el cliente se dio de alta la versión de la aplicación.

5.3.7. Iteración 5

5.3.7.1. Diseño

Tablas a implementar:

Test_pension, Test_alitrans,Tperiodo_alitrans,Tabono_pension, Tabono_alitrans

Interfaces a implementar:

- Cobrar pensiones
- Asignar servicio de alimentación y transporte a estudiante
- Ver pensiones pagadas y pensiones por cobrar
- Ver servicios de alimentación y transportes pagados y por cobrar
- Cobrar por servicios de alimentación y transporte

5.3.7.2. Implementación

Después de crear y probar las últimas tablas de la base de datos, se dio de alta la misma. El proceso de implementación de los diferentes módulos fue similar al realizado en anteriores iteraciones. La implementación de métodos del lado del Modelo fue fundamental para cumplir con varios requerimientos como: reducir la pensión de estudiantes que tienen becas, realizar abonos, etc.

5.3.7.3. Pruebas

En esta iteración las pruebas se realizaron con éxito cumpliendo todos los requerimientos funcionales, y de diseño.

5.4. Pruebas

A lo largo de las diferentes iteraciones que componen el ciclo de de vida de la aplicación se realizaron varias pruebas unitarias con la finalidad de comprobar la funcionalidad de interfaces y clases individuales. La gran mayoría de estas fueron realizadas por el desarrollador.

Para realizar las pruebas de integración se agruparon varios módulos de acuerdo a las necesidades del cliente: para comprobar la funcionalidad del cobro de pensiones se realizo la integración de varios módulos empezando por crear una pensión y un período, asignar una pensión a un período y cobrar las pensiones diferenciadas por la beca a cada uno de los estudiantes de la institución. De la misma manera se realizaron las pruebas restantes.

Las pruebas de aceptación se realizaron conjuntamente con el cliente, durante varios días permitiendo descubrir algunos errores que podrían ir degradando el funcionamiento del sistema, como son las excesivas consultas a la base de datos al momento de hacer operaciones para un conjunto de objetos. Para solucionar este inconveniente se realizo un rediseño de los eventos de las clases involucradas y la llamada a los mimos.

Posterior a la corrección de errores se logro dar una cobertura total de la especificación de requisitos y del manual del usuario.

5.5. COMPROBACIÓN DE LA HIPÓTESIS

5.5.1. Análisis de resultados

Se ha realizado el análisis para demostrar la hipótesis que señala que:

“La implementación de una aplicación web bajo el patrón de diseño Modelo Vista Controlador con un lenguaje de software libre seleccionado mejorará la productividad de desarrollo”, de la tesis titulada: “Análisis del patrón Modelo Vista Controlador implementado en lenguajes de software libre para el desarrollo de aplicaciones web. Caso práctico: Liceo de Talentos Stephen Hawking”.

Se determinó que la estadística inferencial y específicamente el Chi cuadrado es el método más adecuado para este estudio, puesto que se aplica a

pequeñas poblaciones (ya que la muestra es apenas de veinte personas) y no existe un sistema anterior desarrollado con el patrón de diseño MVC.

5.4.2. Estadística Inferencial

La inferencia estadística o estadística inferencial es una parte de la Estadística que comprende los métodos y procedimientos para deducir propiedades (hacer inferencias) de una población, a partir de una pequeña parte de la misma (muestra).

La bondad de estas deducciones se mide en términos probabilísticos, es decir, toda inferencia se acompaña de su probabilidad de acierto.

5.4.3. Selección y determinación de la muestra

Se realizó una encuesta para obtener información necesaria para la comprobación de la hipótesis. Esta encuesta fue aplicada a un grupo de personas elegidas al azar que tenían conocimientos sobre patrones de diseño y MVC. Este grupo de personas pertenecen a diferentes empresas muy importantes dentro de Ecuador como son: TATA y BABELSOFT. A los participantes se les informó que la encuesta era de carácter investigativo y totalmente anónimo. El modelo de la encuesta se encuentra en los anexos.

5.4.4. Obtención de los datos

La encuesta fue aplicada a una población de veinte personas que tienen conocimientos sobre los patrones de diseño y que alguna vez los utilizaron. En este caso, se hizo énfasis en encuestar a personas con nociones en MVC. La encuesta constó de cuatro preguntas referentes a los patrones de diseño (ya que MVC es uno de ellos) y cuatro específicamente a MVC. Las preguntas de la 1 a la 4 están relacionadas con los costos del desarrollo de un proyecto y las preguntas 5 a la 8 se refieren al tiempo del desarrollo.

Las preguntas fueron las siguientes:

1. ¿Cree usted que el uso de MVC implementado en un framework reduce los costos de desarrollo de un proyecto?

2. ¿Cree usted que el uso de MVC implementado en un framework NO reduce los costos de desarrollo de un proyecto?
3. ¿Considera usted que sin el uso de MVC implementado en un framework se reducen los costos de desarrollo de un proyecto?
4. ¿Considera usted que sin el uso de MVC implementado en un framework NO se reducen los costos de desarrollo de un proyecto?
5. ¿Cree que el empleo de MVC implementado en un framework menora el tiempo de desarrollo de software personalizado?
6. ¿Cree que el empleo de MVC implementado en un framework NO menora el tiempo de desarrollo de software personalizado?
7. ¿Cree que sin emplear MVC implementado en un framework menora el tiempo de desarrollo de software personalizado?
8. ¿Cree que sin emplear MVC implementado en un framework NO menora el tiempo de desarrollo de software personalizado?

La calificación de las preguntas a las respuestas está entre uno y cinco (1-5) tomando a uno (1) como la calificación menor y cinco (5) como la calificación mayor. Sin embargo, el encuestado no está en la obligación de responder a todas las preguntas por lo que se ha tomado el valor de 0 (cero) a aquellas que no fueron respondidas.

5.4.5. Clasificación y organización de los datos

Las respuestas emitidas por los encuestados son las siguientes:

	Enunciados							
Encuestados	**Preg1**	**Preg2**	**Preg3**	**Preg4**	**Preg5**	**Preg6**	**Preg7**	**Preg8**
1	5	1	2	3	5	0	2	4
2	5	0	1	3	5	1	1	3
3	4	1	3	3	4	2	1	4
4	4	2	2	4	4	0	1	4
5	3	2	1	4	3	1	3	2
6	5	1	2	4	5	0	0	4
7	4	2	1	3	4	1	2	4

8	4	1	1	5	5	2	1	3
9	3	1	3	4	3	3	2	3
10	4	0	1	3	5	2	0	4
11	5	0	0	4	5	1	0	4
12	5	1	2	4	4	0	1	5
13	4	1	0	4	4	2	0	3
14	2	4	4	2	2	4	3	2
15	3	3	2	3	2	3	2	3
16	4	1	0	5	5	1	2	5
17	2	3	4	2	2	4	3	4
18	5	2	0	3	4	1	0	5
19	4	1	1	3	4	2	1	5
20	5	0	0	5	5	0	1	5
TOTAL	**80**	**27**	**30**	**71**	**82**	**25**	**23**	**77**

Tabla V.XLV: Respuestas a preguntas de la encuesta.

5.4.6. Análisis de los datos

Se muestran los datos de las respuestas de las encuestas tabulados en una matriz dada por filas (i) y columnas (j), para poder aplicar Chi-cuadrado con mayor facilidad.

	Reduce los costos de desarrollo de un proyecto	NO reduce los costos de desarrollo de un proyecto	**TOTAL**
El uso de MVC implementado en un framework	80	27	**107**
Sin el uso de MVC implementado en un framework	30	71	**101**
TOTAL	**110**	**98**	**208**

Tabla V.XLVI: Valores tabulados de las preguntas 1 - 4

	Menora el tiempo de desarrollo de software personalizado	NO menora el tiempo de desarrollo de software personalizado	**TOTAL**
El uso de MVC implementado en un framework	82	25	**107**
Sin el uso de MVC implementado en un framework	23	77	**100**
TOTAL	**105**	**102**	**207**

Tabla V.XLVII: Valores tabulados de las preguntas 5 - 8

Una vez obtenida la matriz de datos tabulados se procedió a calcular los *valores esperados*, a partir de los datos abstraídos, para esto se debe multiplicar los respectivos marginales y dividir por el gran total.

Para las los datos de la Tabla V. XLVIII relacionadas a las cuatro primeras preguntas:

V.E = (∑ fila (i)*∑ columna (j)) / total

$V.E_{11}$ = (107*110)/208 = 56.59

$V.E_{12}$ = (107*98)/208 = 50.41

$V.E_{21}$ = (101*110)/208 = 53.41

$V.E_{22}$ = (101*98)/208 = 47.59

Para las los datos de la Tabla V. XLIXI relacionadas a las cuatro últimas preguntas:

V.E = (∑ fila (i)*∑ columna (j)) / total

$V.E_{11}$ = (107*105)/207 = 54.28

$V.E_{12}$ = (107*102)/207 = 52.72

$V.E_{21}$ = (100*105)/207 = 50.72

$V.E_{22}$ = (100*102)/207 = 49.28

5.4.7. Valores Esperados

	Reduce los costos de desarrollo de un proyecto	NO reduce los costos de desarrollo de un proyecto	**TOTAL**
El uso de MVC implementado en un framework	56.59	50.41	**107**
Sin el uso de MVC implementado en un framework	53.41	47.59	**101**
TOTAL	**110**	**98**	**208**

Tabla V.L: Valores esperados de las preguntas 1 - 4

	Menora el tiempo de desarrollo de software personalizado	NO menora el tiempo de desarrollo de software personalizado	**TOTAL**
El uso de MVC implementado en un framework	54.28	52.72	**107**
Sin el uso de MVC implementado en un framework	50.72	49.28	**100**
TOTAL	**105**	**102**	**207**

Tabla V.LI: Valores esperados de las preguntas 5 - 8

A través de Chi-cuadrado se probará de forma afirmativa o negativa que la distribución de las frecuencias abstraídas difiere significativamente en relación a la distribución de las frecuencias que deberíamos esperar.

$$X^2 = \sum((O-E)^2 / E)$$

En donde:

O = Frecuencia o valores observados

E = Frecuencia o valores Esperados

O	E	O-E	$(O-E)^2$	$(O-E)^2/E$
80	56.59	23.41	548.03	9.68
27	50.41	-23.41	548.03	10.87
30	53.41	-23.41	548.03	10.26
71	49.28	21.72	471.76	9.57
			X^2 =	**40.39**

Tabla V.LII: Chi-Cuadrado para las primeras cuatro preguntas

O	E	O-E	$(O-E)^2$	$(O-E)^2/E$
82	54.28	27.72	768.40	14.16
25	52.72	-27.72	768.40	14.58
23	50.72	-27.72	768.40	15.15
77	49.28	27.72	768.40	15.59
			X^2 =	**59.47**

Tabla V.LIII: Chi-Cuadrado para las últimas cuatro preguntas

Para afirmar o negar la hipótesis se debe comparar el valor obtenido (40.39 y 59.47), con el chi-cuadrado crítico de la tabla de valores crítico, los parámetros que se deben tomar en cuenta son los grados de libertad (GL) y el nivel de significación (P); el primero se define como el (número de columnas-1) X (número de filas-1), en este caso es (2-1) X (2-1) = 1 para ambos casos; el nivel de significación también conocido como nivel de

confianza se refiere a la probabilidad de que los resultados observados se deban al azar, este valor es fijado por el investigador usualmente es de 5 % o 10 % (es decir 0,05 o 0,1).

Considerando P = 0,05 y GL=1, se tiene que X^2 critico es igual a 3,841 de acuerdo a la Tabla de Valores Críticos de la Distribución Chi-cuadrado estándar. Se observa que X^2 calculado 40.39 **>** 3,841 **y** 59.47 **>** 3,841.

Se afirma que existe una asociación entre las variables estudiadas, por lo tanto, se afirma que el uso de MVC implementado en un framework reduce los costos y además, menora el tiempo de desarrollo de un proyecto.

5.4.8. Representación gráfica de los resultados

A continuación se muestra gráficamente los puntajes obtenidos en la encuesta aplicada a una población de 20 personas.

El primer gráfico corresponde a los resultados de las preguntas 1 – 4 (uno a la cuatro) y el segundo de las preguntas 5 – 8 (cinco a la ocho).

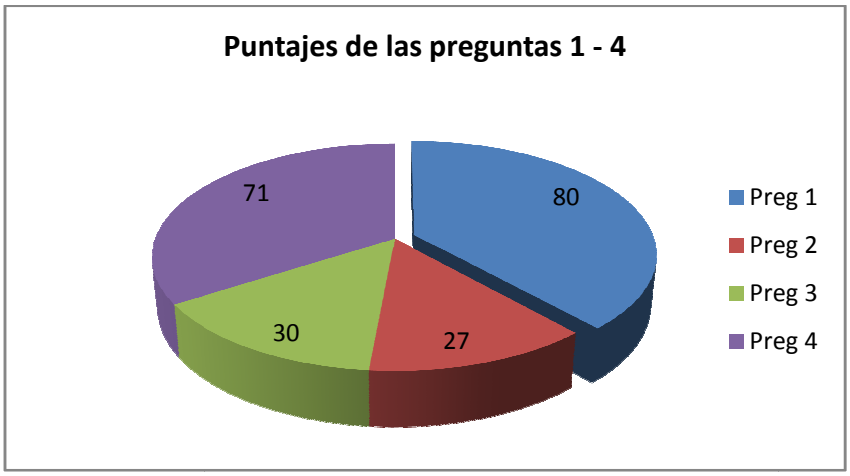


Ilustración V.21: Puntajes de las preguntas 1 - 4

Interpretación: Se puede ver que la primera pregunta tiene un total de 80 puntos es decir es la más alta, a continuación la cuarta con 71 puntos y la segunda y tercera por debajo de las mencionadas anteriormente. Estos

resultados muestran que la mayoría de encuestados piensan que el uso de MVC implementado en un framework reduce los costos de un proyecto.

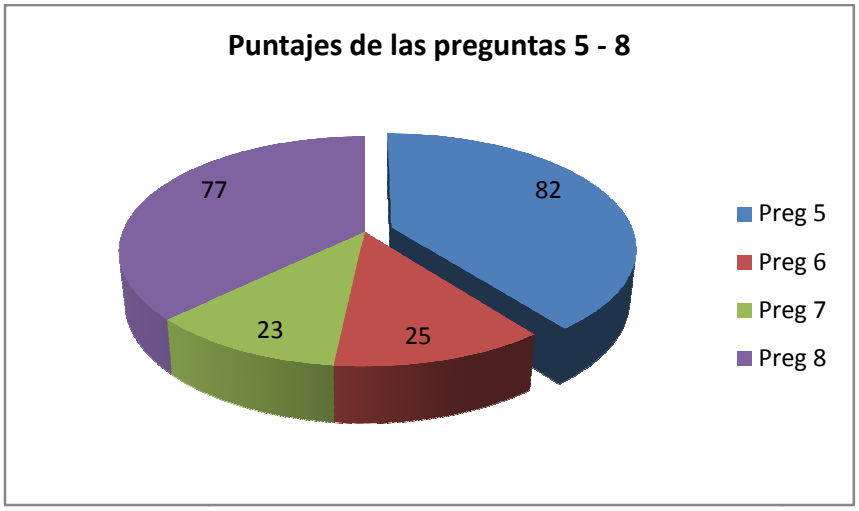


Ilustración V.22: Puntajes de las preguntas 5 - 8

Interpretación: Se puede ver que la quinta pregunta tiene un total de 82 puntos, a continuación la octava con 77 puntos y la sexta y séptima por debajo de las mencionadas anteriormente. Estos resultados hacen interpretar que la mayor parte de encuestados creen que el empleo de MVC implementado en un framework menora el tiempo de desarrollo de software personalizado.

En la encuesta aplicada, las veinte personas en algunos enunciados respondieron con cero a lo que se le va a llamar “ausencia de votante” por lo tanto a continuación se va a detallar el número de votantes y los puntajes obtenidos.

	Enunciados							
	Preg 1	**Preg 2**	**Preg 3**	**Preg 4**	**Preg 5**	**Preg 6**	**Preg 7**	**Preg 8**
Votantes	20	16	15	20	20	14	14	20

Tabla V.LIV: Número de "votantes" por cada pregunta.

Las preguntas 1, 4, 5 y 8 fueron respondidas por todos los votantes, seguida de la segunda y tercera que la responde casi todos y las demás

preguntas son respondidas por 14 votantes. A continuación se muestra una gráfica de los votantes y puntajes obtenidos.

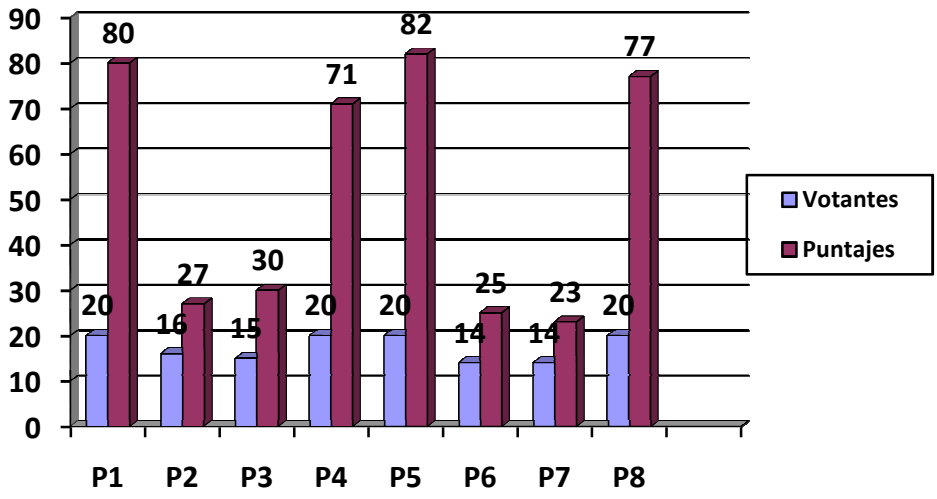


Ilustración V.23: Relación del número de votantes y el puntaje obtenido

Es importante tomar en cuenta que la columna de puntaje que es mucho mayor a la de los votantes porque los votantes valoraron con puntajes altos a las preguntas, por lo cual las columnas que tienen tamaño similar es porque recibieron el mínimo de calificación y en algunos casos sin calificación.

5.4.9. Validación de la hipótesis

La implementación de una aplicación web utilizando un patrón de diseño como es MVC tiene muchas ventajas sobre todo en lo que se refiere al costo del proyecto y al tiempo de desarrollo. El costo del proyecto disminuye debido a que MVC identifica funciones propias del sistema y esto hace que sea más fácil el mantenimiento e implementación de requerimientos nuevos. Además, MVC intenta reutilizar, en lo posible, el código que es más frecuentemente creado como son métodos básicos de

interacción con una base de datos. MVC presenta una estructura y un diseño que se puede tomar como referencia para desarrollar proyectos en menos tiempo. Desarrollar un proyecto en menos tiempo se traduce a menor costo ya que emplea menos esfuerzo de los desarrolladores y menos cantidad de errores.

5.4.10. Conclusión del análisis de la hipótesis

De los resultados obtenidos, se encontró que el Chi-cuadrado calculado es mayor que el Chi-cuadrado crítico dado por defecto, por lo que se puede confirmar la validez de la hipótesis, es decir, que el resultado es efectivo por la naturaleza de los datos obtenidos.

Los resultados de las encuestas afirman que el empleo de MVC implementado en un framework reducen los costos y menoran el tiempo de desarrollo de un proyecto web; lo que implica que, la implementación de una aplicación web utilizando el patrón de diseño MVC (Modelo Vista Controlador) mejora la productividad en el desarrollo.

.

CONCLUSIONES

- Se ha determinado que MVC con frameworks Rails y Symfony crea sistemas web en menos tiempo y empleando menos recursos que el desarrollo de un sistema sin ningún patrón de diseño; a más de ello, adopta una postura de designar funciones específicas como: atender peticiones del usuario, mostrar resultados e interactuar con la base de datos. MVC crea una estructura de diseño que mejora la organización y reutilización de código, el mantenimiento y escalabilidad de una aplicación y facilita el trabajo en equipo.

- Los frameworks Rails y Symfony presentan características similares en lo referente a estructuración del proyecto, interacción con la base de datos y gestión de vistas. Symfony presenta una ligera ventaja sobre Rails debido a que está basado en php, el mismo que tiene gran acogida en el mercado y más tiempo de funcionamiento, lo que implica un mayor respaldo de información y robustez en sus productos.

- Los factores más sobresalientes que permitieron determinar la productividad en el desarrollo de software son: lenguaje y codificación, manejo de errores, tiempo de respuesta, reutilización de código, bondades del lenguaje, interacción con base de datos, ayuda y control de cambios.

- La productividad en el desarrollo es mucho mayor cuando se realiza un proyecto con un framework basado en MVC (Symfony) que sin usarlo, ya que éste presenta herramientas muy eficaces al momento de crear un nuevo proyecto. El empleo de Symfony implica tener a disposición una gama de métodos que ahorran gran cantidad de tiempo de codificación. El uso del framework requiere un alto grado de conocimiento en la herramienta. Php cumple con los objetivos de la implementación pero requiere de más tiempo de codificación y al no seguir una estructura definida, presenta conflictos de mantenimiento y escalabilidad.

- La implementación del Sistema de Cobro de Rubros (SISCRUB) mediante Symfony permite al personal administrativo un mejor control de las cuentas por cobrar de la población estudiantil así como un historial de las cuentas pagadas. El uso de Symfony optimizó el tiempo y los recursos inicialmente planteados para el desarrollo de SISCRUB, ya que aportó con herramientas para la generación de la estructura MVC, simplificó la creación de formularios, facilitó la interacción con la base de datos y la vinculación de plugins.

RECOMENDACIONES

- El patrón Modelo Vista Controlador se puede utilizar para el desarrollo de cualquier proyecto, en especial en aquellos que manejan gran cantidad de datos y transacciones complejas.
- Para facilitar el desarrollo de software mediante el patrón Modelo Vista Controlador es recomendable el uso de frameworks en este, ya que estos brindan muchas ventajas en lo referente a organización y estructuración y velocidad de desarrollo de un proyecto.
- Es recomendable llevar un historial de los proyectos según los parámetros de productividad para poder comparar con los valores de nuevos proyectos desarrollados y tener la posibilidad de mejorar las falencias en los procesos.
- Para todas las páginas, módulos y aplicaciones que presentan en sus interfaces partes similares como encabezados, pies de página, menús, es recomendable el uso de layouts. De esta manera se promueve la reutilización, se minimiza las líneas de código y facilita el mantenimiento.
- Identificar correctamente las responsabilidades que tiene tanto el Controlador como el Modelo, con la finalidad de no mezclar código que incumpla el patrón de diseño MVC.
- Las instituciones educativas del sector privado deberían implantar el Sistema de Cobro de Rubros (SISCRUB) ya que éste permite agilizar los procesos de contratación y cobro de los diferentes servicios que brindan dichas instituciones.

RESUMEN

Se analizó el patrón de diseño Modelo Vista Controlador (MVC) orientado a la implementación de un sistema web para el cobro de varios rubros denominado SISCRUB en el Liceo de Talentos Stephen Hawking con la finalidad de demostrar que la productividad en el desarrollo mediante el uso de frameworks basados en MVC se mejora notablemente.

El patrón de diseño MVC presentó una considerable ventaja para la construcción de nuevos proyectos web, debido a que el fundamento más importante de éste es la designación de funciones para cada área del sistema. Además, se observó que MVC facilita las tareas de reutilización de código, mantenimiento, escalabilidad y corrección de errores. Para comprobar estas teorías se valoró a los frameworks Rails (RoR) y Symfony mediante la codificación de un módulo del SISCRUB. Para evaluar la productividad en el desarrollo de software se definieron parámetros e indicadores en base a pruebas de implementación que se realizaron usando MVC con un framework (Symfony) y sin usarlo (Php). En consecuencia, se evaluaron los parámetros e indicadores establecidos con un carácter más real.

Se determinó que el patrón de diseño Modelo Vista Controlador implementado con software libre mejora la productividad en el desarrollo de aplicaciones web. La implantación del sistema SISCRUB es recomendable para instituciones educativas del sector privado ya que éste permite agilizar los procesos de contratación, cobro y administración de servicios como: pensiones, matrículas, uniformes, rubros especiales, alimentación, transporte y libros.

SUMMARY

It was analyzed the design pattern Model View Controller (MVC) oriented to the web system implementation for the collection of several bills called SISCRUB at the Liceo de Talentos Stephen Hawking in order to demonstrate that the productivity in the development using frameworks based on MVC is greatly enhanced.

The MVC design pattern presented a considerable advantage for the build of new web projects, because the most important foundation of this is the functions assignation for each system area. In addition, MVC facilitates code reuse tasks, maintenance, scalability and error correction. To test these theories the frameworks Rails (RoR) and symfony were valued coding a SISCRUB module. To evaluate the productivity in the software development, parameters and indicators were defined based on deployment tests. These were performed using a MVC framework (symfony) and disuse it (Php). Accordingly, the parameters and indicators evaluated were set out with a more real way.

It was determined that the design pattern Model View Controller implemented with free software improves productivity in web application development. SISCRUB system implementation is recommended for private sector educational institutions as it speeds up the process of recruitment, collection and administration of services such as pensions, fees, uniforms, special bills, food, transportation and books.

REFERENCIAS BIBLIOGRÁFICAS

Libros:

- GAMMA, Erich; HELM, Richard; y otros. **Design Patterns, Elements of Reusable Object-Oriented Software.** EEUU, Addison Wesley, 1994. pp 5-32.

- Mark Grand. **Pattern in Java, Volume 1**. Canadá, John Wiley, 2002. pp 18-25.

- POTENCIER, Fabien; ZANINOTTO, Francois. **Symfony la guía definitiva**. Traducido del inglés, EGUÍLUZ, Javier. Francia, Apress, 2008. pp 18-53

- KOPELMAN. **Administración de la Productividad en las Organizaciones.** México, Mc Graw Hill, 1986. pp 45-49.

- DAVE, Thomas; HEINEMEIER, David; y otros. **Agile Web Development with Rails.** 2.ed. Carolina del Norte-EEUU, The Pragmatic Bookshelf, 2007. pp 62-104.

- KEPNER, C. H; TREGOE, B. **El Nuevo Directivo Racional.** México, Mc. Graw-Hill, 1983. pp 57-59.

Direcciones web:

- **Ruby on Rails.**
  http://es.wikipedia.org/wiki/Ruby_on_Rails
  2010/01/05

- **Symfony.**
  http://es.wikipedia.org/wiki/Symfony

2010/01/05

- **Catalyst.**
  http://es.wikipedia.org/wiki/Catalyst_(framework)
  2010/01/05

- **Apache Struts.**
  http://es.wikipedia.org/wiki/Apache_Struts
  2010/01/05

- **Separación modelo vista controlador**.
  http://www.chuidiang.com/ood/patrones/modelo_vista_controlador.php
  2010/01/05

- **Patrones y Antipatrones.**
  http://msdn.microsoft.com/es-es/library/bb972242.aspx
  2010/02/15

- **Patrón de Diseño**.
  http://es.wikipedia.org/wiki/Patr%C3%B3n_de_dise%C3%B1o
  2010/02/15

- **Arquitectura de Software.**
  http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/rivera_l_a/capitulo2
  2010/02/12

- **¿Qué es realmente un helper?**
  http://www.ruby-forum.com/topic/150487.
  2010/08/04

- **Patrón de Diseño Modelo-Vista-Controlador**.
  http://www.utplonline.edu.ec/cursos/diretorio/apoio_5918_40589/Patr%F3n%20de%20dise%F1o%20Modelo-Vista-Controlador

2010/01/07

- **Productividad en el desarrollo de software.**
  http://www.ines.org.es/node/1485.
  2010/05/23

- **Tools for ruby on rails.**
  http://dedicatedwebserverhosting.co.uk/dedicated-server-hosting/tools-for-ruby-on-rails/
  2010/06/17

- **Calidad Sistémica y Productividad en el Desarrollo De Sistemas.**
  http://www.lisi.usb.ve/publicaciones/02%20calidad%20sistemica/Calidad%20y%20Productividad%20CISCI%202004
  2010/07/25

Anexos

ANEXO I

Implementación con Ruby

Contenido

1. Instalación

Para instalar una nueva instancia de Ruby on Rails (RoR) se necesita un servidor Apache y un gestor de base de datos como es MySql. Para ello se instaló XAMPP en la versión 1.7. A continuación se debe instalar el lenguaje Ruby en el ordenador. Existen dos métodos para instalar Ruby mediante comandos (con ficheros de código Ruby) y con el uso de ejecutables (instala Ruby y sus ficheros en la partición C: por defecto). Se utilizó el archivo ejecutable de ruby versión 1.8.

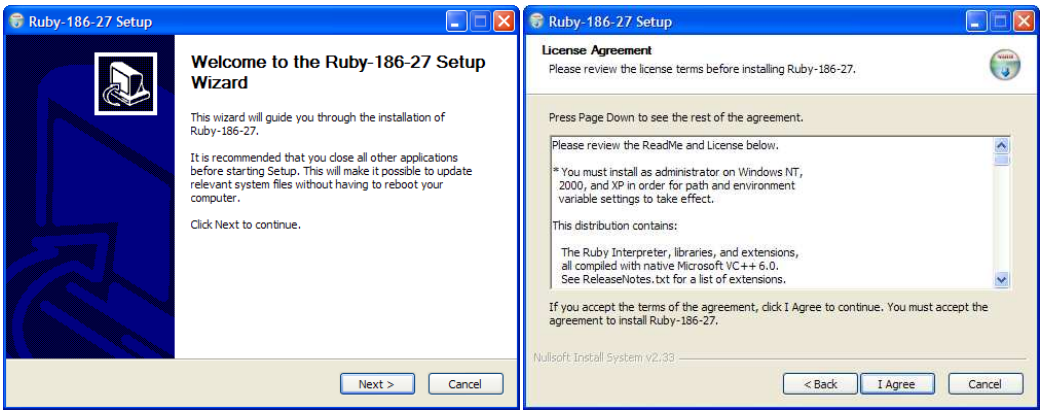


Ilustración 24: Pantallas iniciales de instalación de Ruby

En estas pantallas se da clic en siguiente y se acepta los términos de acuerdo para utilizar Ruby. En las siguientes pantallas se eligen los componentes a instalarse y el directorio que van a ocupar éstos.

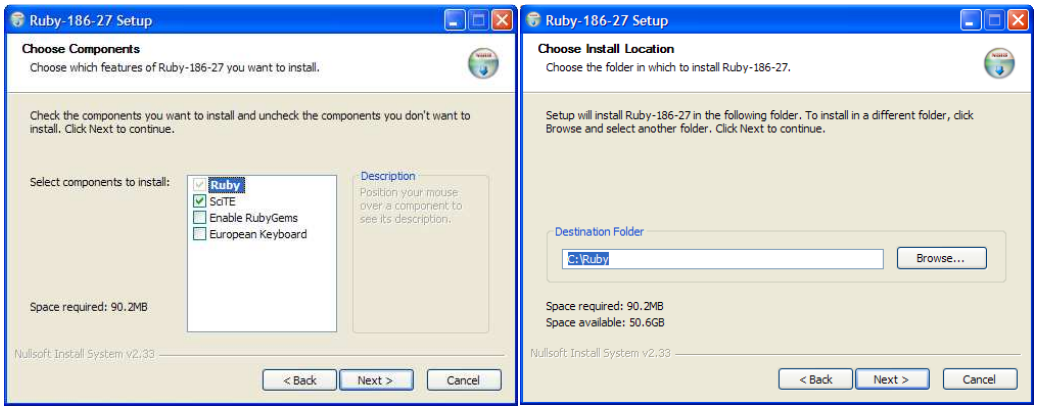


Ilustración 25: Elección de componentes y directorio de Ruby.

En las siguientes pantallas se elige el directorio del menú inicio en el que se va a mostrar Ruby. Se da clic en *Install* para empezar el proceso y luego clic en *close* para culminar con el mismo.

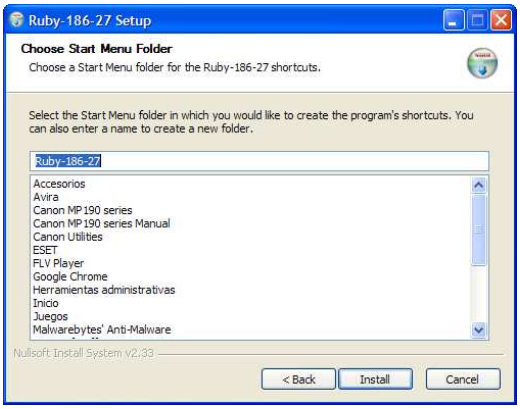


Ilustración 26: Selección de carpeta del menú inicio.

Para continuar con la instalación de Ruby on Rails se copian los archivos de *Instant Rails* en el directorio que se desee, preferentemente en “C:”. *Instant Rails* es un ambiente en tiempo de ejecución que tiene pre configurado Ruby, Rails, Apache y MySQL. No modifica el entorno del sistema. Solamente se debe ejecutar el archivo “InstantRails.exe” que está dentro de los archivos que se copiaron.

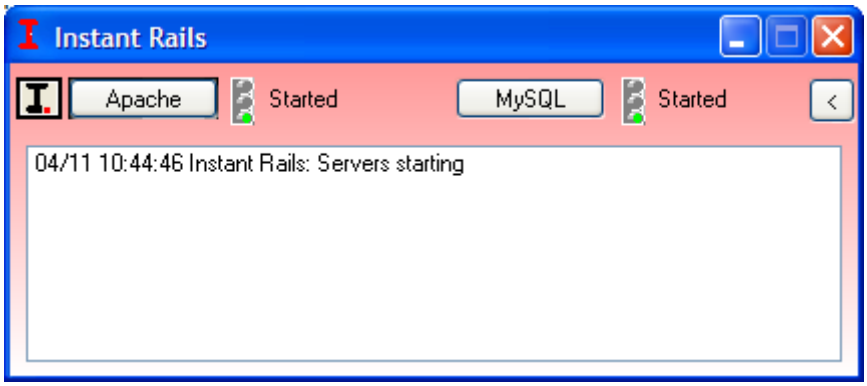


Ilustración 27: Primera pantalla de Instant Rails

Una vez ejecutado el Instant Rails, se observa claramente el inicio del servidor Apache y del gestor de base de datos MySql.

2. Creación de una aplicación web con Ruby on Rails

Para este proyecto o módulo de prueba que se desarrolará en RoR se ha tomado en cuenta sólo una parte del sistema de cobro de rubros para el Liceo de Talentos Stephen Hawking. Este módulo consta de tres tablas que son: Testudiantes, Trubrosespecials y Testrubroespecials. La finalidad de este módulo es la de realizar operaciones básicas con la base de datos para las tres tablas como son: insertar, modificar, listar y eliminar. A más de ello, la función principal del módulo es la de realizar el pago de rubros especiales por estudiante y listar las deudas pendientes de cada uno.

Para crear un nuevo proyecto en Ruby on Rails se da clic en el botón del logo de Instant Rails y se elige la opción *Rails Applications* y luego *Open Ruby Console Windows.*

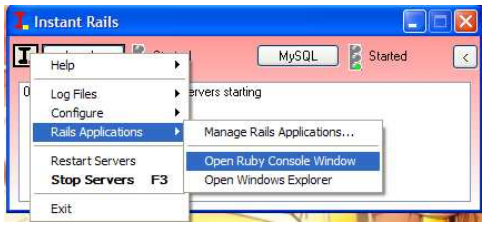


Ilustración 28: Apertura de consola de Ruby.

En ese momento se despliega la consola para escribir los comandos en Ruby para crear el proyecto nuevo.

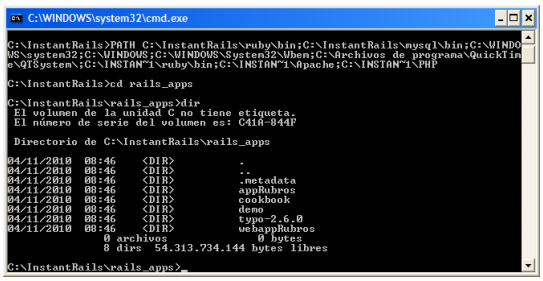


Ilustración 29: Consola de Ruby.

Para la implementación de las funciones de los estudiantes, rubros y las funciones entre ambos, se tiene: en el modelo los métodos para obtener los datos de la base de datos, en el controlador se tiene los métodos para escuchar las peticiones del cliente e interactuar con la base de datos y enviar datos a las vistas y manejar su comportamiento.

2.1. Estructura de RoR

Para generar la estructura típica de un proyecto rails, se ejecuta el comando *rails* seguido del nombre de la aplicación. En este caso se usó el nombre *webappRubros*.

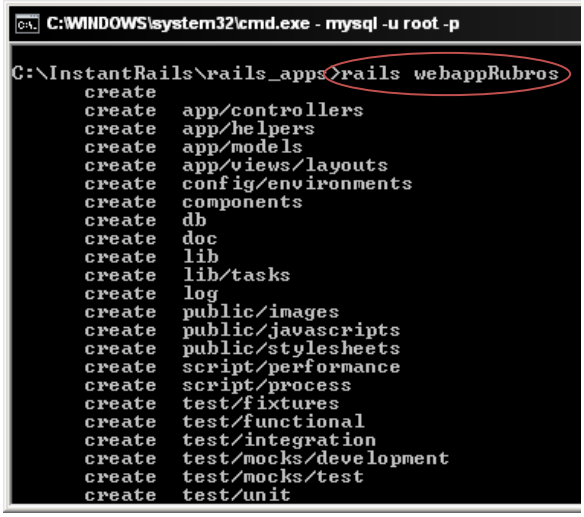


Ilustración 30: Creación de estructura en RoR

Se crean los directorios que contendrán los ficheros correspondientes al controlador, la vista y el modelo; además, se crean los ficheros de vinculación con la base de datos que se elija, etc.

El directorio creado con el comando rails presenta claramente una organización de ficheros para que en el futuro sea fácil el mantenimiento de la aplicación.

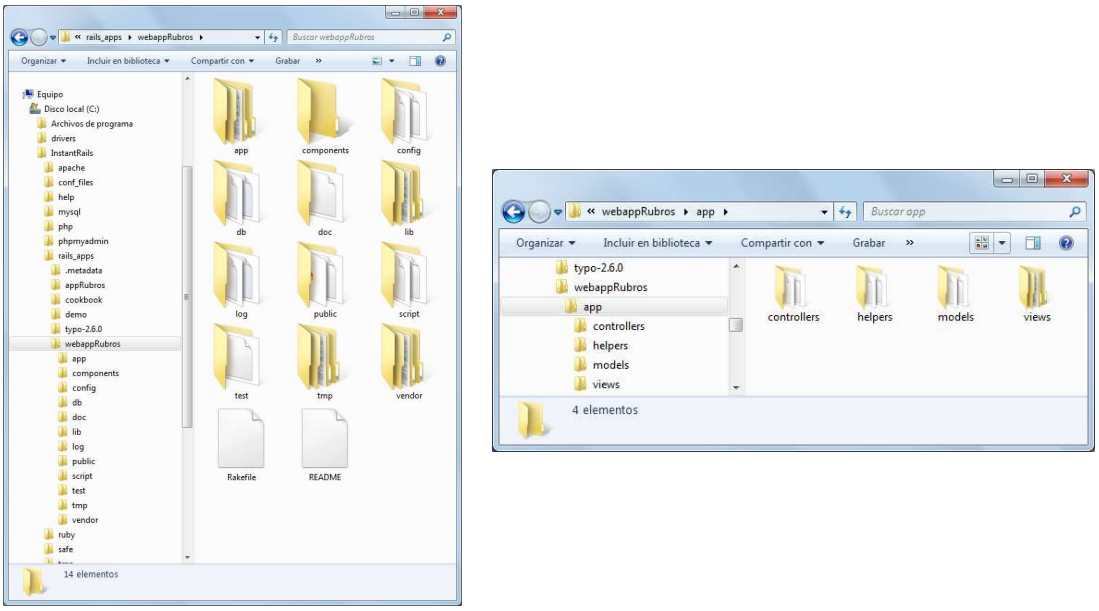


Ilustración 31: Directorio de un nuevo proyecto en RoR.

En el directorio webappRubros están todas las carpetas de la nueva aplicación. En la carpeta app se alojan los ficheros más sobresalientes correspondientes a la estructura MVC de RoR. En la carpeta de las Vistas (Views) existe una carpeta llamada Layouts que alojará

Continuando con la creación de la aplicación, se ubica en *webappRubros* dentro de la consola de Windows y ruby con el comando "cd webappRubros" para que desde allí se pueda ejecutar un comando del *script* como es el *server.* Este sirve para iniciar la aplicación web que se ha creado.

```
C:\InstantRails\rails_apps>cd webapprubros

C:\InstantRails\rails_apps\webappRubros>ruby script/server
=> Booting Mongrel (use 'script/server webrick' to force WEBrick)
=> Rails application starting on http://0.0.0.0:3000
=> Call with -d to detach
=> Ctrl-C to shutdown server
** Starting Mongrel listening at 0.0.0.0:3000
** Starting Rails with development environment...
** Rails loaded.
** Loading any Rails specific GemPlugins
** Signals ready.  INT => stop (no restart).
** Mongrel available at 0.0.0.0:3000
** Use CTRL-C to stop.
```

Ilustración 32: Inicio del servicio web del nuevo proyecto.

Se puede notar que se ha abierto el puerto 3000 para recibir peticiones a la aplicación rails que se creó. En el navegador se escribe la ruta local (127.0.0.1) seguido del puerto que va a escuchar (3000). Si rails está instalado correctamente se desplegará un mensaje como el que se tiene en la siguiente ilustración.

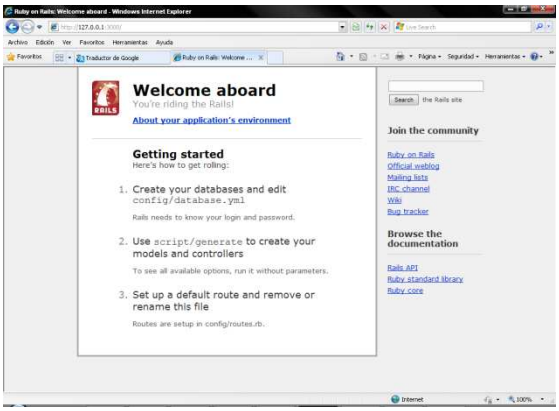


Ilustración 33: Página por defecto de un nuevo proyecto.

Para poder detener el servicio se presiona la combinación de teclas ctrl+c en la consola de Windows y ruby.

2.2. Creación de la Base de datos en MySql

Ingresamos a MySql con el usuario *root* y como no se ha configurado ninguna contraseña, se presiona *enter* al momento en que solicita el password.

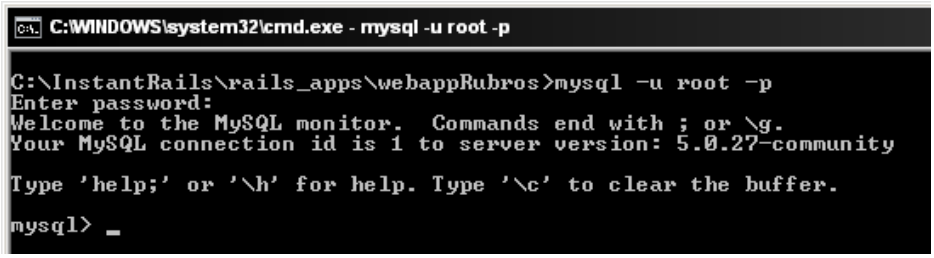


Ilustración 34: Ingreso a MySql desde la consola de ruby.

Después se crearon las bases de datos necesarias para la aplicación. Se usó una base de datos llamada *bdrubros_development* en donde se almacena toda la programación de la base de datos, en fin todo el desarrollo de la base de datos. Otra base de datos llamada *dbrubros_test* que se utiliza para realizar todas las pruebas con el sistema y que se la puede vaciar cada vez que se corra una aplicación. Y por último una base de datos llamada *dbrubros_production* que es la que la aplicación usará cuando se la ponga en línea.

Ilustración 35: Creación de una base de datos mediante comandos en la consola de ruby.

En la estructura del proyecto se ubicó el archivo *database.yml* en el directorio ..*webappRubros**config*. Con cualquier editor de texto se modifica el archivo a las siguientes condiciones:

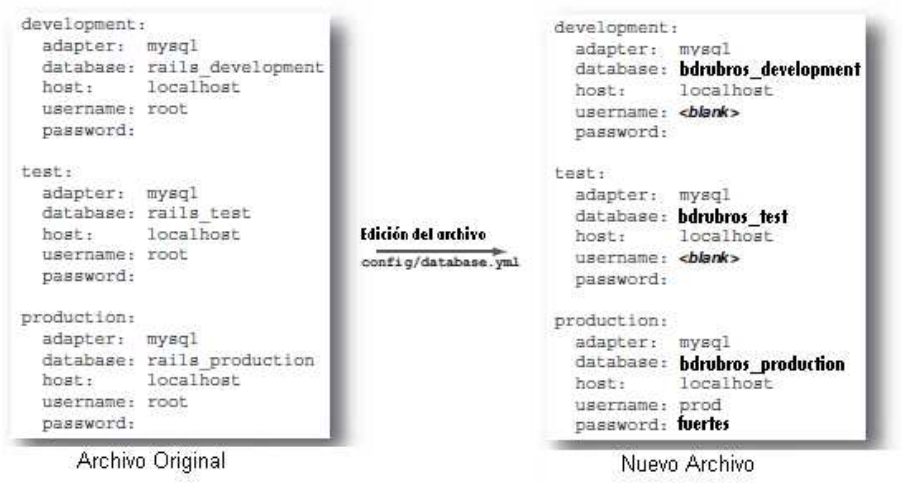


Ilustración 36: Configuración del origen de las bases de datos.

2.3. Conexión con la Base de datos

Una vez configurado el archivo *database.yml,* se configuró la aplicación para que pueda comunicarse con la base de datos. Para ello, se ejecutó un comando que se muestra en la Ilustración 37. Este genera la estructura del modelo.

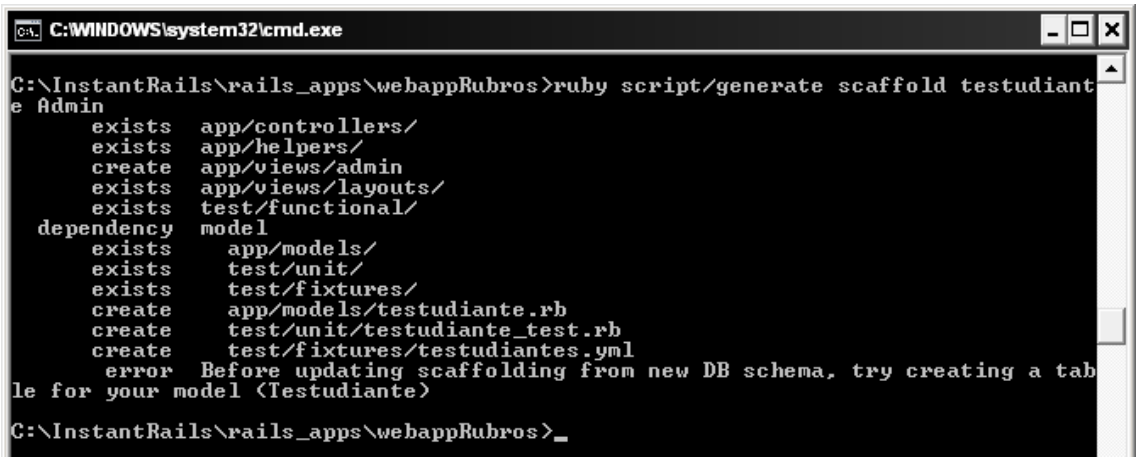


Ilustración 37: Generación de los archivos del modelo correspondientes a una tabla.

El Rails Scaffold es un framework autogenerado para manipular el modelo. Cuando se corre el *generate* o generador, lo que se hace es crear un Scaffold para un modelo en particular y que se desea acceder a éste mediante un controlador que se crea también automáticamente.

En rails un modelo es automáticamente mapeado a una tabla de una base de datos que su nombre sea el plural del nombre de la clase modelo.
Después de haber ejecutado el comando *sccafold* no sólo se han creado las estructuras correspondientes a la parte del modelo sino también, las de la vista y el controlador. Los directorios como carpetas y archivos .rb se alojan en el directorio app y se organizan según su función en las carpetas Controllers, Models y Views, como se observa en la Ilustración 31.

2.4. El Modelo

El código escrito a continuación tiene dos métodos que representan búsquedas de grupos de datos. En la primera definición hace una búsqueda de todos los registros de la tabla Testudiantes. En el segundo método se hace la búsqueda de un único registro de la tabla; se hace la búsqueda por el código o id. Estos dos métodos se han creado sólo como carácter demostrativo ya que el método find de rails es muy versátil y realiza éstos dos tipos de búsqueda y muchas otras más.
El método validates_presence_of hace las validaciones correspondientes a los campos nombre_estudiante y apellido_estudiante, las validaciones se hacen según los tipos de datos de los campos en la base de datos.

```
class Testudiante < ActiveRecord::Base
  validates_presence_of :nombre_estudiante, :apellido_estudiante

  def self.todos
    find(:all)
  end

  def self.find_unico(id)
    find(id)
  end

end
```

Los modelos correspondientes a Testudiantes y Trubro_especials no tienen métodos especiales, sólo tienen los métodos heredados de la clase base ActiveRecord. La tabla intermedia llamada Test_rubro_especials tiene los

métodos para comprobar las deudas pendientes de un estudiante, los rubros pagados para llevar un historial, el pago de un rubro y la asignación de un rubro a un estudiante.

Para asignar un rubro a un estudiante se debe seleccionar a un único estudiante y a un único rubro y poner el estado de la fila en “D” (debe). Para ello se implementó el método *inserción*. En este método se instancia una clase y se asigna los valores deseados a los atributos de ésta.

Para el pago de un rubro se debe buscar la fila que coincida con el id del estudiante y con el id del rubro especial y cambiar su estado a “P” (pagado). Para la implementación de este método se usó el comando *connection.execute* en el cual se especifica la consulta sql con los parámetros especificados (*idest* que es el id del estudiante e *idrubro* que es el id del rubro).

```
class TestRubroEspecial < ActiveRecord::Base

 def self.todos
  find(:all)
 end

 def self.pendientes(id)
  find(:all, :conditions => ["estado = 'D' and estudiante_id = ?", id])
 end

 def self.pagados(id)
  find(:all, :conditions => ["estado = 'P' and estudiante_id = ?", id])
 end

 def self.unico(idest,idrubro)
  find(:all, :conditions => ["estudiante_id = ? and rubro_especial_id = ?", idest, idrubro])
 end

 def self.insercion(idest, idrubro)
  a = TestRubroEspecial.new
  a.estudiante_id = idest
  a.rubro_especial_id =idrubro
  a.estado = "D"
  a.save
 end

 def self.pago(idest,idrubro)
  TestRubroEspecial.connection.execute('update test_rubro_especials set estado= "P"
  WHERE estudiante_id = '+ idest +' and rubro_especial_id = '+ idrubro)
 end
end
```

2.5. La Vista

El framework Rails crea interfaces muy simples cuando se ha ejecutado el comando scaffold. Las interfaces se crean a partir de los campos de las tablas a las que scaffold hace referencia. Rails diseña cuatro páginas como son: Listado general, nuevo, edición y mostrar único. En el listado pone las opciones de mostrar, eliminar y editar.

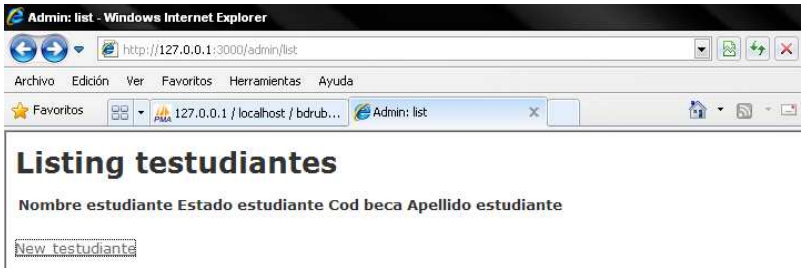


Ilustración 38: Vista del listado de estudiantes.

Como se puede observar las vistas son muy sencillas pero el framework es muy flexible en lo que respecta a la personalización. Se pueden aplicar estilos como son los layouts para evitar repetir el trabajo para cada interfaz.

El código de ruby para las vistas está entre signos de porcentaje (%). Los ficheros de las vistas de RoR tienen la extensión rhtml que es denominado "Ruby embebido".

Para el listado de estudiantes se utilizó el siguiente código:

```
<table cellpadding="5" cellspacing="0" width="100%">
<%
for testudiante in @testudiantes
%>
  <tr valign="top">
  <td>
     <img width="60" height="70" src=<%= testudiante.imagen
%>/>

  <td width="60%">
     <span class="ListTitle"><%=
h(testudiante.nombre_estudiante) %></span><br/>
```

```
      <span class="ListTitle"><%=
h(testudiante.apellido_estudiante) %></span><br/>
  </td>

  <td class="ListActions">
    <%= link_to 'Mostrar', :action => 'show', :id =>
testudiante%><br/>
    <%= link_to 'Editar', :action => 'edit', :id => testudiante
%><br/>
     <%= link_to 'Deudas', :action => 'deudas', :id =>
testudiante %><br/>
    <%= link_to 'Eliminar', { :action => 'destroy', :id =>
testudiante }, :confirm => '&iquest;Est&aacute; seguro que desea
eliminar al estudiante?', :method => :post %>
  </td>
  </tr>
  <tr><td><hr></td></tr>
<% end %>
</table>
```

Ilustración 39: Ingreso de un estudiante

Cuando se ingresa correctamente un registro se muestra un mensaje de satisfactorio con la siguiente línea de código que se crea en el controlador:

flash[:notice] = 'Los datos del estudiante fueron modificados correctamente.'

Ilustración 40: Mensaje de estudiante ingresado

Para la edición de un registro, RoR recoge los datos de éste y los muestra en casillas que tienen la posibilidad de ser modificados y ser almacenados nuevamente. El único campo que no se puede modificar es el id.

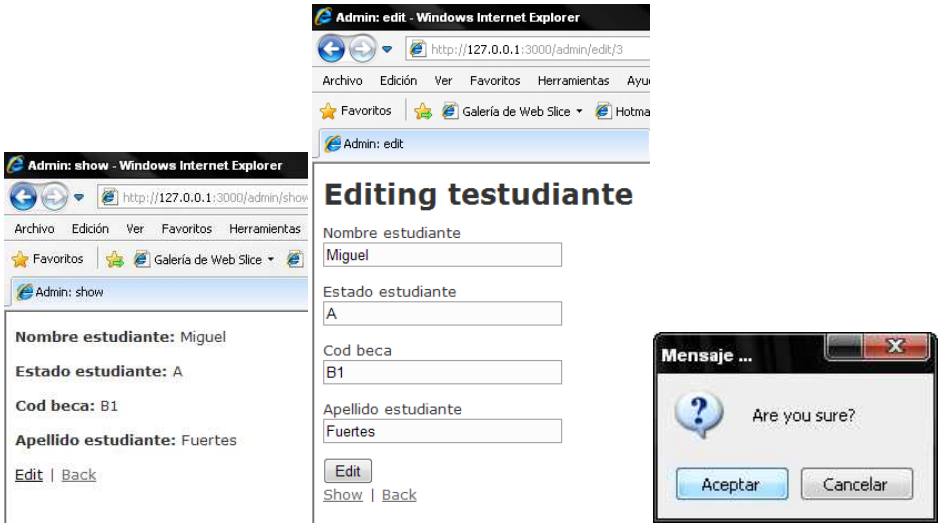


Ilustración 41: Vista de la edición de un estudiante.

Para validar los datos que se intenten ingresar en las tablas es necesario hacerlo en la parte del modelo ya que este es el encargado de los procesos para manejar los datos de la base de datos desde la aplicación.

En la aplicación que se creó se accedió al archivo del modelo correspondiente a la clase *TEstudiante* en el directorio *webappRubros\app\models.* Para validar los datos que se van a ingresar se debe editar el archivo *testudiante.rb* en el directorio mencionado. Se validó que los campos *nombre_estudiante* y *apellido_estudiante* no estén vacíos. Para ello se debió escribir una línea de código y especificar las columnas de la tabla TEstudiantes.

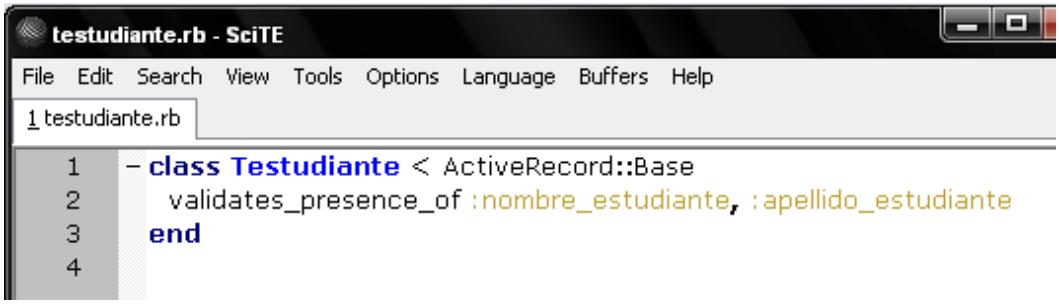


Ilustración 42: Validación de campos vacíos.

Para comprobar la funcionalidad de esta línea de código se intentó ingresar un estudiante sin llenar los campos de nombre y apellido y al dar clic en *create* automáticamente apareció un mensaje indicado la causa del error y señalando en donde se encuentra. Esto normalmente tomaría varias horas programando el código en otro lenguaje.

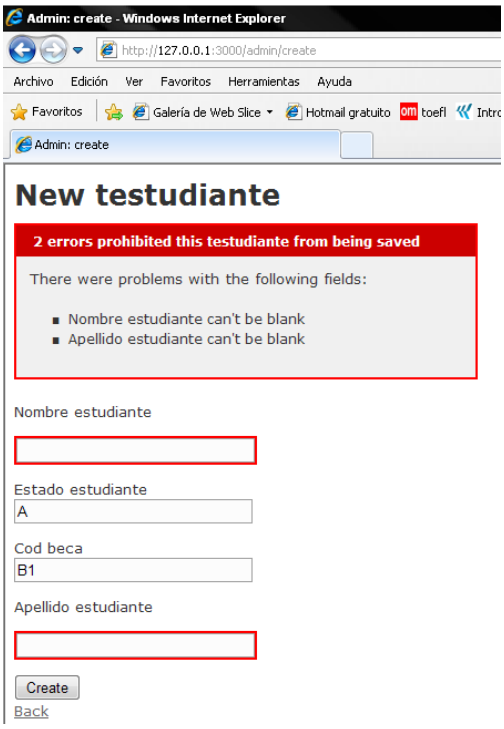


Ilustración 43: Mensaje de error en formularios con campos vacíos.

Nótese que al momento de hacer cambios de edición en el código no se tiene que reiniciar la aplicación, como es en general en los lenguajes de programación habituales. Esta característica provee una gran ventaja en la productividad ya que implica menos tiempo al momento de compilar las aplicaciones cuando se realiza pequeños cambios.

Para crear ambientes más amigables es recomendable usar los Layouts, debido a que facilitan el trabajo cuando se requieren cambios y disminuyen líneas de código. Las vistas después de aplicar Layouts quedaron de la siguiente manera:

- **Listado de Estudiantes**

Ilustración 44: Listado de estudiantes aplicando layouts.

Ingreso de estudiantes

Ilustración 45: Ingreso de un nuevo estudiante aplicando layouts.

Edición de estudiante

Ilustración 46: Edición de estudiantes aplicando layouts.

Listado de rubros especiales

Ilustración 47: Listado de rubros especiales aplicando layouts.

Ingreso de Nuevo rubro

Para los formularios de nuevos rubros se notó que el framework crea automáticamente los combobox correspondientes a los campos de las fechas de tipo DateTime.

Ilustración 48: Vista de un nuevo rubro.

Modificación de un rubro

Ilustración 49: Modificación de un rubro especial.

Pago de rubros

Para el pago de rubros se realizó vistas personalizadas y ficheros nuevos. Una vez listado el estudiante, éste tiene la posibilidad de revisar las deudas que tiene. Esta vista muestra al estudiante seguido de un listado de todas las deudas que éste tiene. Junto a la deuda, esta un vínculo para pagarla.

En la vista del listado de deudas se obtienen los datos de variables que vienen de un proceso desde el controlador. Generalmente estás variables empiezan con @. Estós comandos se usan frecuentemente para recorrer el array desde la vista:

```
<%
     for trubro_especial in @trubro_especials
%>
<% end %>
```

Dentro del cual se trabaja con los datos del array de instancias. Otro comando muy utilizado en las vistas es:

```
<%= h(trubro_especial.nombre_especial) %>
```

Este se usa para obtener el valor del atributo de una instancia. Para visualizar en pantalla algún dato se usa:

```
<%= sprintf("%0.2f", trubro_especial.valor_especial) %>
```

Ilustración 50: Listado de deudas por estudiante.

Historial de Pago

Se puede ver el historial de pagos del estudiante con las fechas en las que se canceló y el valor de los mismos.

Ilustración 51: Historial de pagos.

2.6. Controlador

Para explicar el código del controlador se considerará únicamente el controlador del estudiante.

En el controlador se definen los métodos que interactúan con el modelo para obtener los datos de la base de datos. Los métodos que se crean cuando se ejecuta el comando scaffold son: list, show, new, create, edit, update y destroy. En estos métodos se define una variable que almacena un array o un solo registro o una instancia nula de la consulta a la base de datos mediante el método find que se define en la clase base ActiveRecord.

El método find presenta muchos beneficios y es flexible a varios tipos de consultas de uno o más cláusulas. El método *find* se lo puede relacionar con la sentencia *select* del transact sql.

```ruby
class AdminController < ApplicationController
  def index
    list
    render :action => 'list'
  end

  # GETs should be safe (see http://www.w3.org/2001/tag/doc/whenToUseGet.html)
  verify :method => :post, :only => [ :destroy, :create, :update ],
         :redirect_to => { :action => :list }

  def list
    @testudiante_pages, @testudiantes = paginate :testudiantes, :per_page => 10
  end

  def show
    @testudiante = Testudiante.find(params[:id])
  end

  def new
    @testudiante = Testudiante.new
  end

  def create
    @testudiante = Testudiante.new(params[:testudiante])
    if @testudiante.save
      flash[:notice] = 'El estudiante fue ingresado exitosamente.'
      redirect_to :action => 'list'
    else
      render :action => 'new'
    end
  end

  def edit
    @testudiante = Testudiante.find(params[:id])
  end
```

En este controlador se han definido los métodos para listar las deudas de un estudiante dado su código, pagar un rubro y consultar el historial de pago. Para llamar a un método se usa la siguiente estructura:

Clase del modelo.método del modelo(*params [:*parámetros*])*

Por ejemplo, en el método *deudas* se almacena en la variable *@test_rubro_especials* los resultados del método *pendientes* de la clase modelo *TestRubroEspecial* y se le envía el parámetro id que viene desde la vista. Se define un arreglo unidimensional para alojar los resultados del proceso anterior pero con un mejor orden para controlar de mejor manera. Se hace un for para recorrer los resultados e ir almacenando en el arreglo.

```
def deudas
  @testudiante = Testudiante.find(params[:id])
  @test_rubro_especials = TestRubroEspecial.pendientes(params[:id])
  @trubro_especials = []
  for a in @test_rubro_especials
    @trubro_especials << TrubroEspecial.find(a.rubro_especial_id)
  end
end

def pagar
  a=params[:id]
  b=params[:idrubro]
  TestRubroEspecial.pago(a,b)

  @testudiante= Testudiante.find(params[:id])

  if TestRubroEspecial.pago(a,b)
    render :action => 'edit'
  else
    flash[:notice] = 'El rubro fue cancelado correctamente.'
    redirect_to :action => 'deudas', :id => @testudiante
  end
end

def historial
  @testudiante = Testudiante.find(params[:id])
  @test_rubro_especials = TestRubroEspecial.pagados(params[:id])
  @trubro_especials = []
  for a in @test_rubro_especials
    @trubro_especials << TrubroEspecial.find(a.rubro_especial_id)
  end
end
```

Para el pago de un rubro se envían los ids del estudiante y del rubro a pagar (estos datos vienen desde la vista) para realizar una consulta de un único registro que coincida con estas claves y cambiar el estado de la deuda. Para el método *historial* se hace lo mismo que con el de la *deuda* pero, en el modelo el valor a comparar en el campo *estado* es “P” (pagado).

ANEXO II

Implementación con Php

Creación de proyecto en PHP sin el paradigma MVC

El lenguaje PHP tiene la reputación de que los programas tienen todo el código mezclado, con funciones para extraer de la base de datos, las que hacen los cálculos y el código que visualiza cosas, etc. Esto conlleva algunos problemas: es complicado seguir el flujo del programa, es muy difícil reutilizar el código, y si se necesita modificar el programa para utilizar otra base de datos, puede ser muy complicado.

Para entender mejor los beneficios de MVC se a desarrollo una aplicación en PHP que no sigue este patrón de diseño: El módulo implementado consiste en el pago de rubros especiales por parte de los estudiantes. Una de las páginas del modulo muestra el listado de estudiantes deudores para este fin el código normal sin utilizar MVC es el siguiente:

```
<?php
include("conexion.php");
$link=Conectarse();
$result=mysql_query("select testudiante.id_estudiante,
testudiante.nombre_est, testudiante.apellido_est,
trubro_especial.nombre_especial,
trubro_especial.valor_especial, trubro_especial.id_especial
from testudiante INNER JOIN test_especial ON
testudiante.id_estudiante=test_especial.id_estudiante JOIN
trubro_especial ON
test_especial.id_especial=trubro_especial.id_especial WHERE
test_especial.estado='d'",$link);
     ?>


     <TABLE BORDER=1 CELLSPACING=1 CELLPADDING=1>
     <TR>
```

```
     <TD> Id Estudiante</TD>
     <TD> Nombre </TD>
     <TD> Apellido </TD>
     <TD> Id Rubro </TD>
     <TD> Rubro </TD>
     <TD> Valor </TD>
     <TD> Pagar </TD>

     </TR>
<?php
while($row = mysql_fetch_array($result)) {
printf("<tr><td> %s</td><td> %s </td><td>&nb
sp;%s </td><td> %s </td><td> %s </
td><td> %s </td><td><a
href='accionpagar.php?idest=".$row['id_estudiante']."&nombr
e=".$row['nombre_est']."&apellido=".$row['apellido_est']."&
idrubro=".$row['id_especial']."&rubro=".$row['nombre_especi
al']."&valor=".$row['valor_especial']."'>Pagar</a></td></tr
>",
$row["id_estudiante"],$row["nombre_est"],$row["apellido_est
"],$row["id_especial"],$row["nombre_especial"],$row["valor_
especial"]);
}
mysql_free_result($result);
?>
</table>
     </div>
     <div style="clear: both;"> </div>
     </div>
     <div id="footer">
     <p>Copyright (c) 2008 Sitename.com. All rights
reserved. Design by <a
href="http://www.freecsstemplates.org/">Free CSS
Templates</a>.</p>
```

```
     </div>
</div>
</body>
</html>
```

El script anterior es fácil de escribir y rápido de ejecutar pero difícil de mantener y actualizar. Cabe recalcar que no se ha incluido el código que se encarga del manejo de estilos, banners y menús; el mismo que también se mezcla con el resto de funciones, generando una gran cantidad de código que es difícil de entender y no facilita el trabajo en grupo ya que la única persona que posee la capacidad de entender el código en un desarrollador más no un diseñador.

Mediante la utilización de la arquitectura MVC se separa el código html del php y todas las funciones de las cuales se compone como son: el acceso a datos, la realización de cálculos y la visualización de información. Generando scripts mucho más fáciles de mantener, incluso por personas que no son expertas en programación.

Para realizar el pago de una deuda

Creación del proyecto con el paradigma MVC en Symfony

La vista

```
<table id="tabla" border="1">
  <thead>
    <tr>
      <th>Estudiante</th>
      <th>Id Rubro</th>
      <th>Rubro Especial</th>
```

```
      <th>Valor</th>
      <th>Pagar</th>
    </tr>
  </thead>
  <tbody>
    <?php foreach ($test_especials as $test_especial): ?>
    <tr>
      <td><?php echo $test_especial->getnombreestudiante()
?></td>
     <td><?php echo $test_especial->getCodEspecial()
?></td>
      <td><?php echo $test_especial-
>getnombrerubroespecial() ?></td>
      <td><?php echo $test_especial->getValorTotal()
?></td>
      <td><a href="<?php echo
url_for('estudianterubroespecial/pagar?cod_estudiante='.$te
st_especial-
>getCodEstudiante().'&cod_especial='.$test_especial-
>getCodEspecial().'&fecha='.$test_especial->getFecha())
?>">Pagar</a>
      <br />
    </td>
    </tr>
    <?php endforeach; ?>
  </tbody>
</table>
```

El controlador

```
class estudianterubroActions extends sfActions
{
  public function executeIndex(sfWebRequest $request)
  {
    $this->test_especials =
Doctrine::getTable('TestEspecial')
```

```
      ->createQuery('a')
      ->execute();

     $this->deudores= Doctrine::getTable('TestEspecial')-
>getDeudores();
  }
```

El modelo

```
<?php
class TestEspecialTable extends Doctrine_Table
{
     public function getDeudores()
     {
     $q= $this->createQuery('e')
     ->where('e.estado = ?', 'd');
     return $q->execute();
     }
}
```

Otras implementaciones

Pagar Deuda

El siguiente código no utiliza MVC y su objetivo es realizar el pago de una deuda (cambiar el estado de la deuda a p).

```
<?php
$estudiante=$HTTP_GET_VARS["idest"];
$rubro=$HTTP_GET_VARS["idrubro"];
include("conexion.php");
$link=Conectarse();
$Sql="update    test_especial    set    estado='p'    where
id_especial=$rubro and id_estudiante=$estudiante";
```

```
mysql_query($Sql,$link);
echo "Se actualizaron los datos correctamente";
?>
```

Mediante el uso de MVC el código se agruparía de la siguiente manera

Controlador

```
public function executePagar(sfWebRequest $request)
{
$this->estudiante->pagar();
}
```

Modelo

```
class TestEspecial
{
     public function pagar()
     {
     $this->setEstado('p')
     $this->save();
     }
}
```

Como se puede observar el código implementado en base a MVC es mucho más legible y fácil de mantener, al mismo tiempo que puede ser reutilizado un sin número de veces.

ANEXO III

Implementación con Symfony

Índice

1. Instalación de Symfony

El primer paso para instalar symfony es instalar y levantar un servidor Apache, Php y Mysql a través del XAMPP en su Versión 2.5. Posteriormente instalar el PEAR para esto realizar los siguientes pasos:

- Abrir la consola de comandos Menú inicio -> Ejecutar -> cmd. Ir a la carpeta php la misma que se encuentra en el directorio c:\xampp\php y posteriormente ejecutar el instalador/configurador de Pear con el comando go-pear.bat

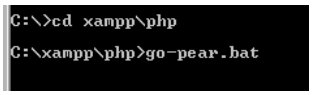


Figura 1. Ejecutar instalador del Pear

- Preguntará si es una copia local. Presionar enter

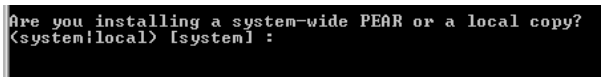


Figura 2. Verificar Copia Local

- Se despliega la lista de directorios PEAR. Presionar enter

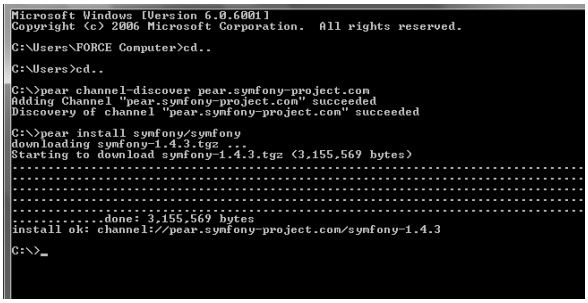


Figura 3. Lista de directorios Pear

- Se crea el archivo PEAR_ENV.reg en c:\xampp\php el cual se deberá ejecutar y confirmar la operación como administrador.

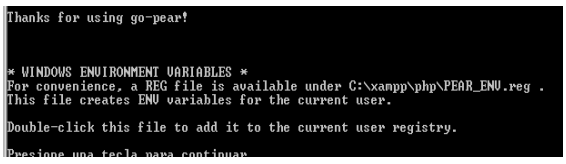


Figura 4. Creación archivo PEAR_ENV.reg

- Al mismo tiempo se creará el archivo pear.bat en la carpeta c:\xampp\php el mimo que se deberá copiar y pegar en la carpeta c:\windows\system32.

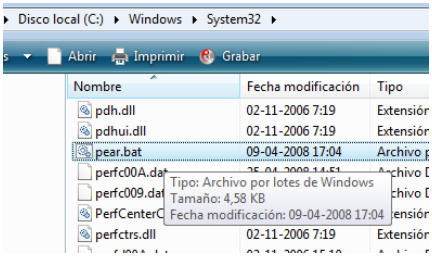


Figura 5. Archivo Pear.bat

- Para comprobar que se ha instalado correctamente, ejecutar el comando pear config-show desde cualquier ubicación, donde mostrará la configuración actual. Si no aparecen errores, se abra instalado correctamente el PEAR.

Una vez instalado PEAR se procede a ***instalar symfony*** para esto se srealizan los siguientes pasos:

- En la consola de comandos agregar el 'channel' de Symfony en Pear ejecutando *pear channel-discover pear.symfony-project.com*

Figura 6. Agregar Channel Symfony

- . Instalar Symfony con el comando *pear install symfony/symfony.*

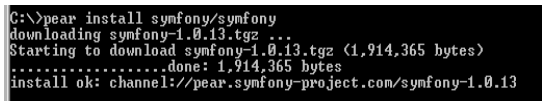


Figura 7. Instalar Symfony

- Si todo va bien se creará el archivo symfony.bat en c:\xampp\php, el cual se deberá copiar y pegar en c:\windows\system32.

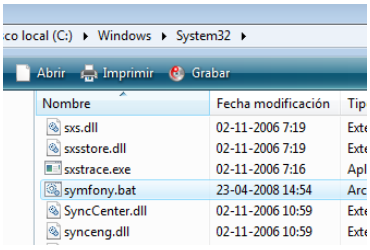


Figura 8. Archivo Symfony.bat

- Como es esta la ruta desde donde se accede, se deberá modificar el symfony.bat que está en la carpeta system32 para que funcione. Las modificaciones son las siguientes:

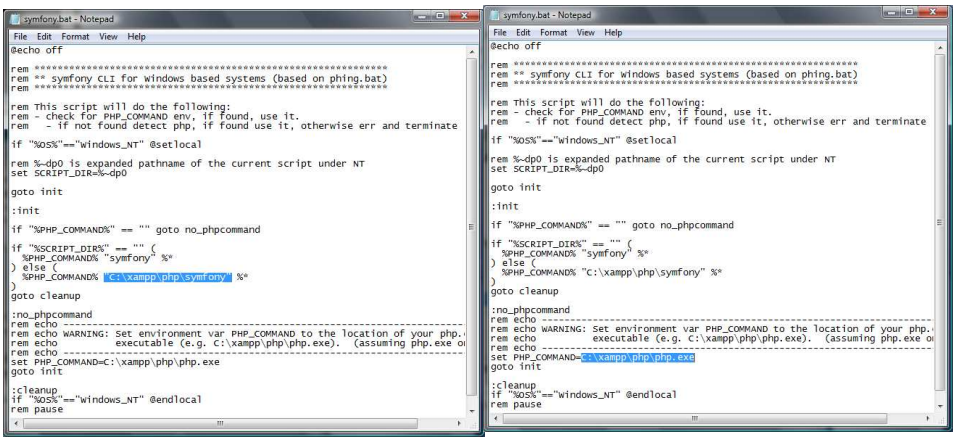


Figura 9. Modificar Symfony.bat

- Si todo está bien, al ejecutar el comando symfony desde cualquier ubicación se deberá mostrar la lista de comandos, como muestra la siguiente imagen.

Figura 10. Lista de comandos symfony

De esta manera se comprueba que symfony ha sido correctamente instalado. El siguiente punto será crear un nuevo proyecto.

2. Trabajo con Symfony

2.1.1. Creación del proyecto en Symfony

Una vez instalado se procede a crear mediante el script que nos proporciona Symfony, un proyecto vacío, para ello, se crea una carpeta *rubros* en, c:*/xampp/httdocs/* y se ejecuta *symfony generate:project rubros.* Esto crea todo el árbol de directorios y archivos necesarios para poner en marcha la aplicación web con este framework. Todo proyecto creado por Symfony consta de una serie de carpetas y archivos que contienen el código base del framework.

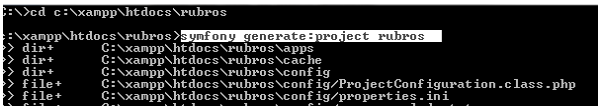


Figura 11. Crear Proyecto Symfony

Los directorios que se generan son los siguientes:

Directorio	Descripción
apps/	Hospeda todas las aplicaciones del proyecto
cache/	Los archivos en caché
config/	Los archivos de configuración del proyecto
lib/	Las bibliotecas y clases del proyecto
log/	Los archivos de registro
plugins/	Los plugins instalados
test/	Los archivos de pruebas unitarias y funcionales
web/	El directorio raíz web (véase más adelante)

Tabla. Directorios de la aplicación Symfony

El proyecto se compone de varias aplicaciones y las aplicaciones de módulos, inicialmente se creará una aplicación frontend por medio de la siguiente línea de comandos.

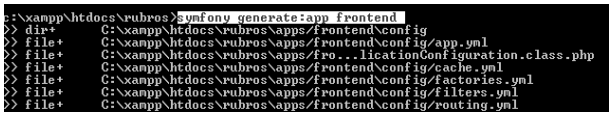


Figura 12. Crear Aplicación Frontend

El comando anterior crea una estructura de directorios para la aplicación bajo el directorio apps/frontend además de proporcionar automáticamente ciertas medidas de seguridad. Los directorios creados se describen en la siguiente tabla

Bajo el directorio web se crean 2 archivos index.php y frontend_dev.php, ambos archivos apuntan a la misma aplicación pero trabajan en diferentes entornos, ya que symfony maneja cuatro entornos que son: *desarrollo, pruebas, staging y producción*.

2.2. Configuración del Servidor Web

Una buena práctica cuando se desarrollan aplicaciones web consiste en crear un servidor virtual por cada proyecto. De esta forma, se puede acceder al proyecto mediante una URL sencilla como http://rubros, en vez de

http://localhost/rubros, http://localhost/sf_sandbox/rubros o cualquier otra URL más larga.

Los pasos realizados para crear un servidor virtual para nuestro proyecto de cobro de rubros son los siguientes:

- En el archivo httpd-vhosts.conf que se encuentra en el directorio c:\xampp\apache\conf\extra descomentar y modificar las siguientes líneas de código:

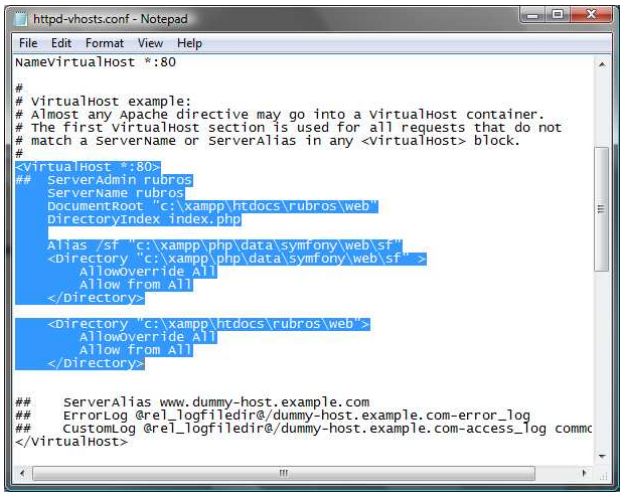


Figura 13. Crear Servidor Virtual

- Posteriormente en el archivo hosts que se encuentra dentro del directorio c:\windows\system32\drivers\etc modificar las siguientes líneas.

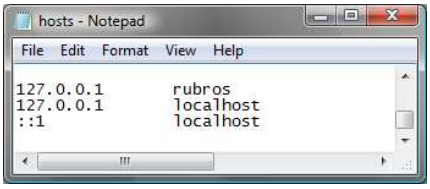


Figura 14. Configurar Hosts

- Guardar los cambios realizados y reiniciar el servidor web. Para probar si funciona, abrimos el navegador y escribimos http://rubros y aparecerá la siguiente pantalla que confirma la correcta configuración.

Figura 15. Pantalla de bienvenida Symfony

2.3. Trabajo con el modelo de datos

El siguiente paso en el desarrollo de nuestra aplicación es el manejo de la base de datos la misma que se realizó en MySQL , para la conexión con nuestra aplicación se utilizará el ORM Doctrine. Symfony es capaz de generar una aplicación totalmente funcional sin la necesidad de escribir mucho código.

Debido a que Symfony es un framework Orientado a Objetos ofrece funcionalidades para mapear las bases de datos relacionales a un modelo de objetos, estas son Propel y Doctrine.

2.3.1. Crear Esquema de Base de Datos

El esquema de la base de datos describe todas las tablas, datos, relaciones, claves, etc. En symfony existen 2 maneras de crear los esquemas una es manualmente y otra es a través de una base de datos existen por medio de comandos propios de doctrine de la siguiente manera.

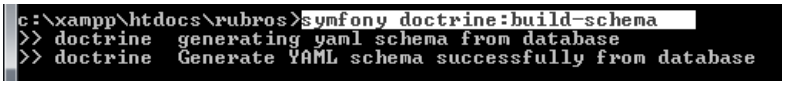


Figura 16. Crear Esquema de datos

Se genera el archivo schema.yml siempre y cuando se tenga correctamente configurado el archivo databases.yml el mismo que almacena la información del gestor de base de datos, el nombre de la base, usuario y la clave. La base de datos que se utilizará para nuestra aplicación es bdrubros.

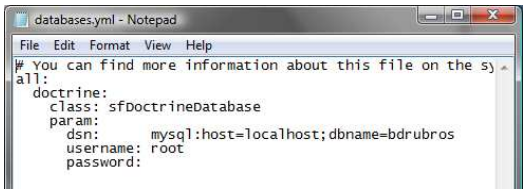


Figura 17. Archivo databases.yml

2.3.2. Crear Modelo de Datos

Procedemos a construir el modelo que es el primer componente del patrón de diseño MVC a partir de los archivos del esquema por medio de los siguientes comandos.

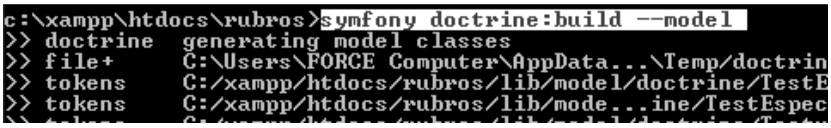


Figura 18. Crear Modelo

Ahora que los modelos existen se puede generar e insertar el SQL. La tarea *doctrine:build --sql* genera comandos SQL en el directorio *data/sql/*, optimizado para el motor de base de datos que se ha configurado.

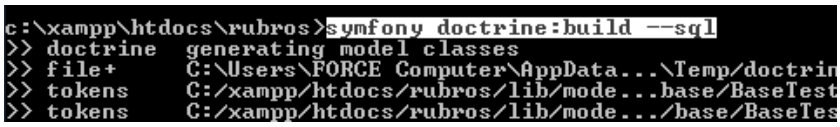


Figura 19. Crear Sql

La tarea doctrine:build --all es un acceso directo para las tareas que han ejecutado anteriormente y al mismo tiempo para generar formularios y validadores para el modelo de clases, los mismos que será explicados más adelante.

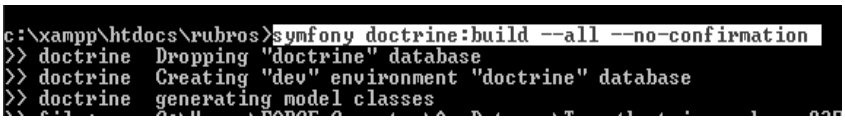


Figura 20. Crear todo

En base a todas las clases, formulario, validadores generados automáticamente por el framework se empieza a personalizar nuestra aplicación de acuerdo a las necesidades que se tengan.

2.3.3. Personalizar Modelo de Datos

Todas las clases pertenecientes al modelo y generadas automáticamente por el framework se almacenan bajo el directorio lib\model\doctrine, las mismas que van a ser utilizadas para añadir nuevos eventos a nuestros objetos. Bajo el subdirectorio \base se almacenan clases que contienen código generado por el framework y que sirven como pilar para la personalización de objetos. Implementa entre otras cosas los métodos get y set de cada objeto.

En nuestra aplicación vamos a necesitar definir ciertos comportamientos de los objetos como:

TestEspecial debe permitir almacenar un nuevo registro. Para esto se a definido el siguiente método:

```
class TestEspecial extends BaseTestEspecial
{public                                    function
guardar($idestudiante,$idespecial,$estado)
   {
    $this->setIdEstudiante($idestudiante);
    $this->setIdEspecial($idespecial);
    $this->setEstado($estado);
    $this->save();
   }
}
```

Testudiante debe devolver el nombre completo del estudiante, es decir concatenar nombres con apellidos

```
class Testudiante extends BaseTestudiante
{
public function nombrecompleto()
   {
return       $this->getNombreEst()."       ".$this-
>getApellidoEst();
   }
}
```

Cuando se desea implementar consultas a la base de datos que involucre una o varias tablas, es ahí que se utilizan los otros archivos

existentes en el directorio del modelo cuyo nombre tiene la forma *NombreTabla***Table.class.php**. En este caso utilizaremos el archivo TestudianteTable para escribir un método que nos devuelva todos los estudiantes de un nivel específico esto nos servirá a futuro para asignar a los estudiantes de un solo nivel un rubro y para obtener los estudiantes activos:

```
class TestudianteTable extends Doctrine_Table
{public function getEstudiantesporNivel($nivel)
   {
   $q= $this->createQuery('e')
   ->where('e.nivel = ?', $nivel);
   return $q->execute();
   }
   public function getactivos()
   {
   $q=  $this->createQuery('e')->where('e.estado_est
   = ?', 'A');
   return $q->execute();
   }
   }
```

De la misma manera se implementara un método que realice una consulta para conocer los estudiantes que deben un rubro es decir que su estado es ‘d’

```
class TestEspecialTable extends Doctrine_Table
{    public function getDeudores()
{
$q= $this->createQuery('e')
->where('e.estado = ?', 'd');
return $q->execute();
}
}
```

2.4. Controlador y Vista

2.4.1. Crear Módulos

Como se sabe los proyectos en symfony se componen de aplicaciones y estos a su vez de módulos los cuales representan una característica de la aplicación. Es por eso que nuestro siguiente paso es crear un módulo que se encargará de listar, ingresar nuevos datos, modificar y eliminar información.

Symfony es capaz de generar automáticamente un módulo para un determinado modelo que proporciona las características básicas de manipulación:

```
c:\xampp\htdocs\rubros>symfony doctrine:generate-module --with-show --non-verbos
e-templates frontend estudiante Testudiante
>> dir+      C:\xampp\htdocs\rubros\apps\frontend\modules/estudiante\actions
>> file+     C:\xampp\htdocs\rubros\apps\fro...iante\actions/actions.class.php
>> dir+      C:\xampp\htdocs\rubros\apps\fro...nd\modules/estudiante\templates
>> file+     C:\xampp\htdocs\rubros\apps\fro...iante\templates/editSuccess.php
>> file+     C:\xampp\htdocs\rubros\apps\fro...ante\templates/indexSuccess.php
```

Figura 21. Crear Módulo

De la misma manera se generarán los módulos restantes (rubroespecial y estudianterubro) necesarios para el desarrollo de la aplicación

Para verificar que los formularios fueron creados de manera correcta abrir el navegador y poner la siguiente url: http://localhost/frontend_dev.php/estudiante se listará el registro de estudiantes almacenados en la base de datos. Para ingresar un nuevo registro dar click en New y se desplegará el formulario en el que podemos ingresar un nuevo estudiante, de la misma manera posee funciones para modificar y eliminar.

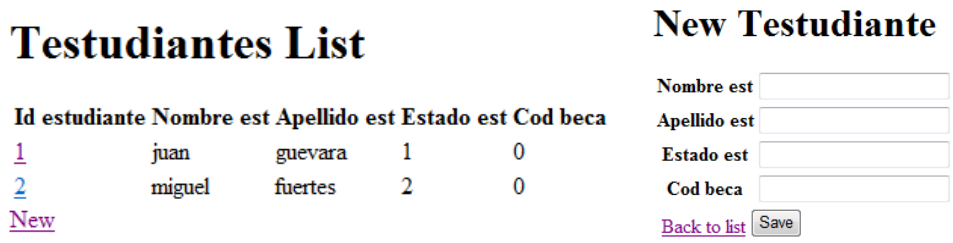


Figura 22. Pantallas Listar y Crear Estudiantes

Otras de las funciones que automáticamente provee la herramienta al momento de crear los formularios es la de validación de los datos ingresados, un ejemplo es ingresar una letra en un campo que solicita un número. En la siguiente imagen se ve como se produce un mensaje de error al momento de querer ingresar un dato incorrecto.

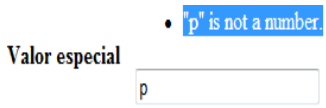


Figura 23. Validadores

2.4.2. Crear Vista

La Vista es con lo que interactúa el usuario final y esta debe servir únicamente para enviar y mostrar información, gracias a Symfony se pueden crear automáticamente varias vistas para las tareas más comunes dentro de un aplicación: listar, ingresar, modificar y eliminar. Estas se almacenan dentro del directorio templates del módulo al que pertenecen.

Un ejemplo del código que se implementa dentro de una vista enfocada a generar la listar los estudiantes es:

```
<table>
  <thead>
    <tr>
      <th>Id estudiante</th>  <th>Nombre</th>
      <th>Apellido</th> <th>Nivel</th>
      <th>Estado</th> <th>Codigo beca</th>
    </tr>
  </thead>
  <tbody>
    <?php foreach ($testudiantes as $testudiante):
?>
    <tr>
      <td><a href="<?php echo
url_for('estudiante/show?id_estudiante='.$testudiant
e->getIdEstudiante()) ?>"><?php echo $testudiante-
>getIdEstudiante() ?></a></td>
      <td><?php echo $testudiante->getNombreEst()
?></td>
      <td><?php echo $testudiante->getApellidoEst()
?></td>
```

```
      <td><?php     echo     $testudiante->getNivel()
?></td>
      <td><?php   echo   $testudiante->getEstadoEst()
?></td>
      <td><?php    echo    $testudiante->getCodBeca()
?></td>
    </tr>
    <?php endforeach; ?>
  </tbody>
</table>
```

Como se puede observar el código implementado únicamente tiene la función de mostrar información, reduciendo considerablemente la responsabilidades y por ende las líneas de código. $testudiante es el objeto que hace llamadas a funciones definidas el Modelo de datos, por ejemplo si se quiere llamar a la función nombrecompleto() definida anteriormente se debería escribir algo así $testudiante-> nombrecompleto(). Donde $testudiante es un objeto de tipo Testudiante.

Los diferentes formularios que manejan la aplicación y la vista, tanto para el ingreso, modificación y eliminación no se definen como tradicionalmente se lo realizaba:

```
<form method=”post” action=”pagina.php”>
<input type=”text” id=”nombre”>
<input type=”text” id=”cedula”>
<input type=”submit” Value=”guardar”>
</form>
```

Con la ayuda de symfony los formularios son definidos mediante un framework completo que almacena la información en clases php los mismos que se explicaran más adelante. Reduciendo de esta manera el código implementado en la vista y promoviendo la reusabilidad.

Las interfaces anteriormente creadas no cuentan con ningún estilo, ni encabezados ni pies de páginas. Es aquí donde se procede a trabajar con los estilos y las plantillas. La plantilla que se va a manejar dentro de la aplicación se escribe en el archivo *layout.php*, que se encuentra

en el directorio *apps/frontend/templates* y en la parte del contenido, es decir la única sección dinámica dentro de la página escribir *<?php echo $sf_content ?>* es aquí donde se mostrara el código html generado en otra acción.

Para configurar los estilos a utilizar en una aplicación simplemente se escribe en el archivo view.yml que se encuentra bajo el directorio *apps/frontend/config* las siguientes líneas y este será referenciado por un helper en la pagina html.

stylesheets: [main.css]

Si muchos archivos se definen, Symfony los incluye en el mismo orden que la definición:

stylesheets: [main.css, style.css]

Para referenciar a los estilos en la página html en este caso *layout.php* se escribe el siguiente helper.

<?php include_stylesheets() ?>

Sin utilizar este helper normalmente en una página html se debería referenciar de la siguiente manera:

```
<link rel="stylesheet" type="text/css" media="screen"
href="/css/main.css" />
<link rel="stylesheet" type="text/css" media="screen"
href="/css/style.css" />
```

Se puede diferenciar claramente la disminución de líneas de comandos y por ende del esfuerzo por medio de la utilización del helper.

Una vez creado el layout y aplicado los estilos al mismo, las páginas pertenecientes al módulo se visualizarán de la siguiente manera:

Figura 24. Interfaces con Estilos

2.4.3. Crear Controlador

Las acciones son el corazón de la aplicación, puesto que contienen toda la lógica de la aplicación. Las acciones utilizan el modelo y definen variables para la vista. Cuando se realiza una petición web en una aplicación Symfony, la URL define una acción y los parámetros de la petición.

Existe un Controlador para cada módulo y este se guarda bajo el directorio actions. Consiste en un archivo en el que se encuentra definidas varias funciones, una por cada vista, por ejemplo para la vista NewSuccess.php existe un evento executeNew();

El código del Contralor necesario para realizar la tarea de listar estudiantes es el siguiente:

```
class estudianteActions extends sfActions
{
public function executeIndex(sfWebRequest $request)
{
  $this->testudiantes                                    =
Doctrine::getTable('Testudiante')
    ->createQuery('a')
    ->execute();
}
}
```

Por medio de este código se obtienen todos los datos de la tabla estudiante y se almacena en la variable testudiantes la misma que es utilizada en la vista para recorrer los registro devueltos por el modelo.

```
    <?php foreach ($testudiantes as $testudiante):
?>
```

Otra de las implementaciones en la que se puede ver el funcionamiento del patrón MVC es cuando se requiere obtener únicamente los datos de los estudiantes activos, en este caso lo único que se tiene que modificar es el controlador, para invocar al método

que *getactivos()*, anteriormente implemento en el Modelo dentro del archivo *TestudianteTable.class.php*

```
public function executeIndex(sfWebRequest $request)
{
       $this->testudiantes=
   Doctrine::getTable('Testudiante') ->getactivos();
}
```

La función implementada dentro del controlador que se encarga de asignar un rubro a los estudiantes de un nivel es la siguiente:

```
public function executeAsignaranivel(sfForm $form,
$nivel)
     {
   $estudiantes= Doctrine::getTable('Testudiante')->
   getEstudiantesporNivel($nivel);
   foreach ($estudiantes as $estudiante)
    {
       $this->estrubro= new TestEspecial();
        //lamada a la funcion guardar con el codigo
   del estudiante,el codigo de rubro y el estado
       $this->estrubro->guardar($estudiante->
       getIdEstudiante(),$form->
       getValue('id_especial'),'d');
    }
```

Descripción

- Llama a la función getEstudiantesporNivel con el parámetro $nivel que es enviado por el usuario y lo almacena en un variable.
- Recorre los datos recibidos y almacenados en $estudiante
- Crea un nuevo objeto TestEspecial para cada estudiante seleccionado y asignar el rubro y el estado ‘d’ (debe).

2.5. Formularios

El framework de formularios de Symfony proporciona al programador todas las herramientas necesarias para mostrar y validar fácilmente datos en un formulario de forma similar a los objetos.
Los formularios se almacenan bajo el directorio \lib\form y son generados automáticamente al momento de escribir el comando *symfony doctrine:build –all –no-confirmation*
De la misma manera que en el modelo existe un archivo por cada formulario (TestudianteForm) que es editable por el desarrollador y un base (BaseTestudianteForm) del cual heredan todas las características generadas por el framework.

Cada campo de un formulario se compone de un widget y un validator. Un **widget** representa un campo de un formulario, por ejemplo:

- *sfWidgetFormInput*: Este widget representa un campo de tipo input.
- *sfWidgetFormTextarea*: Este widget representa un campo de tipo textarea.

La asignación del widget se realiza con el método setWidgets()

```
$this-> setWidgets(array(

  'nombre_est' => new sfWidgetFormInput(),

  'apellido_est' => sfWidgetFormInput(),

  'estado_est' => sfWidgetFormInput(),

  'cod_beca' => sfWidgetFormInput(),

  ));
```

La validación de cada campo la hacen objetos descendientes de la clase sfValidatorBase. Para validar el formulario de estudiante tenemos que definir objetos validadores para cada uno de los campos. Esto se realiza de la siguiente manera:

```
$this-> setValidators(array(
```

```
  'nombre_est' => new sfValidatorString(),

  'apellido_est' => sfValidatorString(),

  'estado_est' => sfValidatorString(),

  'cod_beca' => sfValidatorSNumber(),

  ));
```

De esta manera se crean los formularios y su funcionamiento es similar al de un objeto. Para referenciarlo e incluirlo en una vista, se lo debe invocar desde el controlador de la siguiente manera:

```
public function executeNew(sfWebRequest $request)

  {

 $this->form = new TestEspecialForm();

  }
```

Su apariencia sera la siguiente:

Figura 25. Formulario diseñado

Como se puede observa aún falta refinar varias cosas como las etiquetas, mensajes de error, valores predeterminados, entre otras cosas. El framework de formularios que maneja Symfony permite realizar un refinamiento completo y de manera ágil y sencilla.

Mediante el uso de este tipo de formularios se promueve la reusabilidad y permite al desarrollar ahorrar una gran cantidad de código, necesarias

tanto para el diseño como para la validación de los distintos formularios que intervienen en una aplicación.

ANEXO IV

Encuesta

ESCUELA SUPERIOR POLITECNICA DE CHIMBORAZO
ESCUELA DE INGENIERIA EN SISTEMAS

Objetivo: Obtener información acerca de la utilización de los patrones de diseño en las diferentes empresas desarrolladoras de software y la disposición que poseen los programadores para adaptarse al patrón de diseño MVC para el desarrollo y crecimiento del negocio.
Instrucciones: Seleccione una sola respuesta y conteste con la veracidad que amerita el caso.

Preguntas: De acuerdo a su experiencia en páginas o portales web por favor responda las siguientes preguntas calificándolas entre 1-5 tomando como menor puntaje 1 y mayor puntaje 5, o a su vez, 0 (cero) en caso de abstenerse:

1. ¿Cree usted que el uso de MVC implementado en un framework reduce los costos de desarrollo de un provecto?

   1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐

2. ¿Cree usted que el uso de MVC implementado en un framework NO reduce los costos de desarrollo de un proyecto?

   1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐

3. ¿Considera usted que sin el uso de MVC implementado en un framework se reducen los costos de desarrollo de un proyecto?

   1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐

4. ¿Considera usted que sin el uso de MVC implementado en un framework NO se reducen los costos de desarrollo de un proyecto?

   1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐

5. ¿Cree que el empleo de MVC implementado en un framework menora el tiempo de desarrollo de software personalizado?

1 2 3 4 5
☐ ☐ ☐ ☐ ☐

6. ¿Cree que el empleo de MVC implementado en un framework NO menora el tiempo de desarrollo de software personalizado?

1 2 3 4 5
☐ ☐ ☐ ☐ ☐

7. ¿Cree que sin emplear MVC implementado en un framework menora el tiempo de desarrollo de software personalizado?

1 2 3 4 5
☐ ☐ ☐ ☐ ☐

8. ¿Cree que sin emplear MVC implementado en un framework NO menora el tiempo de desarrollo de software personalizado?

1 2 3 4 5
☐ ☐ ☐ ☐ ☐

¡Gracias por su tiempo y colaboración!

Índice General

CAPÍTULO III

CAPÍTULO IV

Índice de Ilustraciones

Índice de Tablas