Universidad de las Ciencias Informáticas

Facultad 2 "TELECOMUNICACIONES Y SEGURIDAD INFORMÁTICA"

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERO EN CIENCIAS INFORMÁTICAS

Título: Sistema de Análisis Estático de Vulnerabilidades en Python

AUTORES

Teudis Naranjo Ortiz

William Sánchez Espinosa

TUTORES

Ing. Enrique Félix Agudo Veliz

Ing. Ania Bermúdez Peña

Ciudad de La Habana, junio del 2010.

Declaración de Autoría

Declaramos que _____y_____ somos los únicos autores de este trabajo y autorizamos a la Universidad de las Ciencias Informáticas (UCI) y a la Facultad (2) para que hagan el uso que estimen pertinente con este trabajo.

Para que así conste firmamos la presente a los ____ días del mes de julio del 2009.

_____ _____

Firma del Autor Firma del Autor

<Autor 1> <Autor 2>

_____

Firma del Tutor

<Tutor>

Frase

"…lo fundamental es hacer algo nuevo cada día y luego perfeccionar lo que se ha hecho el día anterior…"

Ernesto Guevara de la Serna

Agradecimientos

Queremos agradecerle a nuestro Comandante en Jefe y a la obra de la Revolución Cubana por darnos la posibilidad de graduarnos como ingenieros en Ciencias Informáticas. A nuestra familia por darnos el apoyo cuando lo necesitamos. A nuestros profesores, que proporcionaron gran parte de los conocimientos que tenemos. A nuestros tutores, por haber asumido esta tarea. A aquellos que nos tendieron su mano cuando la necesitamos, a todos, muchas gracias…

A mi familia por apoyar siempre y confiar en mí.

A mis padres por ser mi fuente de inspiración y guiarme para que fuera cada día mejor persona.

A mis amigos de la universidad que siempre hemos estado juntos en las buenas y en las malas, entre ellos están: el "Rojo", la "Bala", el "Wilo", el Rene, Leonel, Eduardo Juarez, Mario, Ángel, Ivan, por solo mencionar los más importantes.

A mi compañero de Tesis por formar un equipo único.

A mis amigos de Bayamo: Tatiana, Chongon, Alien, Norlys, Lázaro que siempre he podido contar con ellos.

Teudis

A mi madre, por su cariño, amor y dedicación en todo momento, lo que soy y seré se lo debo a ella.

A mis abuelos paternos, por todo el afecto entregado a lo largo de todos los años que disfrute con ellos.

A mi familia en sentido general, mi padre, mis hermanos y especialmente mis tías y mis primas Yisel y Yanet, por todo su apoyo, preocupación y afecto entregado a lo largo de estos cinco años.

A mis amigos de siempre Renier y Salvador, porque a pesar de estar en facultades diferentes, siempre estaban ahí cuando los necesitaba.

A mi compañero de tesis por ser el corazón de este trabajo y mirar hacia adelante en todo momento con optimismo.

A mi segunda familia, la que he construido aquí en la UCI y muy especialmente a los que me han acompañado desde el primer día que pise esta tierra, a ustedes, mis hermanos que han aguantado todas mis peleaderas y con los que he pasado hermosos momentos y las chicas que desde el 2106 y hasta este momento me han acompañado.

A mis vecinos Ana Gloria, Bertha y Delis por ser parte esta proeza, prestándome su ayuda desinteresada en todo momento.

A todos ustedes, miles de gracias, aquí esta el fruto de su amor.

William

Dedicatoria

A mi familia, en especial a mis padres.

A mi hermana, que ojalá estuviera viva para ver este momento.

Teudis

Dedico este título de diploma a la persona que más quiero en este mundo y la que ha sido el motor impulsor de mi vida; la persona, que más ha dado de si por entregarme hasta lo que no ha tenido; la persona que me ha bridado tanto, que sin ella nuca habría podido ser lo que soy en estos momentos. Por su enorme sacrificio, entrega, apoyo, preocupación, dulzura y amor, mereces este esto y mucho más. Es para ti "mima" el fruto de estos cinco largos años.

Dedico este trabajo además, a otras dos personas que he querido mucho en mi vida: mis abuelos paternos Rafael y Flor Edelma, por entregarme su amor desde los primeros años de mi vida, convirtiéndose en mis segundos padres. El camino para llegar aquí ha sido largo y es muy duro para mí ver que no hayan podido llegar hasta el final. Pero no se preocupen, que un lugar de mi corazón es solo para ustedes. Por todo esto y mucho más, reciban este título allá donde quiera que estén.

William

Resumen

En la actualidad, el desarrollo del *software* mantiene un gran paso, trayendo consigo la utilización de varios lenguajes de programación, dentro de los cuales se encuentra *Python*. Dicho lenguaje presenta varias características que hacen del mismo una buena opción para realizar aplicaciones; pero, no quiere decir que sea inmune a las vulnerabilidades referentes a la seguridad de *software* que pueden presentar debido a la no utilización de técnicas de programación segura. Una manera de de solucionar estas vulnerabilidades es utilizando herramientas de análisis estático.

En la Universidad de las Ciencias Informáticas existen proyectos productivos que trabajan en el lenguaje de programación *Python*, pero estos no cuentan con una aplicación que realice el análisis de código estático. Por tanto, estos proyectos son entregados sin la calidad requerida, con problemas de seguridad y sin utilizar buenas prácticas de programación. Si tenemos en cuenta que algunos de estos proyectos son de exportación, estaríamos entregando al cliente un producto con vulnerabilidades, lo cual causaría una posible pérdida de mercado, un gasto de recursos en vano y la perdida de confianza del cliente.

En este trabajo se presenta una solución para mejorar la calidad con que se entregan los proyectos realizados en el lenguaje de programación *Python* en la Universidad. De esta forma se crea un sistema que utiliza herramientas de análisis estático para lograr darle solución al problema que se plantea, utilizando un grupo de funcionalidades.

Palabras claves: análisis, estático, herramientas, vulnerabilidades.

Tabla de contenido

INTRODUCCIÓN

Actualmente el desarrollo del *software* ha tenido gran auge en el mundo, trayendo consigo el uso de diversos lenguajes de programación, bajo la filosofía que propone la Programación Orientada a Objetos (POO) como paradigma. Uno de los lenguajes más difundidos en el mundo del *software* libre es el *Python*, (creado por *Guido van Rossum*[1]) el cual puede ser interpretado o de script.

La ventaja de los lenguajes compilados es que su ejecución es más rápida. Sin embargo, los lenguajes interpretados son más flexibles y más portables. *Python* tiene, no obstante, muchas de las características de los lenguajes compilados, por lo que se podría decir que es semi-interpretado (1), pero una característica distintiva es el tipado dinámico. Por tanto, se puede decir que "*Python* no es adecuado para la programación de bajo nivel o para aplicaciones en las que el rendimiento sea crítico. Algunos casos de éxito en el uso de *Python* son *Google*, *Yahoo*, la *NASA*, Industrias *Light & Magic*, y todas las distribuciones *Linux*, en las que *Python* cada vez representa un por ciento mayor". (2)

Además, *Python* es un lenguaje de programación sencillo pero potente y versátil, cuya popularidad crece día a día, pero, no está libre de vulnerabilidades, pues, las diferentes aplicaciones que se desarrollan en este lenguaje pueden ser atacadas y vulnerables, debido al poco uso de técnicas de programación segura como son: control del desbordamiento del *buffer*, establecimiento de control de acceso, privilegios mínimos y criptografía segura, provocando males como denegaciones de servicio, inestabilidad del sistema operativo y resultados inesperados en el sistema *Python*, lo cual no garantiza los principios básicos de seguridad, los cuales son: integridad, confidencialidad y disponibilidad. Una alternativa para solucionar parte de los problemas que se pueden presentar en el desarrollo de dichas aplicaciones es utilizando herramientas de análisis de código estático.

[1] Guido van Rossum es un científico de la computación, más conocido por ser el autor del lenguaje de programación *Python*. Nació y creció en los Países Bajos. En el ambiente de los desarrolladores del lenguaje *Python* también se le conoce por el título **BDFL** (*Benevolent Dictator for Life*). En el año 2001 recibió el *Free Software Award* como reconocimiento por su trabajo. En diciembre de 2005 fue contratado como desarrollador por la empresa estadounidense *Google,* donde actualmente trabaja.

El análisis estático se refiere a las herramientas que analizan el código fuente, "estático", lo cual significa que el código es analizado en un servidor antes de la compilación. Dichas herramientas son utilizadas principalmente con fines relacionados a la seguridad informática, pero una de las posibles soluciones para evitar gran parte del problema de la calidad es utilizar las mismas. La última generación de estas herramientas puede hacer más que el análisis sintáctico, pues tratan de predecir las rutas de código en tiempo de ejecución y buscar los errores. El análisis de código estático tiene como ventaja que se puede analizar todas las posibles rutas de código, algo que es muy difícil de ver con los métodos de prueba tradicionales. Esta última generación de herramientas de análisis estático se puede ajustar además a su base de código específico y puede filtrar informes ajenos para ayudarle al programador a concentrarse en los errores.

Los proyectos productivos que se realizan en el lenguaje de programación *Python* no cuentan con una aplicación que realice el análisis de código estático. El equipo de desarrollo que realiza dichos proyectos no tienen en cuenta las vulnerabilidades que pueden presentan los mismos, sin saber que estas vulnerabilidades son un punto crítico en lo que a debilidades se refiere. Por tanto, estos proyectos se entregan sin la calidad requerida, con problemas de seguridad, con la existencia de vulnerabilidades y sin utilizar el empleo de buenas prácticas de programación. Si tenemos en cuenta que algunos de estos proyectos son de exportación, se estarían entregando al cliente un producto con grandes problemas referentes a la seguridad informática, lo cual causaría una posible pérdida de mercado, deterioro de la confianza del cliente y un gasto de recursos en vano. De esta manera, la **Situación Problemática** para el área enmarcada en la Universidad de las Ciencias Informáticas (UCI), contempla lo planteado anteriormente.

Toda esta situación lleva a definir que el **Problema Científico** de la investigación corresponde a: ¿Cómo validar la seguridad del código en *Python* en la Universidad de las Ciencias Informáticas mediante un análisis estático?

Partiendo del problema científico planteado, **El Objeto de Estudio** se enfocará en el análisis estático de código.

El **Campo de Acción** estará centrado en el análisis estático de código para *Python* en la Universidad de las Ciencias Informáticas.

En aras de resolver el problema planteado, se define como **Objetivo General** desarrollar una aplicación *web* utilizando un analizador estático para la revisión de los proyectos desarrollados en *Python*.

Para complementar lo formulado en el objetivo general se trazaron los siguientes **Objetivos Específicos:**

- Definir una arquitectura que corresponda a las necesidades del sistema a implementar.
- Integrar componentes que permitan dar cumplimiento a los requisitos planteados.
- Identificar una herramienta capaz de analizar código en *Python* en cuanto a programación segura.

Las **Tareas de la Investigación** que se han definido para darle solución a los objetivos propuestos son:

- Realización de un estudio del arte de las diferentes herramientas de análisis estático para el lenguaje de programación *Python* a nivel mundial.
- Análisis y uso de las diferentes tecnologías a utilizar para la realización de la aplicación.
- Modelamiento del Negocio.
    1. Realización de Casos de Uso (CU) y el Diagrama de CU.
    2. Confección del Diagrama de Actividades
    3. Confección del Modelo de Objeto.

- Levantamiento de Requisitos
    1. Definición de Requerimientos Funcionales
    2. Definición de Requerimientos No Funcionales

- Modelamiento del Sistema
    1. Definición de Actores y CU
    2. Realización del Diagrama de CU

- Realización del análisis y diseño del sistema a desarrollar.

- Análisis
  1. Definición Modelo de Análisis y del Modelo de Clases de Análisis.
  2. Realización del Diagrama de Clases de Análisis.

- Diseño
  1. Confección del Diagrama de Clases de Diseño.
  2. Confección del Diagrama de Interacción.
  3. Definición del diseño y patrones a seguir (Modelo Vista Controlador (MVC)).
  4. Diseño de la Base de Datos (BD).
  5. Diseño del Modelo Lógico y Físico de datos.
  6. Diseño de la Interfaz.

- Implementación de la aplicación *web* para gestionar la revisión de los diferentes proyectos realizados en el lenguaje de programación *Python*.
- Implementación
  1. Confección del Diagrama de Despliegue.
  2. Confección del Diagrama de Componentes.
  3. Desarrollo de funcionalidades que le permitan a los usuarios registrarse y una vez hecho esto puedan subir sus proyectos, para que sean analizados.

- Realización y Estudio de la Factibilidad del proyecto a desarrollar.
- Factibilidad
  1. Aplicar Métodos de Estimación.
  2. Calculo de los Puntos de Casos de Uso y Esfuerzo.
  3. Análisis de Costo.

El contenido de este trabajo está estructurado de la siguiente forma:

- **Capítulo 1. Fundamentación Teórica**.

En este capítulo se hace referencia al análisis estático de vulnerabilidades, las herramientas utilizadas para este fin, así como las tendencias, técnicas, tecnologías, metodologías de software en los que se apoyó el equipo de desarrollo para la construcción de la solución del problema que se presenta.

- **Capítulo 2. Características del Sistema**

Este capítulo refleja la investigación realizada con los procesos que tiene lugar en el negocio como objeto de estudio, la situación problemática existente además del objetivo general a cumplir durante el desarrollo del producto. Se describe la propuesta del sistema, se aborda lo referente al funcionamiento del negocio, sus reglas, descripción, y las mejoras y las mejoras que propone el mismo. Se describe además, la solución propuesta utilizando los requisitos funcionales, no funcionales, los casos de uso, el diagrama de casos de uso del sistema y un prototipo de interfaz de usuario.

- **Capítulo 3. Análisis y Diseño del Sistema**

En este capítulo se realizará el análisis y el diseño del sistema a desarrollar, con el propósito de refinar y estructurar los requisitos obtenidos con anterioridad para facilitar la comprensión, preparación, modificación y mantenimiento de los mismos. Se describen los aspectos referentes al diseño de la solución propuesta, se modelan los diagramas de clases del diseño y se especifican los principios para el diseño gráfico.

- **Capítulo 4. Implementación del Sistema**

El contenido de este capítulo está dedicado a tratar los aspectos relacionados con la construcción de la solución propuesta. Se modelan los diagramas de componentes y despliegue, se aborda la descripción de los estándares de diseño, codificación y tratamiento de errores en la solución del sistema.

- **Capítulo 5. Estudio de Factibilidad**

Este capítulo está dedicado a determinar que tan factible es el sistema a implementar. Para ello se hace uso del método “Puntos por Casos de Uso”, el cual arrojará los datos necesarios para determinar

dicha factibilidad. Además, se expondrán los beneficios tangibles e intangibles que se obtendrán con el producto desarrollado, así como el análisis de los costos.

- **Capítulo 6. Pyntch y Pylint**

Este capítulo creado adicionalmente, tiene como objetivo realizar un estudio de las herramientas de análisis de código estático que fueron seleccionadas para vincularlas al sistema. Para ello se verán temas de vital importancia que ayudaran a lograr una mejor comprensión de la utilización de dichas herramientas.

Capítulo 1
FUNDAMENTACIÓN TEÓRICA

1.1. Introducción

El presente capítulo tiene como objetivo abordar sobre el análisis estático de código para el lenguaje de programación *Python*. Para ello se analizarán algunas de las herramientas existentes y se justificará el por qué de las seleccionadas para ser utilizadas en el sistema. También se profundizará sobre las diferentes tecnologías existentes actualmente, así como la metodología utilizada para darle solución al problema planteado.

1.2. Análisis estático de seguridad

La industria del *software* a pesar de seguir un sin número de metodologías de desarrollo de *software* y cumplir diferentes normas de calidad, debe tener en cuenta las amenazas de seguridad. “La seguridad es un requisito básico para los proveedores de *software*, obligados por las fuerzas del mercado, dada la necesidad de proteger infraestructuras de gran importancia y crear y preservar la confiabilidad en la computación”. (3) ¿Qué consecuencias puede traer las infracciones de seguridad? Entre las consecuencias más notables se pueden mencionar: pérdida de beneficios, deterioro de la confianza del cliente, pérdida o compromiso de la seguridad de los datos, perjuicio de la reputación, por solo mencionar algunas. La clave para cumplir la exigencia actual de una mayor seguridad está en implementar procesos reproducibles que proporcionen de manera confiable una mayor seguridad que se pueda medir. Por tanto, los proveedores de *software* deben adoptar un proceso de desarrollo más estricto que se centre, en mayor medida, en la seguridad. Este proceso debe diseñarse para minimizar el número de vulnerabilidades de seguridad presentes en el diseño, la programación y la documentación, así como para detectarlas y eliminarlas cuanto antes en el ciclo de vida de desarrollo. Como parte de diferentes experiencias en el desarrollo de *software* las razones por lo que fallan las diversas aplicaciones son:

- Cambios en el ambiente de ejecución.

- Desbordamientos y chequeos de sintaxis.
- Convenientes pero peligrosas características del diseño del *software*.
- Invocaciones no controladas.
- *Bypass* a bajo nivel.
- Fallas en la implementación de protocolos.
- Fallas en *software* de base.

Para minimizar el número de fallas debemos tener en cuenta el uso de diferentes estándares y buenas prácticas de programación, para ello no podemos olvidar la programación segura. Pero, ¿qué es la programación segura? “Es la arquitectura, diseño y desarrollo de programas que siguen unos principios básicos con el objetivo de reducir la existencia de problemas de seguridad en su código y garantizar los principios básicos de seguridad, los cuales son integridad, confidencialidad, disponibilidad”. (4)

Entre las técnicas de programación segura se destacan: control del desbordamiento del *buffer*, establecimiento de control de acceso, privilegios mínimos y criptografía segura. Como alternativa para solucionar partes de los problemas que se pueden presentar en el desarrollo de aplicaciones, garantizando buenas prácticas de programación y el uso de estándares de codificación es utilizando herramientas de análisis estático.

Como se ha expresado anteriormente el análisis estático de seguridad se refiere a aquellas herramientas que sin ejecutar el código del *software*, tratan de identificar las vulnerabilidades explotables en las aplicaciones y servicios *web*. Pero, antes de analizar detalladamente estas herramientas es necesario saber cuáles son los ataques posibles a realizar, las causas por las que se producen estos ataques y cómo funciona el análisis estático.

Los principales ataques que pueden ser ejecutados por los intrusos contra las aplicaciones *web* son:

- Inyección *SQL*.
- Inyección de código en cliente Web (*XSS*).
- Ataques a la autenticación y secuestro de sesiones.
- Captura no autorizada de información/configuración.

- División *HTTP*, navegación por el sistema de ficheros.
- Desbordamiento de *buffer*.
- Denegación de servicio.

Existen varias causas que hacen posible que individuos ajenos a la aplicación (*hackers*) y con el objetivo de hacer daño, entren a la misma, como las que se mencionan a continuación:

- Modelo económico del software que no incorpora el riesgo o la seguridad del producto.
- Falta de recursos (tiempo, herramientas, personal…).
- Falta de mercado.
- Falta de conocimientos elementales de seguridad en los equipos de desarrollo de software.
- Principios de diseño de seguridad.
- Desconocimiento del cómo usar el protocolo *HTTP*.
- Mala validación de entradas.
- Proceso de desarrollo *software* que ignoran la seguridad.

Una vez mencionado los diferentes ataques a los que está expuesta la aplicación a implementar y las causas que posibilitan que los intrusos puedan acceder a dicha aplicación, se puede analizar cómo se realiza el análisis estático.

"Las primeras etapas son similares a las que sigue un compilador cualquiera. La etapa de análisis semántico permite pasar a una representación interna del *software* en la que están representadas las nociones básicas para determinar defectos de seguridad: flujo de datos y llamadas, tipos y tamaños de las variables, entradas y recursos alcanzables, y qué entradas están bajo control del usuario. Este modelo incluye el entorno (librerías, funciones de sistema, etc.). La "inteligencia del analizador" está codificada en reglas que un verificador aplica sobre el modelo interno para determinar posibles defectos". (5) La información gráfica a las interioridades del análisis estático está reflejada en el **Anexo 1**.

Como se puedo haber observado, las herramientas de análisis estático tienen el mismo diseño que un compilador más, pero, este analiza el código sin ejecutar el *software*. Ahora, esto tiene ventajas y desventajas que se enumeran a continuación:

Ventajas

- Consistencia.
- Detección precoz. La aplicación no tiene que estar integrada ni necesita ejecutarse.
- Su ejecución es barata. Se necesitan pocos recursos para realizar la misma.

Desventajas

- Falsos positivos.
- Falsos negativos.

1.2.1. Herramientas a utilizar

En el mundo actual existen varios fabricantes de herramientas de análisis estático de seguridad, varios de ellos exitosos por los servicios que prestan a las diferentes empresas y compañías que necesitan sus servicios, muchos de ellos ofreciendo como garantía realizar un análisis gratis que demuestre que tan buena o útil le pudiera ser dicha herramienta. Como ejemplo de estos tenemos a *Klocwork*, *Coverity*, *Ounce Labs*, *Veracode*, *Parasoft*, *Grammatech*, *Polyspace*, y *Gimpel*, pero hay que tener en cuenta los siguientes detalles, ofrecen sus servicios pero de forma privada y dentro de los lenguajes de programación que atienden no se encuentra *Python.*

Por lo anteriormente planteado decidimos hacer un estudio de las herramientas existentes para este lenguaje. Con este estudio se pretende mostrar las potencialidades de cada una, destacando las características principales, para así obtener como resultado final la selección del analizador a utilizar. Las herramientas a estudiar son: *PyFlakes, PyChecker*, *Pylint, Pyntch,* entre otras*.*

PyFlakes

PyFlakes es un programa para analizar programas en *Python* y detectar errores. Realiza un chequeo de errores lógicos, pero no tiene en cuenta el estilo del código. No presenta soporte para módulos, lo que

hace que solo verificaría líneas de código. Es relativamente limitado el número de errores detectados. Sin embargo es un programa muy rápido pero no tiene opciones de configuración.

PyChecker

PyChecker es una herramienta de análisis estático para encontrar errores en el código fuente de *Python*. Actualmente permite la importación de archivos y módulos, característica semántica que no realizan otros analizadores. Comparado con *PyFlakes,* durante el chequeo brinda más información. El desarrollo de esta herramienta ha sido lento durante el periodo del 2006 al 2007, el cual ha sido prácticamente muerto, pero en el 2008 se reanudó de nuevo. Durante el análisis puede recursivamente comprobar importaciones, puede ser algo molesto para la librería estándar, lo cual puede solucionarse deshabilitando algunos tipos de chequeos. También puede ser importado dentro del módulo *space* para así hacer análisis en tiempo de ejecución. Esta herramienta es bastante configurable, la cual permite habilitar y deshabilitar diferentes chequeos, es importante mencionar que los más importantes (chequeos), por defecto están deshabilitados. Se puede decir que su funcionamiento es estable para *Python* 2.2 y 2.3.

RATS (Rough Auding Tool for Security)

RATS, desarrollado por los ingenieros del "*Secure Software*", que actualmente pertenece a "*Fortify Software*", es una herramienta para escanear el código fuente de las aplicaciones escritas en *C*, *C++*, *Perl*, *PHP* y *Python* con el fin de encontrar errores de seguridad como desbordamiento de *buffer* o condiciones de carrera *TOCTOU* (*Time Of Check to Time Of Use*). *RATS* utiliza una base de datos para encontrar sentencias del código que pueden ser una vulnerabilidad potencial. El usuario puede decidir cual base de datos utilizar en el escaneo y que nivel de riesgo asumir. Por cada problema potencial encontrado, *RATS* provee la descripción del mismo y sugiere una solución al respecto. Además indica el grado de severidad del problema encontrado para facilitarle al auditor la asignación de prioridades. *RATS* ejecuta un análisis básico para descartar condiciones que no causan problemas de seguridad. El análisis con *RATS* no es completo ya que no encuentra todos los errores existentes y puede destacar como errores cosas que no lo son.

Owasp-python-static-analysis

Durante el año 2007, Dmitry Kozlov, Igor Konnov y Georgy Klimov comenzaron un prototipo de estilo de análisis estático para aplicaciones *web* de *Python*. Esta herramienta se basa en el proyecto *Pixy*. Es capaz de encontrar las vulnerabilidades de validación de entrada de seguridad en *Python* de aplicaciones basadas en *web*. Apoya subconjunto limitado de *Python*: funciones, módulos, clases y estructuras de datos, pero no generadores, comprensión, funciones lambda, etc. Cuenta con el apoyo sólo de *mod_python* para aplicaciones *web*. Actualmente se puede obtener el código fuente pero está en fase de desarrollo.

Pyntch

Pyntch es un analizador de código estático para el lenguaje de programación *Python.* Detecta errores en tiempo de ejecución antes de que se ejecute el código. Permite solucionar problemas con errores de tipo *TypeError* y *AttributeError*. *Pyntch* examina el código fuente y deduce todos los tipos posibles de las variables, atributos, argumentos de la función y los valores de retorno de cada función o método. A diferencia de otros analizadores no tiene en cuenta el estilo de código. Es bastante rápido, lo cual hace que pueda analizar varias líneas de código en poco tiempo. Actualmente soporta *Python* 2.x. Se puede decir que es un proyecto relativamente joven pero su desarrollo no ha sido lento, comenzó en el año 2008 y hasta el momento se han realizado cerca de 6 versiones, cada una con cambios importantes. (Para más información consultar el capítulo 6).

Pylint

Pylint es una herramienta de análisis estático que tiene como fundamental objetivo detectar errores así como también establecer un estándar de codificación. Ha tenido un desarrollo bastante continuo, empezó en el 2006 y hasta la actualidad se han desarrollado alrededor de 15 versiones. Muestra una serie de mensajes posterior al análisis donde se tienen en cuenta estadísticas tales como son: número de advertencias, errores encontrados en los diferentes archivos y una nota global basada en los errores encontrados, esto último puede motivar a los desarrolladores a mejorar su código. Permite adicionar

estilos de chequeos y su configuración es muy fácil. Otra característica es que advierte sobre el código duplicado, pero este análisis lo hace de forma muy simple y esto hace que sea fácil de engañar. Posibilita adicionar chequeos adicionando plugins, entre los que podemos mencionar *RawChecker* y *ASTNGChecker*. Compatible con cualquier versión superior a *Python* 2.2. (Para más información consultar el capítulo 6).

Considerando las características de cada analizador cada cual tiene sus particularidades en específico, así como sus ventajas y desventajas. Por lo que, para realizar una aplicación robusta que cumpla con los requisitos necesarios consideramos la utilización de dos de ellas, ya que, en conjunto pueden formar un bueno analizador. Las herramientas seleccionadas fueron: *Pylint* y *Pyntch, debido a que Pylint* posee casi las mismas características de *Pychecker*, pero es fácilmente configurable y analiza el estilo de código, garantizando con esto el uso de estándar de codificación en los diferentes proyecto a analizar. En el caso de *Pyntch* es una herramienta rápida que permite la revisión de archivos en poco tiempo y también permite la detección de errores de tipo *TypeError* y *AttributeError*, lo cuales serían capturados sin necesidad de que se ejecute la aplicación. Además, hay que tener en cuenta el desarrollo de las mismas ha sido continuo. Pensamos que para un futuro se debe tener en cuenta el proyecto *Owasp-python-static-analysis* debido a que el mismo todavía no ha sido revisado y esto hace que no sea confiable, pero es una herramienta a tener en cuenta para un futuro muy próximo, y más con el desarrollo actual del *internet* y el gran número de páginas *web* existentes. Es importante destacar que las herramientas de análisis estático no serán la solución a todos los problemas pero las mismas contribuyen a la realización a aplicaciones más seguras.

1.3. Metodología y Herramientas

Una vez que se ha realizado la selección de las herramientas de análisis estático a utilizar en el sistema a implementar, se dedicará la totalidad del capítulo actual a seleccionar las herramientas y metodología disponibles. Para ello, se harán comparaciones basadas en parámetros esenciales para lograr un mejor funcionamiento del sistema.

1.3.1. Metodología de Desarrollo

***Rational Unified Process* (*RUP*)**

El Proceso Unificado es un proceso de desarrollo de *software*. Un proceso de desarrollo de s*oftware es* el conjunto de actividades necesarias para transformar los requisitos de un usuario en un sistema *software*. Sin embargo, el Proceso Unificado es más que un simple proceso; es un marco de trabajo genérico que puede especializarse para una gran variedad de sistemas de *software*, para diferentes áreas de aplicación, diferentes tipos de organización, diferentes niveles de aptitud y diferentes tamaños de proyectos. El Proceso Unificado está basado en componentes, lo cual quiere decir que el sistema *software* en construcción está formado por componentes *software* interconectados a través de interfaces bien definidas. (6)

Características Principales

- Guiado por casos de usos.
- Centrado en la Arquitectura.
- Iterativo e Incremental.
- Basado en Componentes.
- Utilización de un lenguaje único: *UML*.

Fases

- Inicio.
- Elaboración.
- Construcción.
- Transición.

Principales Elementos.

- Trabajadores, personas o entes involucrados en cada proceso.
- Actividades, son los procesos que se llegan a determinar en cada iteración.
- Artefactos, puede ser un modelo, un documento, o un elemento de un modelo.

Flujos de Trabajo.

- Modelado del negocio.
- Requisitos.
- Análisis y Diseño.
- Implementación.
- Prueba.
- Despliegue.
- Administración de proyecto.
- Configuración y control de cambios.
- Entorno.

RUP es una metodología robusta y bien definida, probada en diferentes proyectos en la UCI, pues esta realiza un levantamiento exhaustivo de requerimientos, determina cuáles son los defectos del proyectos desde su fase inicial, realiza un análisis y diseño lo más completo posible. Posee una amplia documentación en diferentes idiomas, así como también es la metodología estudiada.

1.3.2. Lenguaje de Modelado

Lenguaje Unificado de Modelado (*UML*).

El éxito de los proyectos de desarrollo de aplicaciones o sistemas se debe a que sirven como enlace entre quien tiene la idea y el desarrollador. El *UML* (Lenguaje Unificado de Modelado) es una herramienta que cumple con esta función, ya que le ayuda a capturar la idea de un sistema para comunicarla posteriormente a quien esté involucrado en su proceso de desarrollo; esto se lleva a cabo mediante un conjunto de símbolos y diagramas. Cada diagrama tiene fines distintos dentro del proceso de desarrollo. (7)

Conforme aumenta la complejidad del mundo, los sistemas informáticos también deberán crecer en complejidad. Si se desea crear sistemas que se involucren con este nuevo milenio, ¿cómo manejar tanta complejidad? La clave está en organizar el proceso de diseño de tal forma que los analistas, clientes,

desarrolladores y otras personas involucradas en el desarrollo del sistema lo comprendan y convengan con él. El *UML* proporciona tal organización.

La necesidad de diseños sólidos ha traído consigo la creación de una notación de diseño que los analistas, desarrolladores y clientes acepten como pauta. El *UML* es esa misma notación.

Además, el *UML* es una de las herramientas mas utilizadas en el mundo actual del desarrollo de sistemas, debido a que permite a los creadores de sistemas generar diseños que capturen sus ideas en una forma convencional y fácil de comprender para luego comunicarlas a otras personas.

Este lenguaje está compuesto por diversos elementos gráficos que se combinan para conformar diagramas. Estos diagramas son: Diagrama de clases, de objetos, de casos de uso, de estados, secuencias, de actividades, de colaboraciones, de componentes y de distribución (despliegue).

El *UML* proporciona características que le permiten organizar y extender los diagramas. Esto se realiza mediante paquetes dada la necesidad de organizar los elementos de un diagrama en un grupo o tal vez para mostrar que ciertas clases o componentes son parte de un subsistema en particular. Estos paquetes se representan como una carpeta tabular.

1.3.3. Herramienta *CASE*

Para el modelado del proceso del negocio, se utilizará como Herramienta *CASE*, *Visual Paradigm For UML*. Esta herramienta profesional soporta el ciclo de vida completo del desarrollo de *software*: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. El *software* de modelado *UML* ayuda a una más rápida construcción de aplicaciones de calidad y a un menor coste. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación.

Además, *Visual Paradigm For UML* es una Herramienta Case Cruzado de Ciclo de Vida y se caracteriza por lo siguiente:

- Soporta las últimas versiones de *UML* y la Notación y Modelado de Procesos de Negocios.
- En adición al soporte de modelado *UML,* esta herramienta provee el modelado de procesos de negocios, además de un generador de mapeo de objetos-relacionales para los lenguajes de programación *Java* .*NET* y *PHP*.
- Para desarrolladores independientes existe una versión llamada *Community Edition* en la que se caracteriza por ser de uso No Comercial.
- Está disponible en varias ediciones, cada una destinada a unas necesidades: *Enterprise*, *Professional*, *Community*, *Standard*, *Modeler* y *Personal*.
- Ayuda a los equipos de desarrollo de software para sobresalir todo el modelo de acumulación de trabajo y desplegar el proceso de desarrollo de software, lo que permite maximizar y acelerar tanto las contribuciones individuales como las de equipo.
- Proporciona el código y compatibilidad hasta con 10 lenguajes.

Visual Paradigm For UML apoya para la importación y exportación de *XML* de versiones 1,0, 1,2 y 2,1. También tiene conexión con *Rational Rose* en sus archivos de proyecto (.MDL / .CAT), que además pueden ser importados a *Visual Paradigm UML* a través de esta importante característica. (8)

Sin importar el IDE, *Visual Paradigm* ofrece los siguientes beneficios:

- Navegación intuitiva entre el código y el modelo.
- Poderoso generador de documentación y reportes *UML PDF*/*HTML*/*MS Word*.
- Demanda en tiempo real, modelo incremental de viaje redondo y sincronización de código fuente.
- Superior entorno de modelado visual.
- Soporte completo de notaciones *UML*.
- Diagramas de diseño automático sofisticado.

Visual Paradigm For UML proporciona soporte a varios lenguajes en generación de código e ingeniería inversa a través de plataformas java. Estos lenguajes son:

- *C++*
- *CORBA IDL*
- *PHP*
- *XML*
- *Schema*
- *Ada*
- *Python.*

1.3.4. Herramienta para la gestión de los datos

Sistema de Gestión de Bases de Datos (SGBD).

Consiste en un conjunto de programas, procedimientos y lenguajes que nos proporcionan las herramientas necesarias para trabajar con una base de datos. Incorporar una serie de funciones que nos permita definir los registros, sus campos, sus relaciones, insertar, suprimir, modificar y consultar los datos. (*Access*, *SQL Server*, *Informix*, etc.) (9)

PostgreSQL

Características de *PostgreSQL*.

- Implementación del estándar *SQL*92/*SQL*99.
- Soporta distintos tipos de datos: además del soporte para los tipos base, también soporta datos de tipo fecha, monetarios, elementos gráficos, datos sobre redes, cadenas de *bits*, etc. También permite la creación de tipos propios.
- Permite la declaración de funciones propias, así como la definición de disparadores.

- Soporta el uso de índices, reglas y vistas.
- Incluye herencia entre tablas (aunque no entre objetos, ya que no existen), por lo que a este gestor de bases de datos se le incluye entre los gestores objeto-relacionales.
- Permite la gestión de diferentes usuarios, como también los permisos asignados a cada uno de ellos.

MySQL

Características de *MySQL*

- Aprovecha la potencia de sistemas multiprocesadores, gracias a su implementación multihilo.
- Soporta gran cantidad de tipos de datos para las columnas.
- Dispone de *APIs* en gran cantidad de lenguajes (*C*, *C*++, *Java*, *PHP*, etc.).
- Gran portabilidad entre sistemas.
- Gestión de usuarios y *passwords*, manteniendo un muy buen nivel de seguridad en los datos.

Se llegó a la conclusión que ambos gestores son potentes y de los más utilizados en el mundo debido sus potencialidades. Pero, se decidió seleccionar *PostgreSQL* debido a que posee una gran escalabilidad. Es capaz de ajustarse al número de *CPUs* y a la cantidad de memoria que posee el sistema de forma óptima, haciéndole capaz de soportar una mayor cantidad de peticiones simultáneas de manera correcta, de esta forma se prefiere la escalabilidad respecto a la velocidad, ya que *MySQL* posee mayor velocidad haciéndolo esto unos de los gestores con mayor rendimiento.

1.3.5. Servidor *Web*

El Servidor Apache *HTTP* es un servidor *web* de tecnología *Open Source* sólido y para uso comercial desarrollado por la Apache *Software Foundation.* El servidor w*eb* Apache junto con el módulo *mod_rewrite* puede convertirse en una herramienta muy jugosa para crear páginas con enlaces amigables para los buscadores. Además, “su configurabilidad, robustez y estabilidad hacen que cada vez millones de servidores reiteren su confianza en este programa”. (10)

¿Pero, por qué tiene tanta popularidad este *software* libre grandemente reconocido en muchos ámbitos empresariales y tecnológicos? Algunas de las razones se expresan a continuación:

- Corre en una multitud de Sistemas Operativos, lo que lo hace prácticamente universal.
- Apache es una tecnología de código fuente abierto gratuita. El hecho de ser gratuita es importante pero no tanto como que se trate de código fuente abierto. Esto le da una transparencia a este *software* de manera que si queremos ver que es lo que estamos instalando como servidor, lo podemos saber, sin ningún secreto, sin ninguna puerta trasera.
- Apache es un servidor altamente configurable de diseño modular. Es muy sencillo ampliar las capacidades del servidor *web* Apache. Actualmente existen muchos módulos para Apache que son adaptables a este, y están ahí para que se instalen cuando se necesiten. Otra cosa importante es que cualquiera que posea una experiencia decente en la programación de *C* o *Perl* puede escribir un módulo para realizar una función determinada.
- Apache trabaja con *Perl*, *PHP* y otros lenguajes de *script*. También trabaja con *Java* y páginas *jsp*. Teniendo todo el soporte que se necesita para tener páginas dinámicas.
- Apache te permite personalizar la respuesta ante los posibles errores que se puedan dar en el servidor. Es posible configurar Apache para que ejecute un determinado *script* cuando ocurra un error en concreto.
- Tiene una alta configurabilidad en la creación y gestión de *logs*. Apache permite la creación de ficheros de *log* a medida del administrador, de este modo se puede tener un mayor control sobre lo que sucede en el servidor.

Como se pudo observar, el servidor *web* Apache es la opción ideal para el sistema que se desea implementar, su robustez, configurabilidad, estabilidad y las demás características descritas, hacen de él un excelente servidor.

1.4. Lenguajes de Programación

Desde los inicios de *Internet*, fueron surgiendo diferentes demandas por los usuarios y se dieron soluciones mediante lenguajes estáticos. A medida que paso el tiempo, las tecnologías fueron desarrollándose y surgieron nuevos problemas a dar solución. Esto dio lugar a desarrollar lenguajes de programación para la *web* dinámicos que permitieran interactuar con los usuarios y utilizaran sistemas de Bases de Datos (11). Actualmente existen diferentes lenguajes de programación dinámicos para desarrollar en la *web* como por ejemplo: *ASP*, *JSP*, *PHP*, *Ruby*, *Python*, entre otros. ¿Pero, cuál es el más indicado para desarrollar la aplicación? Debido a las características que presenta el sistema a implementar, fue seleccionado el lenguaje de programación *PHP*. En el epígrafe que viene a continuación se profundizan sobre las razones de su selección.

1.4.1. *PHP*

PHP es un lenguaje de *script* interpretado que se ejecuta del lado del servidor, utilizado para la generación de páginas *web* dinámicas, embebidas en páginas *HTML* y ejecutadas en el servidor. *PHP* no necesita ser compilado para ejecutarse. Para su funcionamiento necesita tener instalado Apache o *IIS* con las librerías de *PHP*. La mayor parte de su sintaxis ha sido tomada de C, *Java* y *Perl* con algunas características específicas. Los archivos cuentan con la extensión (*php*). (11)

Una de sus características más potentes es su soporte para gran cantidad de bases de datos. Entre su soporte pueden mencionarse *InterBase*, *mSQL*, *MySQL*, *Oracle*, *Informix*, *PostgreSQL*, entre otras. *PHP* también ofrece la integración con varias bibliotecas externas, que permiten que el desarrollador haga casi cualquier cosa desde generar documentos en formato "*pdf*" hasta analizar código *XML*. (12)

Como producto de código abierto, *PHP* goza de la ayuda de un gran grupo de programadores, permitiendo que los fallos de funcionamiento se encuentren y se reparen rápidamente. También, es utilizado en aplicaciones *web* relacionadas por algunas de las organizaciones más prominentes tales como *Mitsubishi*, *Redhat*, *Der Spiegel, MP3-Lycos*, *Ericsson* y *NASA*. (12)

Algunas ventajas y desventajas que presenta este lenguaje son:

Ventajas:

- Muy fácil de aprender.
- Se caracteriza por ser un lenguaje muy rápido.
- Soporta en cierta medida la orientación a objeto. Clases y herencia.
- Es un lenguaje multiplataforma: *Linux*, *Windows*, entre otros.
- Capacidad de expandir su potencial utilizando módulos.
- Posee documentación en su página oficial la cual incluye descripción y ejemplos de cada una de sus funciones.
- Es libre, por lo que se presenta como una alternativa de fácil acceso para todos.
- Incluye gran cantidad de funciones.
- No requiere definición de tipos de variables ni manejo detallado de bajo nivel.

Desventajas:

- Se necesita instalar un servidor *web*.
- Todo el trabajo lo realiza el servidor y no delega al cliente. Por tanto puede ser más ineficiente a medida que las solicitudes aumenten de número.
- La legibilidad del código puede verse afectada al mezclar sentencias *HTML* y *PHP*.

Seguridad

PHP es un poderoso lenguaje e intérprete, ya sea incluido como parte de un servidor *web* en forma de módulo o ejecutado como un binario *CGI* separado, es capaz de acceder a archivos, ejecutar comandos y abrir conexiones de red en el servidor. Estas propiedades hacen que cualquier cosa que sea ejecutada en un servidor *web* sea insegura por naturaleza. *PHP* está diseñado específicamente para ser un lenguaje más seguro para escribir programas *CGI* que *Perl* o *C*, y con la selección correcta de opciones de

configuración en tiempos de compilación y ejecución, y siguiendo algunas buenas prácticas de programación. (11)

1.5. Framework

¿Qué es un *Framework*?

El término *framework*, se refiriere a una estructura *software* compuesta de componentes personalizables e intercambiables para el desarrollo de una aplicación. “Un *framework web*, por tanto, se puede definir como un conjunto de componentes (por ejemplo: clases en java y descriptores y archivos de configuración en *XML*), un diseño reutilizable que facilita y agiliza el desarrollo de sistemas *web*.” (13) Los objetivos principales que persigue un *framework* son: acelerar el proceso de desarrollo, reutilizar código ya existente y promover buenas prácticas de desarrollo, como el uso de patrones.

La mayoría de los *frameworks web* implementan el patrón Modelo-Vista-Controlador (MVC), el cual permite una arquitectura que ofrece mayor interactividad con los usuarios. Este patrón organiza la aplicación en tres capas, la primera capa es un modelo que representa los datos de la aplicación y sus reglas de negocio, la segunda capa es un conjunto de vistas que representa los formularios de entrada y salida de información, la tercera capa conocida como controladora es la que procesa las peticiones de los usuarios y controla el flujo de ejecución del sistema.

1.5.1. KumbiaPHP

KumbiaPHP es un *framework web* libre escrito en *PHP5*. Basado en las mejores prácticas de desarrollo *web*, usado en *software* comercial y educativo, *Kumbia* fomenta la velocidad y eficiencia en la creación y mantenimiento de aplicaciones *web*, reemplazando tareas de codificación repetitivas por poder, control y placer. (14) El número de prerrequisitos para instalar y configurar es muy pequeño, apenas *Unix* o *Windows* con un servidor *Web* y *PHP5* instalado. *KumbiaPHP* es compatible con motores de base de datos como *MySQL*, *PostgreSQL* y *Oracle*.

KumbiaPHP representa una alternativa para proyectos en *PHP* con características como:

- Sistema de Plantillas sencillo
- Administración de *Cache*
- *Scaffolding* Avanzado
- Modelo de Objetos y Separación MVC
- Soporte para *AJAX*
- Generación de Formularios
- Componentes Gráficos
- Seguridad

Está basado en los siguientes conceptos:

- Compatible con muchas plataformas
- Fácil de instalar y configurar
- Fácil de aprender
- Listo para aplicaciones comerciales
- Convención sobre Configuración
- Simple en la mayor parte de casos pero flexible para adaptarse a casos más complejos
- Soportar muchas características de Aplicaciones *Web* actuales
- Soportar las prácticas y patrones de programación más productivos y eficientes
- Producir aplicaciones fáciles

El principal principio es producir aplicaciones que sean prácticas para el usuario final y no solo para el programador. La mayor parte de tareas que le quiten tiempo al desarrollador deberían ser automatizadas por *KumbiaPHP* para que él pueda enfocarse en la lógica de negocio de su aplicación.

Después de haber descrito algunas características de *KumbiaPHP*, surge la siguiente interrogante: ¿cuál es la diferencia con los demás *frameworks*? La gran diferencia radica en que *KumbiaPHP* presenta un alto rendimiento en cuanto a rapidez de respuesta. Una muestra de ello se representa en la serie de

benchmark[2] *realizados (con fecha de publicación* 27 de marzo del 2009) a *frameworks* mundialmente conocidos como *Symfony*, *Zend*, *CakePHP* y a *KumbiaPHP* versión 1.0 *stable*.

"Cada *framework* que se le aplico la prueba se hizo utilizando las más mínimas medidas de configuración y control posible, de manera de hacer efectiva un "Hola Mundo"*.* El enfoque minimalista mide la capacidad de respuesta de los componentes del *framework*. No hay código de aplicación para ejecutar el controlador de las acciones en el *framework* de cada uno haga lo menos posible para llamar a una vista. Esto muestra el máximo rendimiento posible, añadiendo código sólo reducirá la respuesta." (15) Como servidor, se utilizo Apache, para las mediciones de las solicitudes por segundo.

El servidor propuesto para realizar dichos *benchmark* presenta las siguientes características:

- *Intel*(R) *Core*(TM)2 *Duo CPU* T7300 @ 2.00GHz
- *3GiB* Memoria *RAM*
- *160GiB* Disco Duro.
- *Server version*: Apache/2.2.11
- S.O. *Debian Squeeze*
- *ApacheBench*, *Version* 2.3"

En el **Anexo 2** se encuentran reflejados los resultados de las pruebas realizadas, destacándose por encimas de los demás el *framework KumbiaPHP* con un índice de 34.07 req/seg, distante del 23.71 logrado por el *framework Zend* 1.6.2., ocupante del segundo lugar.

Con este resultado finaliza la investigación para la selección del *framework* adecuado para realizar la aplicación, pues un parámetro indispensable en su funcionamiento lo conforma el tiempo de respuesta que demora el sistema cuando analizan los diferentes proyectos que son cargados por los usuarios.

[2] Un *benchmark* es el resultado de la ejecución de un programa informático o un conjunto de programas en una máquina, con el objetivo de estimar el rendimiento de un elemento concreto o la totalidad de la misma, y poder comparar los resultados con máquinas similares.

1.6. Herramienta para la Transferencia de Datos

El sistema a desarrollar presenta como una necesidad subir los proyectos de los usuarios hacia algún lugar seguro para su futuro análisis. Un protocolo que garantiza la transferencia segura hacia un destino determinado lo constituye: *VSFTPD* (*Very Secure FTP Daemon*). A continuación, se abordará sobre este importante servidor, destacando sus características principales.

1.6.1. Servidor *VSFTPD*

FTP es utilizado para transferir documentos o *software* de un sitio a una cuenta en *internet* o para un dispositivo de almacenamiento (disco duro, memoria *flash,* entre otras). En un principio, solo fue para computadoras con sistema operativo *UNIX,* pero con el tiempo ha sido posible utilizarlo en diferentes entornos. Diversas compañías y universidades tienen sitios *FTP* en los cuales se puede obtener información y aplicaciones relacionadas con la entidad.

VSFTPD

VSFTPD (*Very Secure FTP Daemon*) es una programática utilizada para implementar servidores de archivos a través del protocolo *FTP*. Se distingue principalmente porque sus valores por defecto son muy seguros y por su sencillez en la configuración, comparado con otras alternativas como **Wu-ftpd**. Actualmente se presume que *VSFTPD* es quizá el servidor *FTP* más seguro del mundo. (16)

Además, es un servidor con licencia *GPL* para sistemas *UNIX*, incluyendo *Linux* y presenta gran rapidez. Otras características que presenta este servidor son:

Características

- Configuración de *IP* virtuales.
- Usuarios virtuales.
- Encriptación mediante *SSL*.
- *IPv6*.

- Potente para la configuración de usuarios.

1.7. Conclusiones

Con la culminación de este capítulo se crearon las condiciones preliminares para comenzar el ciclo de desarrollo del sistema a implementar. En primer lugar, se realizó un estudio detallado sobre algunas de las herramientas de análisis estático que existen en el mundo, con fin de seleccionar la más adecuada. En este caso, resultaron seleccionadas dos: *Pylint* y *Pyntch,* debido a sus potencialidades; integrando los resultados de ambos análisis en uno solo para obtener uno mas óptimo. En un segundo instante se analizaron las metodologías y herramientas indispensables para garantizar que la futura aplicación realice un buen funcionamiento para con los usuarios interesados en recibir los servicios de la misma. El lenguaje a utilizar será *PHP* con *framework* de desarrollo *KumbiaPHP*, como gestor de base de datos fue seleccionado *PostgreSQL* y el servidor *web* para la aplicación es Apache. La herramienta *CASE* propuesta es *Visual Paradigm*, utilizando lenguaje de modelado *UML* y se contará como metodología de desarrollo, la propuesta por *RUP*.

Capítulo 2
CARACTERÍSTICAS DEL SISTEMA

2.1. Introducción

En el presente capítulo se realiza una descripción de la propuesta de solución para el sistema de análisis estático de vulnerabilidades para *Python*. Debido a la poca organización de los procesos del negocio y para lograr mejor el contexto en el cual se desarrolla el sistema, se acuerda desarrollar un Modelo de Dominio donde se expone un marco conceptual y se establecen las relaciones entre los conceptos que lo conforman. Por otra parte, se especifican los requerimientos funcionales y no funcionales, agrupándose los primeros en casos de uso, con el fin de estructurar el Diagrama de Casos de Uso del sistema. Finalmente, se mostrará una descripción de los Casos de Uso, así como los actores que intervienen en el sistema.

2.2. Modelo de Dominio

El Modelo de Dominio o Modelo Conceptual, constituye una representación visual para el usuario de los conceptos u objetos del mundo real significativos para un problema o área de interés. Representa conceptos del mundo real, no de los componentes de *software*. A diferencia de los otros modelos propuestos por la metodología de desarrollo *RUP*, este no está formado por un conjunto de diagramas que describen clases u objetos de *software* con responsabilidades, sino que puede considerarse como un diccionario visual de las abstracciones más relevantes en el dominio de definición del producto. Aprovechando las ventajas de los Diagramas *UML* para representar conceptos, el Modelo de Domino se muestra en forma de diagramas de clases donde figuran las principales definiciones y roles del sistema en cuestión.

Se decide realizar un Modelo de Dominio debido a que no pueden ser identificados los procesos de negocio asociados al contexto del sistema, como consecuencia de la poca estructuración y fronteras de los mismos. A continuación, es presentada la propuesta de Modelo de Domino elaborada.

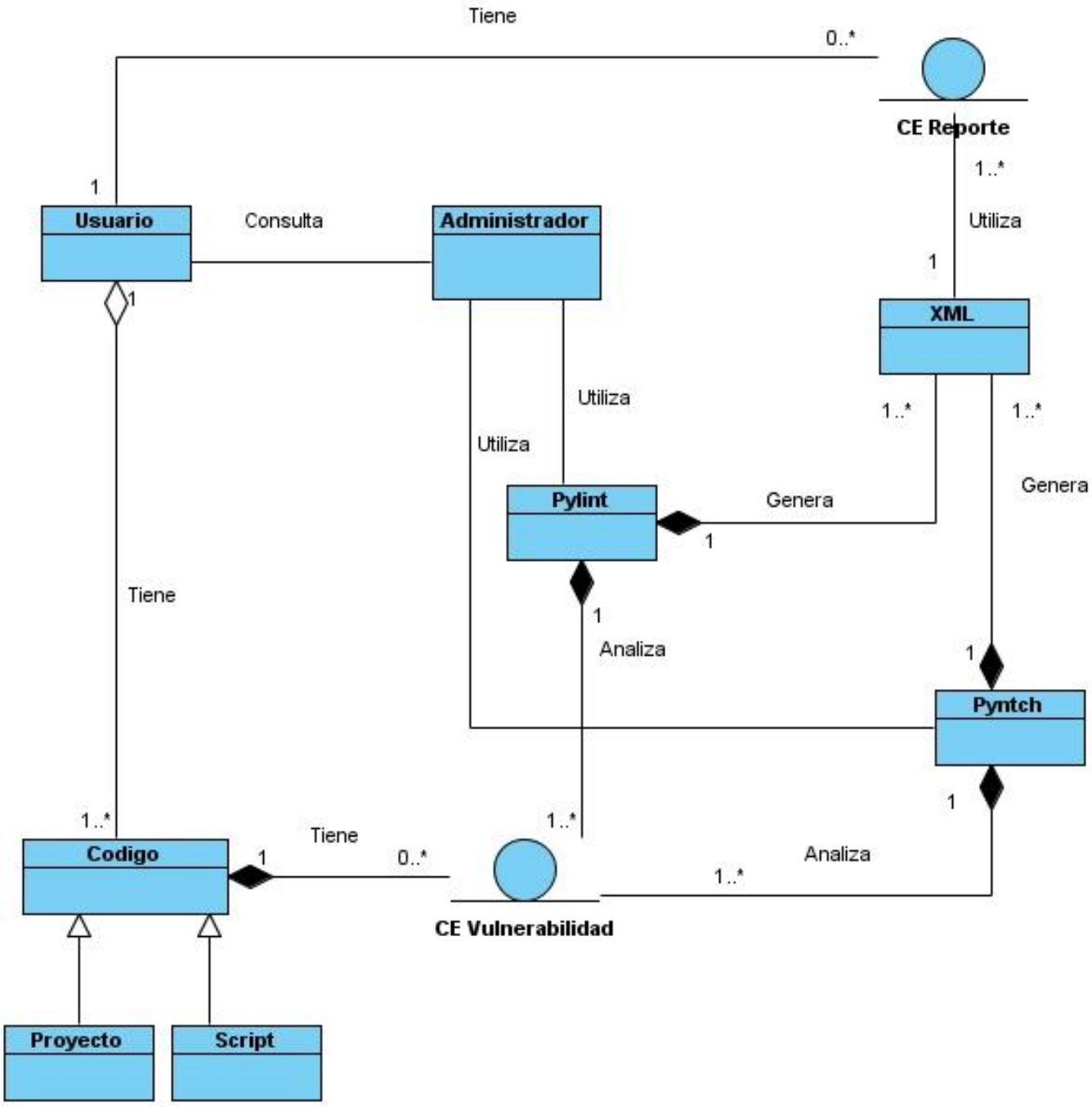


Fig. 2.1 Modelo de Dominio

2.3. Propuesta del Sistema

Un sistema informático está formado por varios elementos como hardware, *software* y soporte humano. Generalmente la elaboración de un sistema comienza conformando una visión global del ambiente donde será implantado, el cual está comprendido en el Modelo de Dominio, de gran utilidad para analizar cada uno de los conceptos reconocidos, así como sus interrelaciones con el fin de identificar todos los requerimientos que se deben cumplir. Una vez que estos hayan sido identificados, el Modelo del Sistema puede ser realizado.

2.3.1. Relación de los requerimientos

La especificación de los requerimientos de *software* constituye un elemento de vital importancia para la elaboración de un *software* de calidad superior. El Glosario Estándar *de* Terminología de Ingeniería de *Software* (*IEEE: Standard Glossary of Software Engineering Terminology*) define los requerimientos de *software* como condiciones o capacidades que deben estar presentes en un sistema o componente de este, para satisfacer un contrato, estándar, especificación u otro documento formal. Es necesario hacer énfasis en la precisión con que se debe realizar esta tarea por cumplir un papel primordial en el proceso de producción de *software*, pues se enfoca en un área fundamental: la definición de lo que se desea producir, permitiendo describir con mayor claridad el comportamiento del sistema, minimizando los problemas derivados de su desarrollo.

2.3.2. Listado de los requerimientos funcionales

Los requerimientos funcionales determinan de forma clara y concisa las acciones que debe ser capaz de hacer el sistema, éstas se corresponden con opciones que ejecutará el *software*, operaciones realizadas de forma oculta o condiciones extremas a determinar por el sistema. En el sistema que se diseña, los requisitos funcionales son los siguientes:

RF1. Autenticar Usuarios

Mediante este requerimiento se le dará la posibilidad al usuario de ingresar al sistema. Para ello, este debe introducir datos como: usuario y contraseña del dominio UCI. El sistema comprobará la validez de estos datos, dándole acceso en caso positivo y una vez que haya leído las condiciones de uso.

RF2. Subir Ficheros

Este requerimiento funcional le permitirá al usuario subir proyectos que cuenten con los formatos exigidos. Para esto, el usuario debe buscar el proyecto a analizar y una vez cargado, el mismo se guardará automáticamente en el sistema.

RF3. Subir Código

Este requerimiento funcional representa otra vía para subir proyectos. En este caso, el usuario deberá copiar su código en un área de texto que se encuentra en la interfaz correspondiente, para luego guardarlo en el sistema utilizando el botón que señaliza dicha acción.

RF4. Revisar Proyecto

Mediante este requerimiento el usuario podrá realizar la revisión de un proyecto que ha sido subido por alguna de las vías descritas en los Requisitos Funcionales 2 y 3. Luego, el usuario puede pasar a la realización del análisis. Una vez realizado el mismo, el sistema notificará su culminación y se generará un reporte en caso de existir vulnerabilidades.

RF5. Mostrar Reportes

Este requerimiento le permitirá al usuario examinar un reporte que ha sido generado después de la revisión de un proyecto. Para ello, el usuario debe seleccionar la opción “Reportes” de la página principal y luego de que se han mostrado los reportes, este puede elegir el que desee observar.

RF6. Eliminar Reportes

Mediante este requerimiento al usuario se le permitirá eliminar los reportes que han sido generados por la revisión de sus proyectos. Para realizar dicha acción debe examinar los reportes existentes (Ver RF5), y seleccionar la opción “Eliminar”.

RF7. Mostrar Estadísticas
Este requerimiento le permitirá al usuario observar las estadísticas resultantes de las revisiones de sus proyectos, así como las estadísticas generales de acuerdo a los errores presentados en todos los proyectos analizados en el sistema.

RF8. Cancelar análisis en curso
Este requerimiento le permitirá al usuario cancelar el análisis en curso de algún proyecto.

RF9. Generar Reporte
Mediante este requerimiento el sistema debe generar un reporte como resultado del análisis de un proyecto (Ver RF4).

RF10. Descargar Reporte
Este requerimiento le permitirá al usuario descargar una copia en formato *PDF* de un reporte generado anteriormente. Para realizar dicha descarga debe examinar los reportes existentes y elegir el reporte deseado (Ver RF5).

2.3.3. Listado de los requerimientos no funcionales

Los requerimientos no funcionales son propiedades o cualidades que el producto debe tener. Debe pensarse en estas propiedades como las características que hacen al producto atractivo, usable, rápido y confiable.

Apariencia o interfaz externa

La interfaz del sistema debe ser agradable y fácil de utilizar, de forma que no se necesite mucha práctica para la manipulación de la misma.

Debido al gran uso que presupone la aplicación, es necesario que se combinen correctamente los colores, tipo de letra y tamaño.

La interfaz debe brindar una ayuda del uso de la misma al usuario.

Usabilidad

La aplicación debe brindar la posibilidad al usuario de realizar el análisis de su(s) proyecto(s).

Todas las opciones brindadas deben ser fácilmente entendibles.

La manipulación del sistema podrá ser usado por toda persona con conocimientos básicos de ambientes *web*.

Rendimiento

La aplicación requiere de un procesamiento elevado de información debido al análisis que se realiza por parte de la herramienta de código estático, la misma sin embargo, tiene niveles de desempeño aceptables debido al algoritmo de búsqueda de vulnerabilidades que implementa. El tiempo de respuesta del sistema está en dependencia de la cantidad de archivos a analizar, el promedio para la respuesta al usuario de un archivo es de aproximadamente diez segundos. La cantidad de usuarios conectados simultáneamente al sistema es de aproximadamente quince usuarios. Los clientes no deben necesitar muchos recursos para ejecutar la aplicación.

Soporte

La aplicación debe ser extensible y bien documentada para facilitar el soporte de la misma en caso que sea necesario.

Configuración requerida en *PHP*:

- Activar o instalar la extensión para *ldap*.
- Activar o instalar la extensión para *PostgreSQL*.

- Activar o instalar la extensión para descompactar archivos .*RAR*.

Configuración requerida para servidor de Base de datos *PostgreSQL*:

Ajustar la conexión desde direcciones IP de otras computadoras.

Configuración requerida para servidor *FTP VSFTPD* (Consultar documento de despliegue):

- Opción anonymous_enable=NO.
- Opción local_enable=YES.
- Opción local_enable=YES.
- Activar complemento para SSL.

Configuración requerida para servidor Apache:

- Activar complemento para *SSL*.
- Activar módulo de reescritura *URL mod_rewrite*.
- Restricción de cantidad máxima de usuarios conectados al servidor de quince usuarios.
- Eliminar los módulos innecesarios.

Configuración para el cliente:

Se debe tener instalado un navegador *web* con *JavaScript* habilitado y *Acrobat Reader* en cualquier versión instalado.

Seguridad.

El acceso al sistema será controlado por usuario y contraseña, mediante el dominio *UCI*.

Los administradores son los únicos que tienen acceso a la parte administrativa del sistema.

Los reportes son personales, ningún usuario está autorizado a ver otros reportes.

La navegación por el sistema y la comunicación con el servidor *FTP* es mediante el protocolo de capa de conexión segura *SSL*.

Confiabilidad

Las salidas del sistema tienen que ser totalmente seguras y precisas.

Los reportes deben ser resguardados por la importancia que representan.

Debe garantizarse las salvas de seguridad de la Base de Datos para la recuperación en caso de fallas.

Aspectos legales

La aplicación y la documentación de la misma, pertenecen a la Universidad de las Ciencias Informáticas y sólo a ésta se le permite su uso.

Las herramientas de desarrollo son libres, por tanto, las licencias están avaladas.

Diseño e Implementación

Los lenguajes de programación a utilizar son *Python* por parte del analizador, *PHP* como lenguaje del lado del servidor y *JavaScript* del lado del cliente.

El *framework* de desarrollo para *PHP* que se requiere es *Kumbia 1.0 stable.*

El estilo arquitectónico será el MVC para la separación de los datos, interfaz de usuario y lógica de

negocio.

Hardware

Servidores:

Se requiere un servidor *web* que tenga como mínimo 1 *GB* de memoria *RAM*, procesador *Pentium IV* ó superior y 5 *GB* de disco duro disponibles.

Se requiere un servidor de Base de Datos y otro de *FTP*, los que deben disponer de un disco duro de 160 *GB* de capacidad como mínimo.

Se requiere tarjeta de red.

Clientes:

Se requiere tarjeta de red.

Computadora con al menos 256 *MB* de memoria *RAM*, al menos 100 *MB* de disco duro, procesador *Pentium* II ó superior.

Software

Se requiere la instalación de *Linux*, específicamente Ubuntu Server 9.10.

El servidor *Web* es Apache 2.0 con soporte para *SSL* tanto para la navegación como la comunicación con el servidor *FTP.*

El servidor de Base de Datos es *PostgreSQL* 8.3 ó superior.

El servidor *FTP* es *VSFTPD 2.2.2*.

El lenguaje de programación es *Python*, versión 2.6.

Se debe instalar de igual forma, el *Framework KumbiaPHP* versión 1.0 *stable.*

Las versiones de los analizadores de código estático son *Pylint* 0.19 y *Pyntch* 20091028.

Las computadoras clientes deben tener instalado navegador *Mozilla Firefox* versión 2.0.0.1 ó superior. De igual forma, se necesita *Acrobat Reader* para la lectura de reportes.

2.4. Modelo de Casos de Uso del Sistema

El modelo de Casos de Uso del sistema es un artefacto de la Ingeniería de *Software* que describe bajo la forma de acciones y reacciones el comportamiento del sistema desde el punto de vista del usuario, permitiendo de esta forma el establecimiento de un acuerdo entre clientes y desarrolladores sobre las condiciones y requerimientos que debe cumplir el sistema. Este modelo está formado por actores, casos de uso y las relaciones que se establecen entre ellos, es decir, representa gráficamente a los procesos y la interacción con los actores y constituye una entrada de gran valor para las siguientes fases de construcción de *software*.

A continuación, se realiza la descripción de los actores que intervienen en la propuesta, así como los casos de uso que son objetos de automatización.

2.4.1. Actores del Sistema. Descripción

Un actor del sistema es una persona o abstracción de un *software* que interactúa de alguna manera con el sistema: puede intercambiar información con él, ser un recipiente pasivo de información y representa el rol que juegan una o varias personas, un equipo o un sistema automatizado. A continuación, se presenta el actor del sistema:

Actor	Justificación
Usuario_Sistema	Actor que constituye un usuario del dominio UCI el cual interactúa con el sistema con el fin de realizar las actividades referentes a la revisión del proyecto, tales como subir los archivos, escribir el código y enviar la orden de revisarlo. Así como también podrá ver los diferentes reportes y las estadísticas generadas de los reportes.

2.4.2. Diagrama de Casos de Usos del Sistema

En este diagrama se representan las relaciones existentes entre el actor y los casos de uso. A continuación se muestra el diagrama:

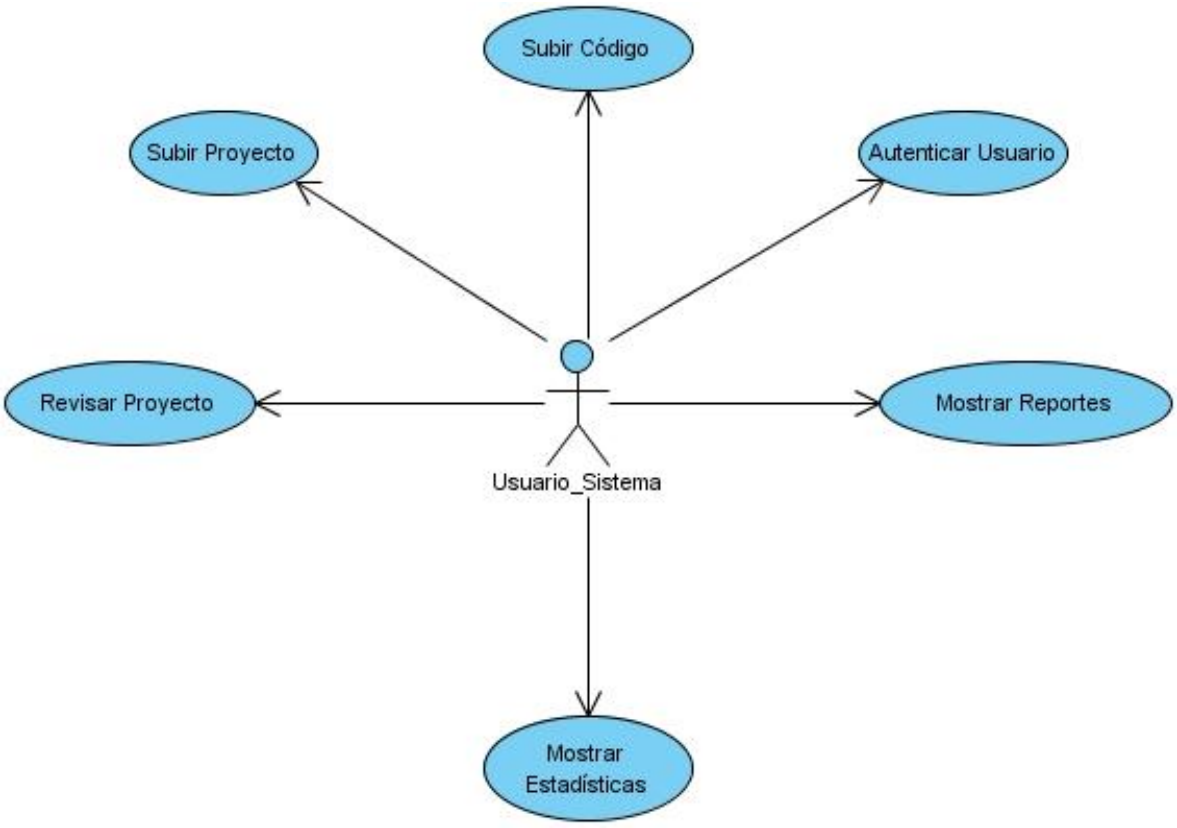


Fig. 2.2 Diagrama de Casos de Uso del Sistema

2.4.3. Descripción textual de los Casos de Uso

Caso de Uso:	Autenticar Usuarios
Actores:	Usuario (inicia)
Resumen:	El CU se inicia cuando el usuario introduce sus datos en el sistema para autenticarse y según los datos introducidos, le son asignados al usuario sus privilegios. En caso de que no sean correctos se le niega el acceso mostrando un mensaje de error.

Precondiciones:	Debe ser un usuario del dominio UCI
Poscondiciones	El sistema otorga los permisos correspondientes a cada usuario.
Referencias	RF 1
Prioridad	Normal

Flujo Normal de Eventos	
Sección "Autenticar Usuarios"	
Acción del Actor	**Respuesta del Sistema**
1. Accede a la Interfaz del Sistema.	2. Muestra los campos a llenar por el usuario. Usuario: Contraseña:
3. Introduce los datos y selecciona el botón Ingresar.	4. El sistema verifica que los datos son correctos.
	5. El sistema verifica que el usuario no está registrado en la BD.
	6. Se muestra una interfaz de inicio correspondiente a los privilegios del usuario.
Prototipo de Interfaz	
Flujo Alterno #1	

4.1 "Datos Incorrectos Autenticar Usuario"	
Acción del Actor	**Respuesta del Sistema**
	4.1. El sistema verifica que los datos son incorrectos y muestra un mensaje de error.

Caso de Uso:	Subir Proyecto
Actores:	Usuario (inicia)
Resumen:	El CU se inicia cuando el usuario selecciona el archivo a subir, en caso que ocurra con éxito muestra un mensaje informando que el archivo fue subido correctamente, en caso contrario también muestra un mensaje notificando que se produjo un error.
Precondiciones:	El usuario debe estar autenticado y en la sesión de *Python*.
Poscondiciones	El archivo fue subido con éxito.
Referencias	RF 1 y RF 2
Prioridad	Crítica

Flujo Normal de Eventos	
Sección "Subir Proyecto"	
Acción del Actor	**Respuesta del Sistema**
1. El usuario selecciona la opción "Subir Fichero".	2. El sistema muestra una interfaz con la opción para subir archivos.

3. El usuario selecciona el botón “Subir Fichero”.	4. El sistema muestra al usuario una ventana para buscar el fichero a subir.
5. El usuario selecciona el fichero y presiona la opción “Abrir”.	6. El sistema verifica que el formato subido es válido y muestra un mensaje de satisfacción.
Prototipo de Interfaz	
Flujo Alterno #1	
5.1 “Cancelar el archivo a subir”	
Acción del Actor	**Respuesta del Sistema**
5.1. El usuario selecciona la opción “Cancelar”.	5.2. El sistema cierra la ventana para buscar el fichero a subir.
Flujo Alterno #2	
6.1 “Formato incorrecto”	
Acción del Actor	**Respuesta del Sistema**
	6.1 El sistema verifica que el formato del fichero es incorrecto y muestra un mensaje de error notificándolo.

Caso de Uso:	Subir Código
Actores:	Usuario (inicia)
Resumen:	El CU comienza cuando el usuario escribe un fragmento de código; posteriormente este pasa a salvarlo en el servidor.
Precondiciones:	El usuario debe estar autenticado y en la sesión de *Python*.
Poscondiciones	El código se guarda en el servidor.
Referencias	RF 1, RF 3
Prioridad	Crítica

Flujo Normal de Eventos	
Sección "Escribir Código"	
Acción del Actor	**Respuesta del Sistema**
1. El usuario seleciona la opción "Subir Código".	2. Se muestra un área de texto donde el usuario podrá introducir el código de su proyecto, así como personalizar su aspecto visual.
3. El usuario introduce el código y selecciona la opción "guardar" que se encuentra en la parte superior del área de texto.	4. El sistema guarda el código en el servidor y muestra un mensaje notificando que el código ha sido salvado satisfactoriamente.
Prototipo de Interfaz	

Caso de Uso:	Revisar Proyecto
Actores:	Usuario (inicia)
Resumen:	El CU se inicia cuando el usuario desea revisar un proyecto que ha subido anteriormente. A continuación, selecciona la opción “analizar” y se muestra un mensaje una vez que ha culminado dicha acción.
Precondiciones:	El usuario debe estar autenticado y en la sesión de *Python*
Poscondiciones	Se genera un reporte.
Referencias	RF 1, RF 2, RF 3, RF 4
Prioridad	Crítica

Flujo Normal de Eventos	
Sección “Revisar Proyecto”	
Acción del Actor	**Respuesta del Sistema**
1. El Usuario selecciona la opción “Analizar”.	2. El sistema muestra una interfaz para realizar el análisis de los archivos subidos.
3. El usuario selecciona el botón “Analizar”	4. El sistema muestra un mensaje de confirmación del análisis.
5. El usuario selecciona la opción “Aceptar”.	6. El sistema muestra un mensaje notificando el éxito del análisis.
Prototipo de Interfaz	

Flujo Alterno #1	
5.1 "Cancelar análisis a realizar"	
Acción del Actor	**Respuesta del Sistema**
5.1. En usuario selecciona la opción Cancelar.	5.2. El sistema muestra un mensaje de cancelación.
Flujo Alterno #2	
6.1 "Cancelar análisis en curso"	
Acción del Actor	**Respuesta del Sistema**
6.1. En usuario selecciona la opción Cancelar.	6.2. El sistema muestra un mensaje informando la cancelación del análisis.

Caso de Uso:	Mostrar Reportes
Actores:	Usuario (inicia)
Resumen:	El CU comienza cuando se selecciona la opción "Mostrar Reportes" y al seleccionar un reporte, el sistema le muestra el mismo. En caso de querer eliminar algún reporte el usuario lo selecciona y presiona la opción eliminar.
Precondiciones:	El usuario debe estar autenticado y en la sesión de *Python*.

Poscondiciones	El sistema muestra un reporte.
Referencias	RF 1, RF 5 y RF 6
Prioridad	Crítica

Flujo Normal de Eventos	
Sección "Mostrar Reportes"	
Acción del Actor	**Respuesta del Sistema**
1. El usuario seleciona la opción "Reportes".	2. Se muestran todos los reportes que hay guardados en la base de datos.
3. El usuario selecciona el reporte que desea ver.	4. El sistema muestra el reporte seleccionado.
Sección "Eliminar Reportes"	
Acción del Actor	**Respuesta del Sistema**
5. El usuario selecciona el botón eliminar.	6. El sistema elimina el reporte y actualiza la lista de reportes.
Prototipo de Interfaz	
Flujo Alterno #1	
2.1 "No existen reportes"	
Acción del Actor	**Respuesta del Sistema**

.	2.1 El sistema muestra un mensaje notificando que no existen reportes en la base de datos.

Caso de Uso:	Mostrar Estadísticas
Actores:	Usuario (inicia)
Resumen:	El CU comienza cuando un usuario desea ver las estadísticas sobre los reportes generados.
Precondiciones:	El usuario debe estar autenticado y en la sesión de *Python*.
Poscondiciones	Las estadísticas se muestran exitosamente
Referencias	RF 1 y RF 7
Prioridad	Normal

Flujo Normal de Eventos	
Sección "Mostrar Estadísticas"	
Acción del Actor	**Respuesta del Sistema**
1.El usuario selecciona la opción "Estadísticas".	2. Se muestra una interfaz donde aparecen las opciones de las estadísticas a mostrar.
3. El usuario selecciona la estadística deseada.	4. El sistema muestra las estadísticas solicitadas.
Prototipo de Interfaz	

Flujos Alternos	

2.5. Conclusiones

En el presente capítulo se inició el desarrollo de la propuesta de solución al problema planteado, donde a partir del análisis de los procesos del negocio, el cual fue plasmado en las descripciones detalladas del mismo, en conjunto con las acciones que realizan el actor y los trabajadores, se obtuvo el listado de requerimientos necesarios para realizar el sistema y los casos de uso de este, sentando así las bases para continuar con el análisis y diseño del proceso.

Capítulo 3
ANÁLISIS Y DISEÑO DEL SISTEMA

3.1. Introducción

En el presente capítulo se expone el análisis y diseño propuesto para la solución del sistema. Se especifican las clases de análisis de cada caso de uso, las relaciones entre ellas y las entidades. Luego del análisis del sistema se desarrolla el modelo de diseño, en el cual quedarán esclarecidas las clases del mismo, así como los patrones que se utilizaron. Se modela el Diagrama Entidad-Relación que da paso al Modelo de Datos.

3.2. Modelo de Análisis

El Modelo de Análisis contiene las clases y sus objetos organizados en paquetes que colaboran. Durante el análisis, se analizan los requisitos que se describieron en la captura de requisitos, refinándolos y estructurándolos. El objetivo es desarrollar una serie de modelos que describan el *software* al trabajar, para satisfacer un conjunto de requisitos definidos por el cliente. El análisis debe lograr tres objetivos primarios: describir lo que quiere el cliente, establecer una base para la creación de un diseño de *software* y definir un conjunto de requisitos que se puedan validar una vez que se construya el *software*. La realización del análisis proporciona una visión general del sistema, que puede ser más difícil de obtener mediante el estudio de los resultados del diseño y la implementación, por contener demasiados detalles.

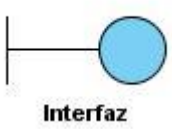
3.2.1. Diagrama de Clases del Análisis

Clases del Análisis: Una clase de análisis representa la abstracción de una o varias clases y/o subsistemas del diseño del sistema.

Estas, están asociadas con el contexto del dominio del problema por lo que representan conceptos y relaciones. Poseen un conjunto de características tales como: se centra en el tratamiento de los requisitos

funcionales y pospone los no funcionales, su comportamiento se define mediante responsabilidades a un nivel más alto y menos formal que las operaciones, definen atributos y participan en relaciones, entre otras.

Las clases del análisis se clasifican en tres tipos:

Nombre	Descripción	Representación
Interfaz	Modelan la interacción entre el sistema y sus actores.	
Control	Coordinan la realización de uno o unos pocos casos de uso coordinando las actividades de los objetos que implementan la funcionalidad del caso de uso.	
Entidad	Modelan información que posee larga vida y que es a menudo persistente.	

A continuación, más características de estas clases:

- **Clases de interfaz**: se utilizan para modelar la interacción entre el sistema y sus actores (usuarios y sistemas externos), la cual implica recibir y presentar información y peticiones, desde y hacia los usuarios y los sistemas externos. Las clases interfaz describen lo que se obtiene con la interacción, es decir, la información y las peticiones que se intercambian, no es necesario que describan cómo se ejecuta físicamente la interacción, ya que esto se considerará en las actividades de diseño e implementación subsiguientes.

- **Clases de control**: Los aspectos dinámicos del sistema se modelan con clases de control, debido a que ellas manejan y coordinan las acciones y los flujos de control principales y delegan trabajo a otros objetos (objetos de interfaz y de entidad). Representan coordinación, secuencia, transacciones, y control de otros objetos y se usan con frecuencia para encapsular el control de un caso de uso en concreto. Las clases de control también se utilizan para representar derivaciones y cálculos complejos.

- **Clases de entidad**: modelan la información y el comportamiento asociado de algún fenómeno o concepto, como una persona, un objeto, o un suceso del mundo real. Un objeto de entidad no ha de ser necesariamente pasivo y puede tener en ocasiones un comportamiento complejo relativo a la información que representa. Los objetos de entidad aíslan los cambios en la información que representan. Las clases de entidad suelen mostrar una estructura de datos lógica y contribuyen a comprender de qué información depende el sistema.

Diagrama de Clases del Análisis: El diagrama de clases del análisis es un artefacto en el que se representa los conceptos fundamentales en un dominio del problema. Los diagramas de clases del análisis, representan las definiciones y relaciones entre las clases.

A continuación, se muestran los diagramas de clases del análisis de los Casos de Uso que presenta el sistema:

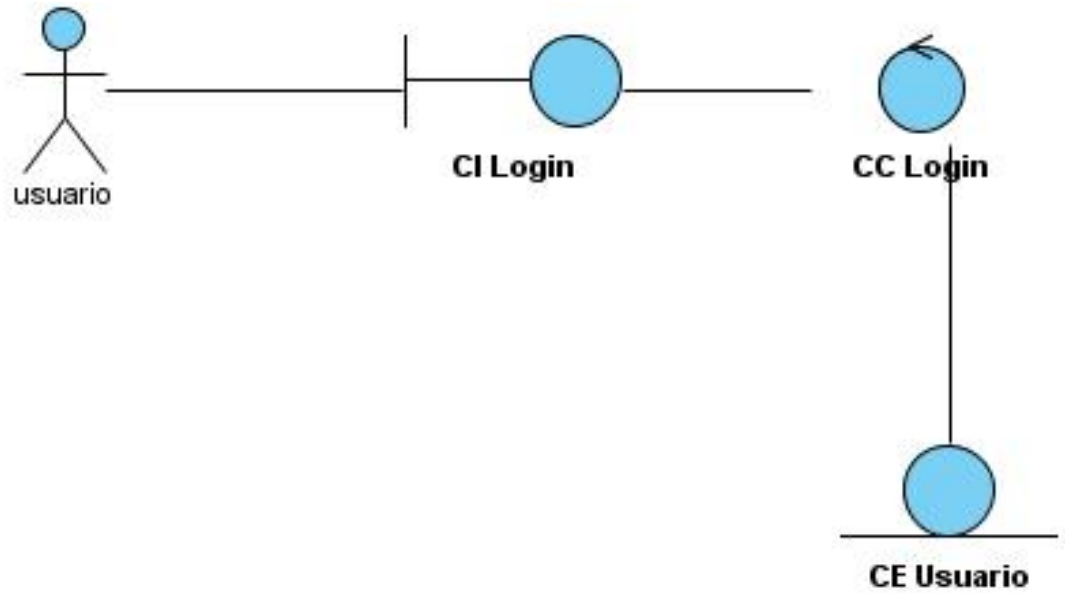


Fig. 3.1 DCA Autenticar Usuario

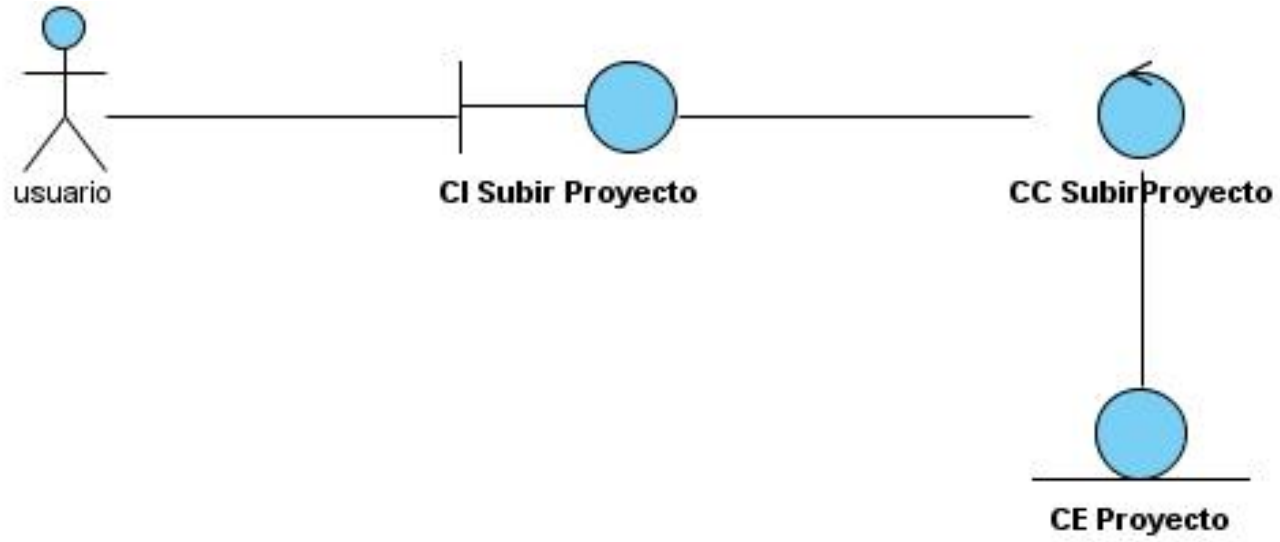


Fig. 3.2 DCA Subir Proyecto

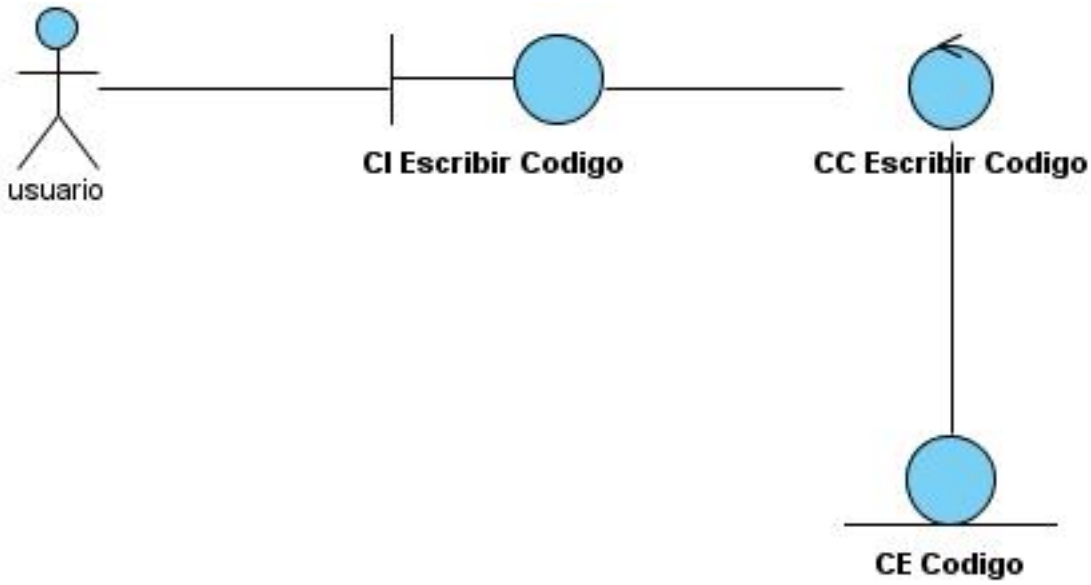


Fig. 3.3 DCA Escribir Código

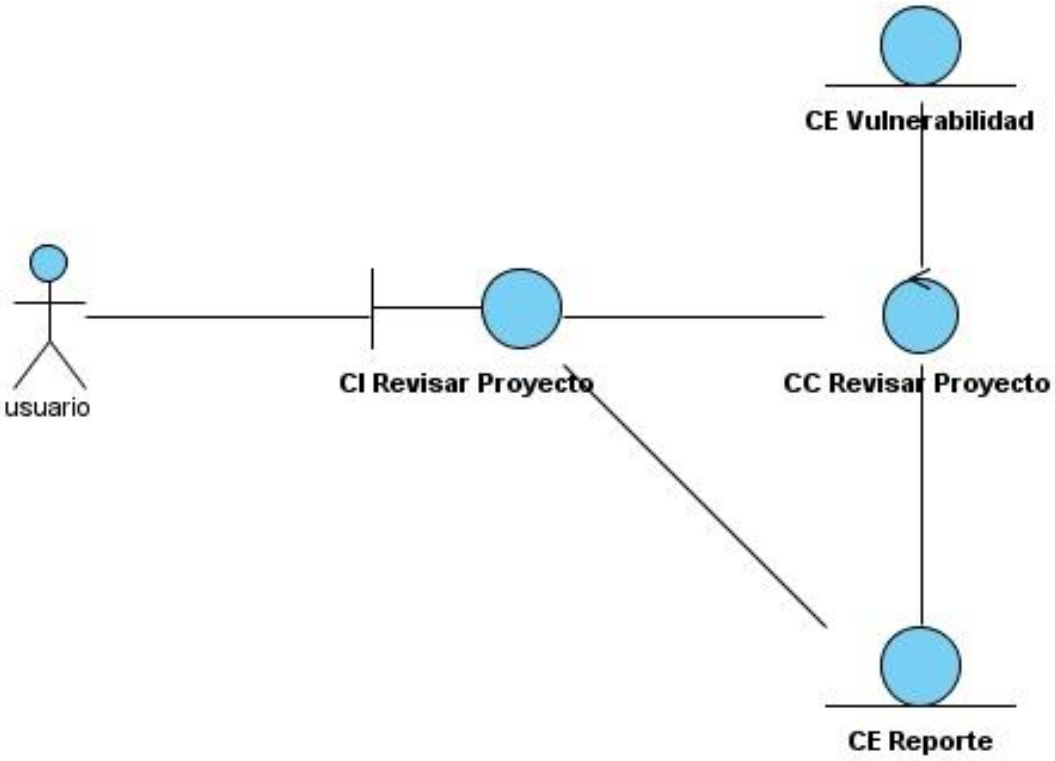


Fig. 3.4 DCA Revisar Proyecto

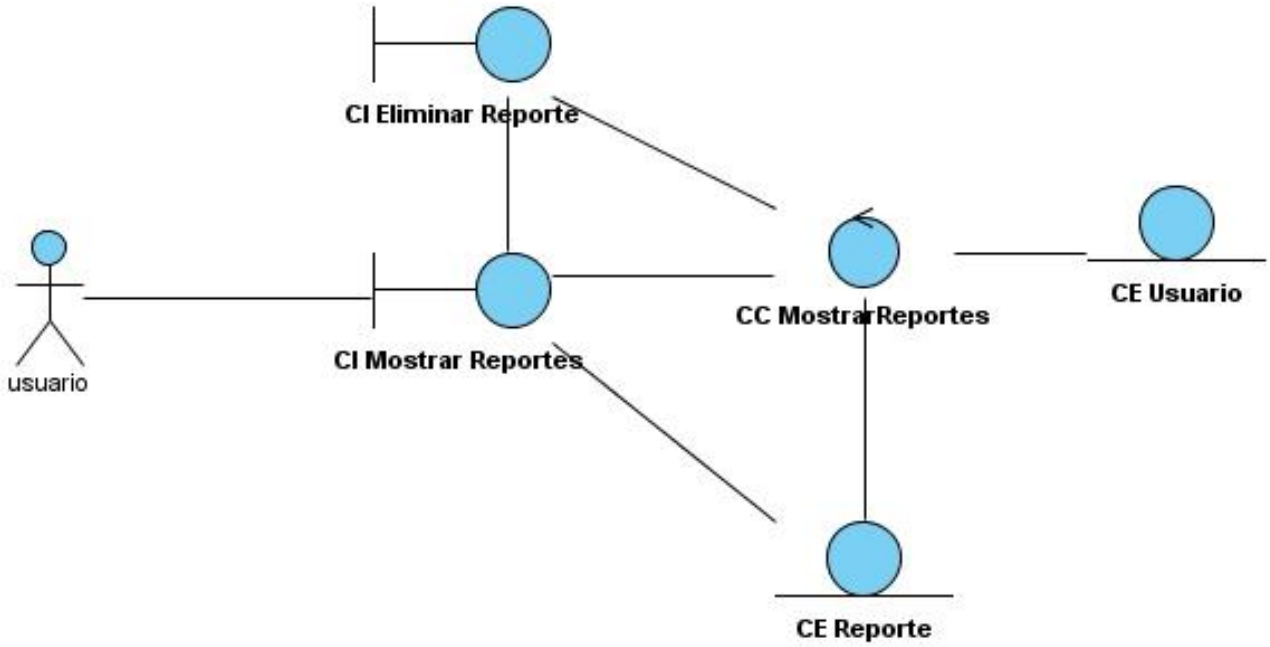


Fig. 3.5 DCA Mostrar Reportes

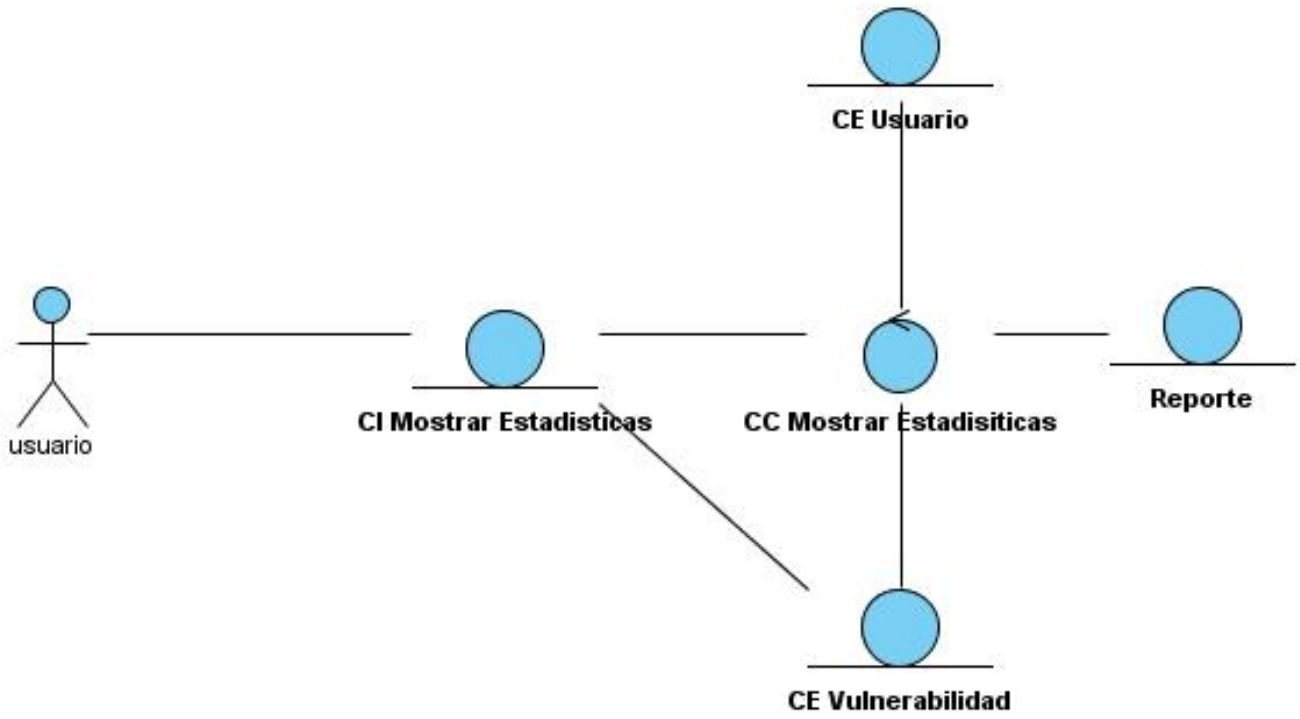


Fig. 3.6 DCA Mostrar Estadísticas

3.3. Modelo de Diseño

El diseño del *software* se encuentra en el núcleo técnico de la ingeniería del *software*. Es la etapa del proceso de desarrollo donde se decide cómo se llevará a cabo el sistema. Una vez que se analizan y especifican los requisitos del *software*, el diseño es la primera de las tres actividades técnicas (diseño, generación del código y pruebas) que se requieren para construir y verificar el *software*.

El diseño se utiliza para modelar el sistema y encontrar su forma para que soporte todos los requisitos, incluyendo los requisitos no funcionales y demás restricciones. Se crea una entrada apropiada y un punto de partida para actividades de implementación subsiguientes capturando los requisitos o subsistemas individuales, interfaces y clases. El diseño debe ser una guía legible y comprensible para aquellos que generan código y para aquellos que comprueban y consecuentemente, dan soporte al *software*; deberá proporcionar una imagen completa del *software*, enfrentándose a los dominios de comportamiento, funcionales y de datos desde una perspectiva de implementación.

3.3.1. Diagrama de Clases del Diseño

Clases del Diseño: Una clase del diseño es una abstracción de una clase o construcción en la implementación del sistema. El lenguaje utilizado para especificar una clase del diseño es lo mismo que el lenguaje de programación. Consecuentemente, las operaciones, parámetros, atributos, tipos y demás son especificados utilizando la sintaxis del lenguaje de programación elegido. Al construir los diagramas de interacción se van utilizando clases procedentes del modelo conceptual, junto con otras creadas para encargarse de responsabilidades específicas. El conjunto de todas las clases usadas, junto con sus relaciones, forma el Diagrama de Clases del Diseño.

Diagrama de Clases del Diseño: Los diagramas de clases de diseño exponen un conjunto de interfaces, colaboraciones y sus relaciones. Se utilizan para modelar la vista de diseño estática de un sistema. Al igual que los demás diagramas, los diagramas de clases pueden contener notas y restricciones. Los diagramas de clases también pueden contener paquetes o subsistemas, los cuales se usan para agrupar los elementos de un modelo en partes más grandes.

En la siguiente tabla se representan los estereotipos más utilizados para las aplicaciones *Web*:

Nombre	Descripción	Representación
Server Page	Representa la página *Web* que tiene código que se ejecuta en el servidor. Este código interactúa con recursos en el servidor. Las operaciones representan las funciones del código y los atributos, las variables visibles dentro del alcance de la página.	 Server Page
Client Page	Una instancia de Página Cliente es una página *Web*, con formato *HTML*. Mezcla de datos, presentación y lógica. Son interpretadas por el navegador. Sus atributos son las variables declaradas dentro del *script* que son accesibles para páginas.	 Client Page
Form	Colección de elementos de entrada que son parte de una página cliente. Se relaciona directamente con la etiqueta de igual nombre del *HTML*. Sus atributos son los elementos de entrada del formulario (*input boxes*, *text areas*, *radio buttons*, *check boxes* y *hidden fields*). No tienen operaciones.	 Form

Para la realización de los diagramas de clases del diseño se tuvo en cuenta la arquitectura que presenta el *framework* a utilizar (*KumbiaPHP*). Basados en la forma en que este *framework* implementa el patrón arquitectónico MVC (ver epígrafe 3.5.2), fueron representados los diagramas de clases del diseño de la siguiente forma:

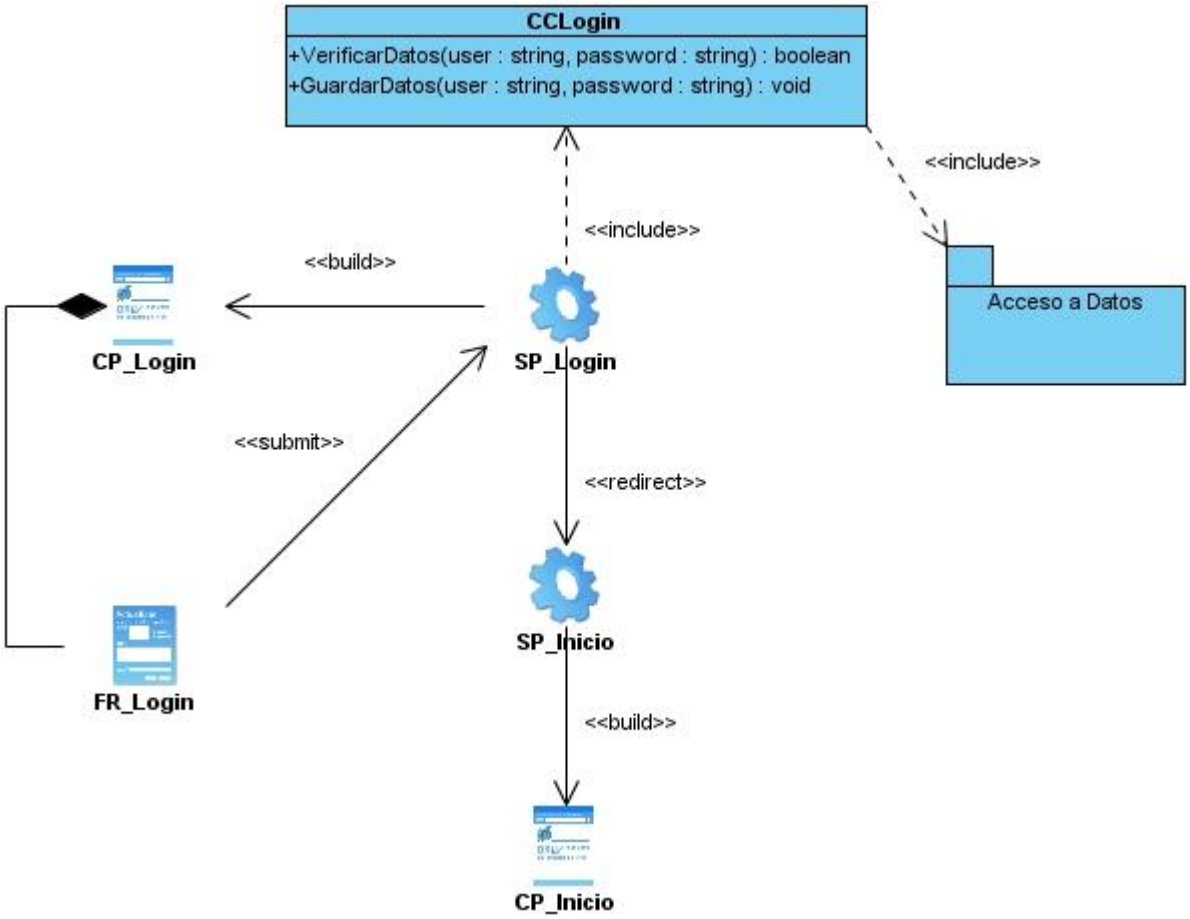


Fig. 3.7 DCD Autenticar Usuario

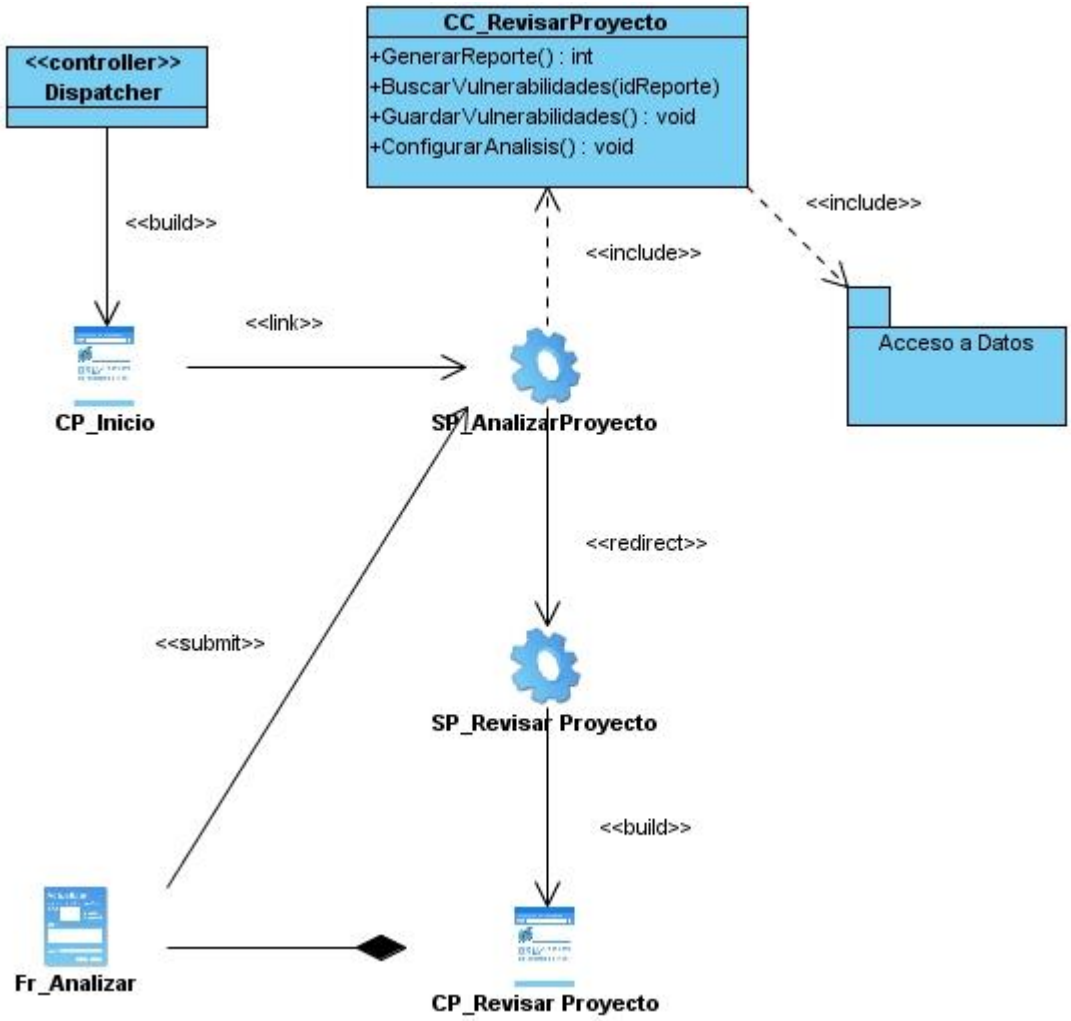


Fig. 3.8 DCD Subir Proyecto

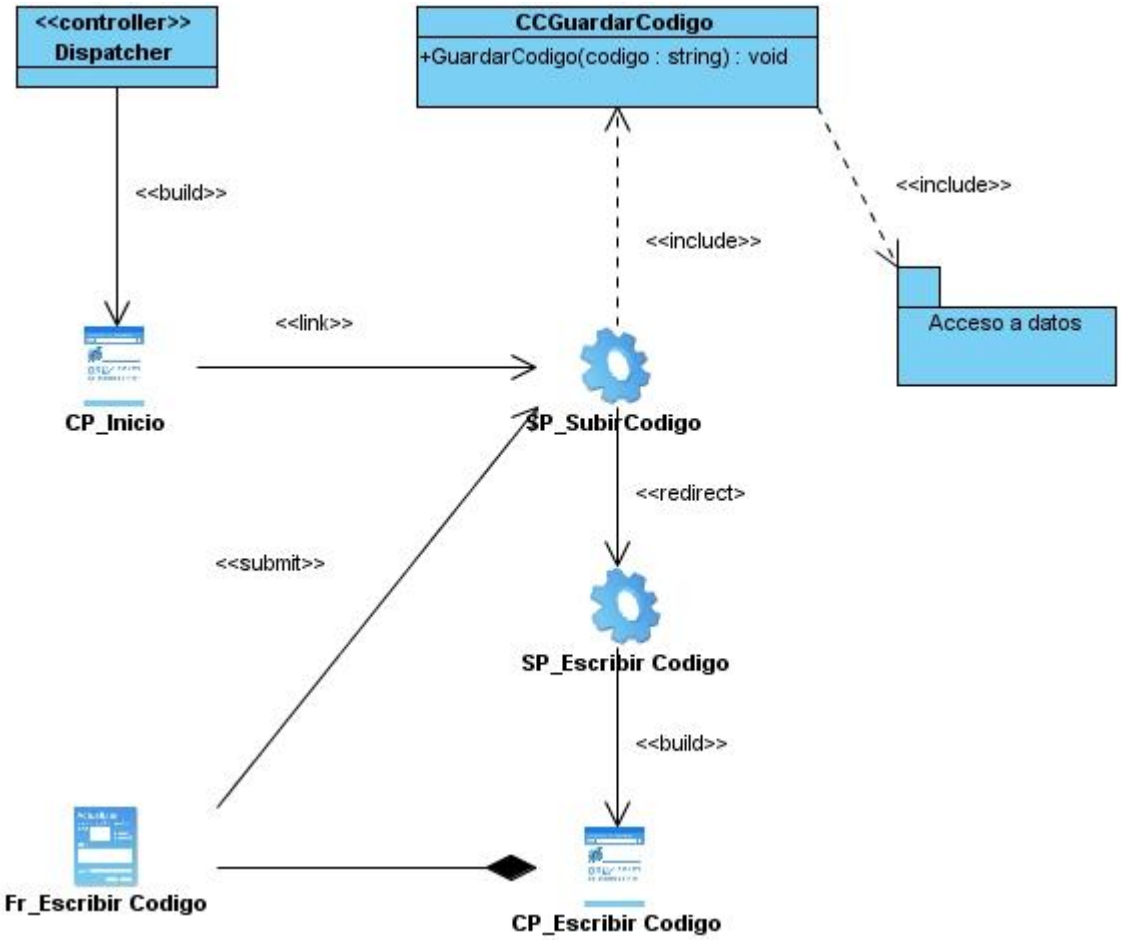


Fig. 3.9 DCD Escribir Código

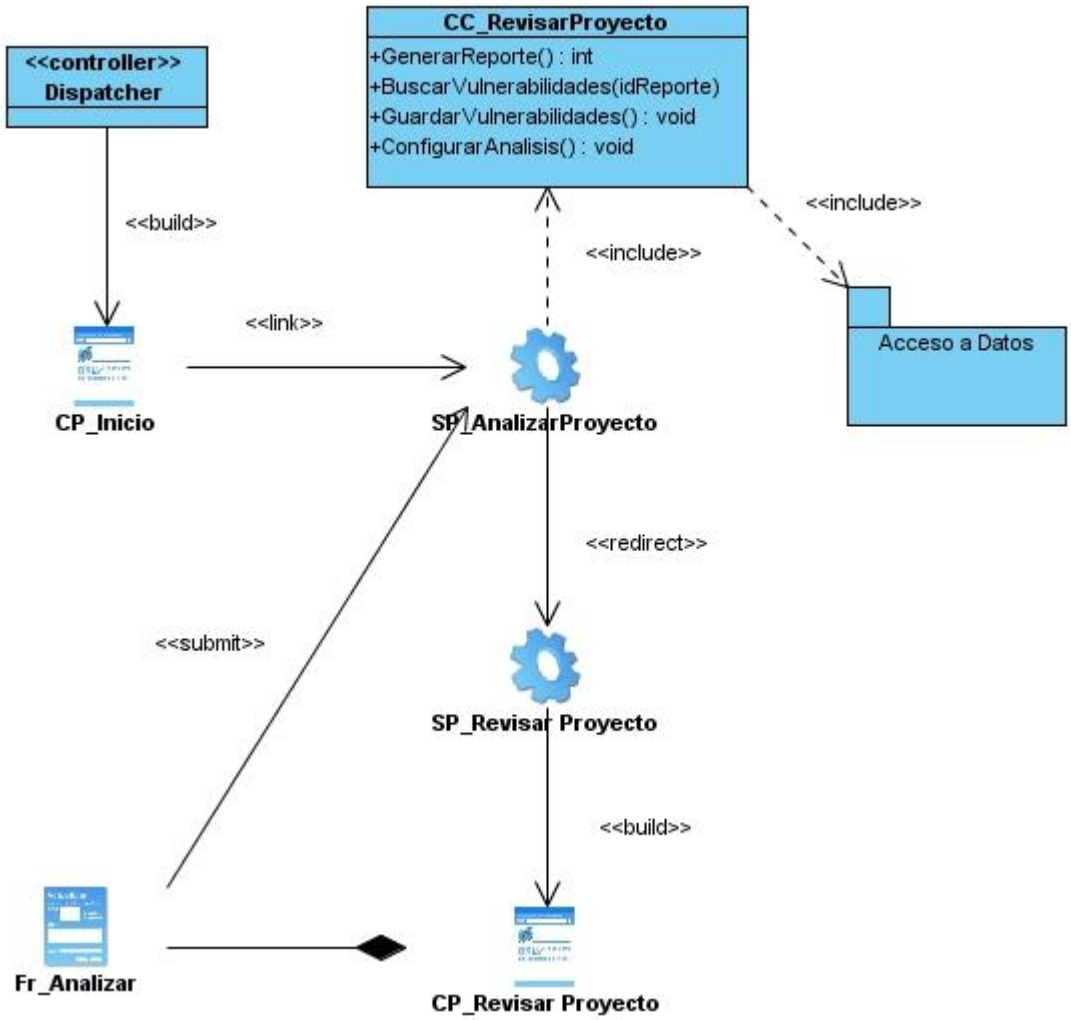


Fig. 3.10 DCD Revisar Proyecto

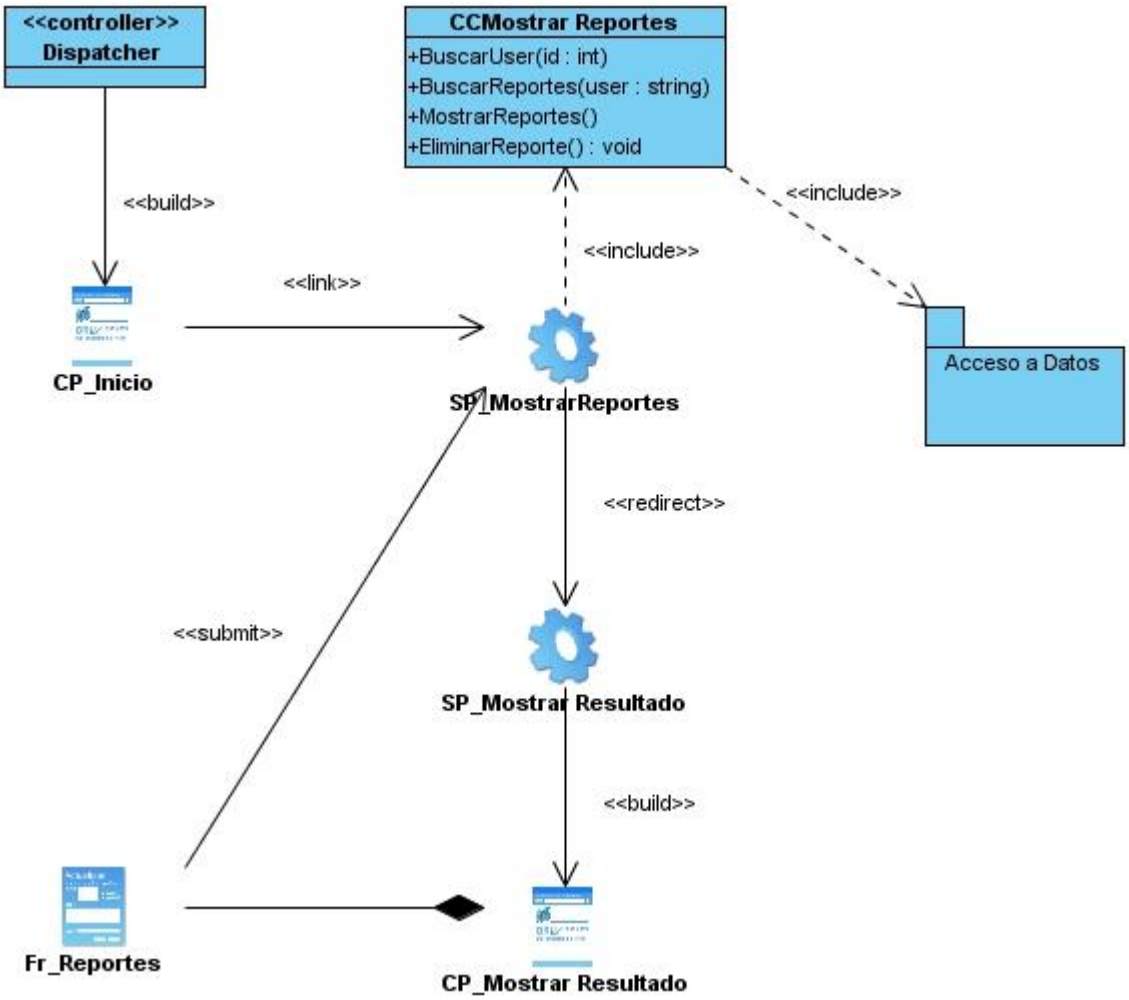


Fig. 3.11 DCD Mostrar Reportes

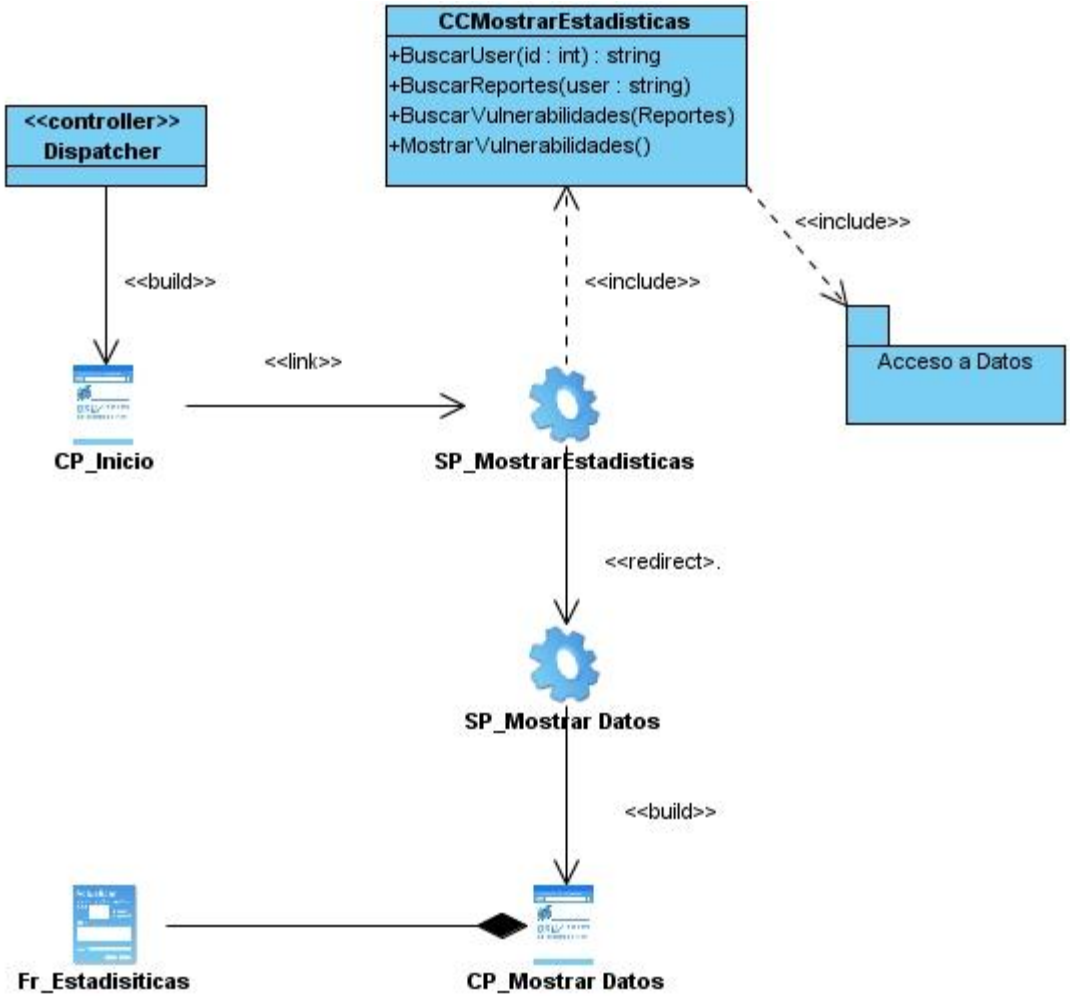


Fig. 3.12 DCD Mostrar Estadísticas

3.3.2. Diagramas de Interacción

La vista de interacción describe secuencias de intercambios de mensajes entre los roles que implementan el comportamiento de un sistema. Un rol es la descripción de un objeto, que desempeña un determinado papel dentro de una interacción, distinto de los otros objetos de la misma clase. Esta visión proporciona una vista integral del comportamiento del sistema, es decir, muestra el flujo de control a través de muchos objetos. La vista de interacción se exhibe en dos diagramas centrados en distintos aspectos pero complementarios: centrados en los objetos individuales y centrados en objetos cooperantes.

Los diagramas de interacción muestran cómo se comunican los objetos en una interacción. En *UML* los diagramas de interacción pueden representarse a través de los Diagramas de Colaboración y/o de los Diagramas de Secuencia; ambos son representaciones alternas de interacciones. Los Diagramas de Secuencia muestran interacciones entre objetos basadas en el tiempo y los Diagramas de Colaboración muestran como los objetos se asocian unos con otros.

3.3.3. Diagrama de Secuencia

Los diagramas de secuencia son más adecuados para observar la perspectiva cronológica de las interacciones, muestran la secuencia explícita de mensajes y son mejores para especificaciones de tiempo real y para escenarios complejos. A diferencia de los diagramas de colaboración, los diagramas de secuencia incluyen secuencias temporales pero no incluyen las relaciones entre objetos. Pueden existir de forma de descriptor, describiendo todos los posibles escenarios, y en forma de instancia, describiendo un escenario real.

En el **Anexo 3** se muestran los diagramas de secuencia para cada caso de uso.

3.4. Modelo de Datos

Una base de datos es un conjunto estructurado de datos interrelacionados y sin redundancias, registrados o almacenados sobre soportes accesibles por ordenador para satisfacer simultáneamente a varios

usuarios en tiempo oportuno, y que pueden procesarse por uno o más sistemas de aplicación. El artefacto resultante de la actividad Diseñar la Base de Datos es el Modelo de Datos. Un Modelo de Datos está constituido por herramientas conceptuales para describir los datos, las relaciones que existen entre ellos, semántica asociada a los datos y las restricciones de consistencia.

Además, un Modelo de Datos es aquel que describe de forma abstracta cómo se representan los datos. Básicamente, consiste en una descripción de algo conocido como contenedor de datos, así como los métodos para almacenar y recuperar información de estos contenedores. El modelo de datos tiene gran importancia en el proceso de desarrollo de *software*, debido a que define formalmente las estructuras permitidas y las restricciones a fin de representar los datos y constituye un elemento básico para el desarrollo de la metodología del diseño de la base de datos. Este modelo está formado por objetos (entidades que existen y que se manipulan), atributos (características básicas de estos objetos) y relaciones (forma en que enlazan los distintos objetos entre sí).

3.4.1. Diagrama de Clases Persistentes

Se usan para describir datos en los niveles conceptuales y de visión, es decir, con este modelo se representan los datos de tal forma como se captan en el mundo real, tienen una capacidad de estructuración bastante flexible y permiten especificar restricciones de datos explícitamente. Existen diferentes modelos de este tipo, pero el más utilizado por su sencillez y eficiencia es el modelo Entidad-Relación.

A continuación, se representa el diagrama de clases persistentes de la base datos del sistema:

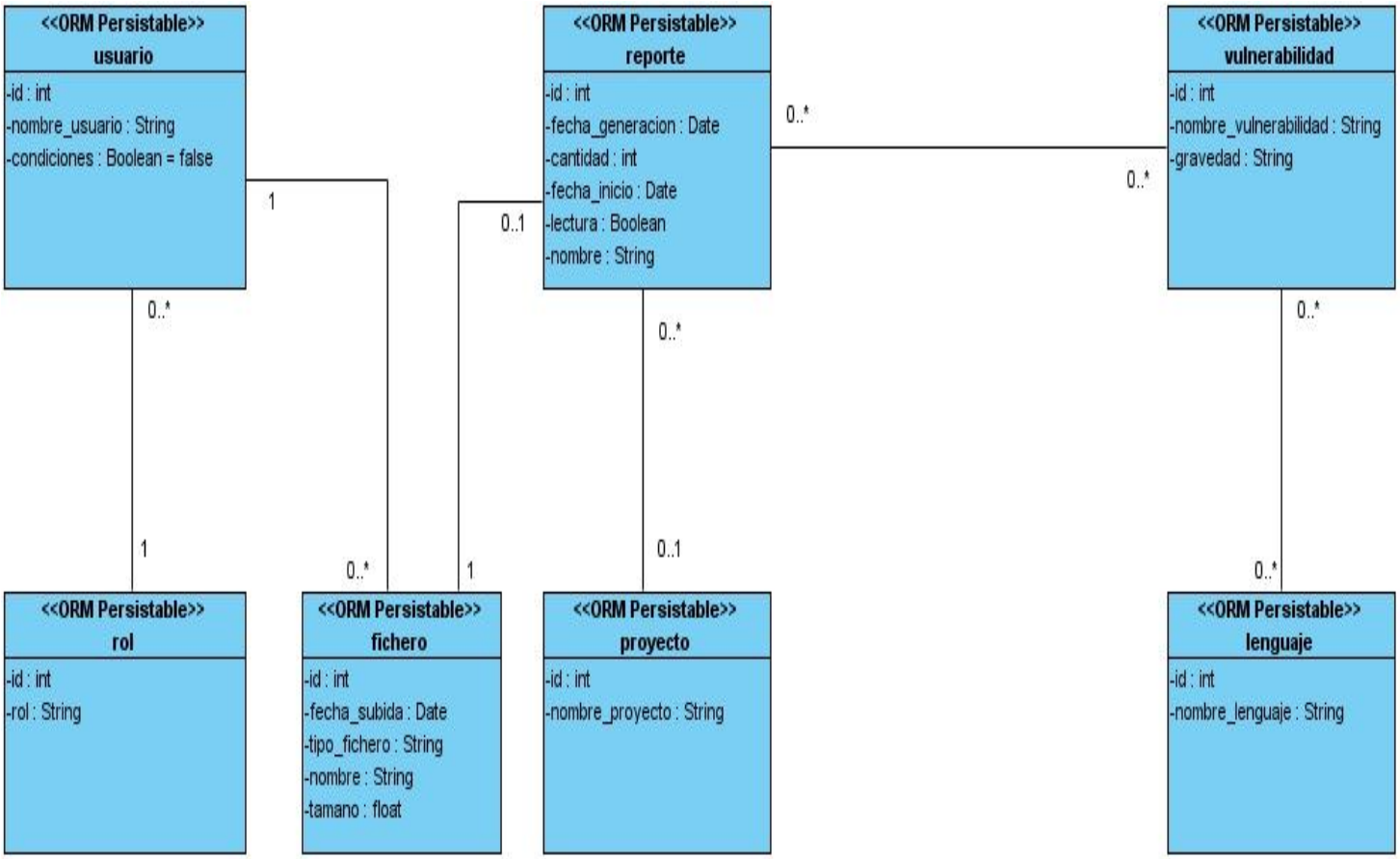


Fig. 3.13 Diagrama de Clases Persistentes.

3.4.2. Modelo Físico de Datos

Se usan para describir a los datos en el nivel más bajo, aunque existen muy pocos modelos de este tipo, básicamente capturan aspectos de la implementación de los sistemas de base de datos.

A continuación, se representa el modelo físico de la base datos del sistema:

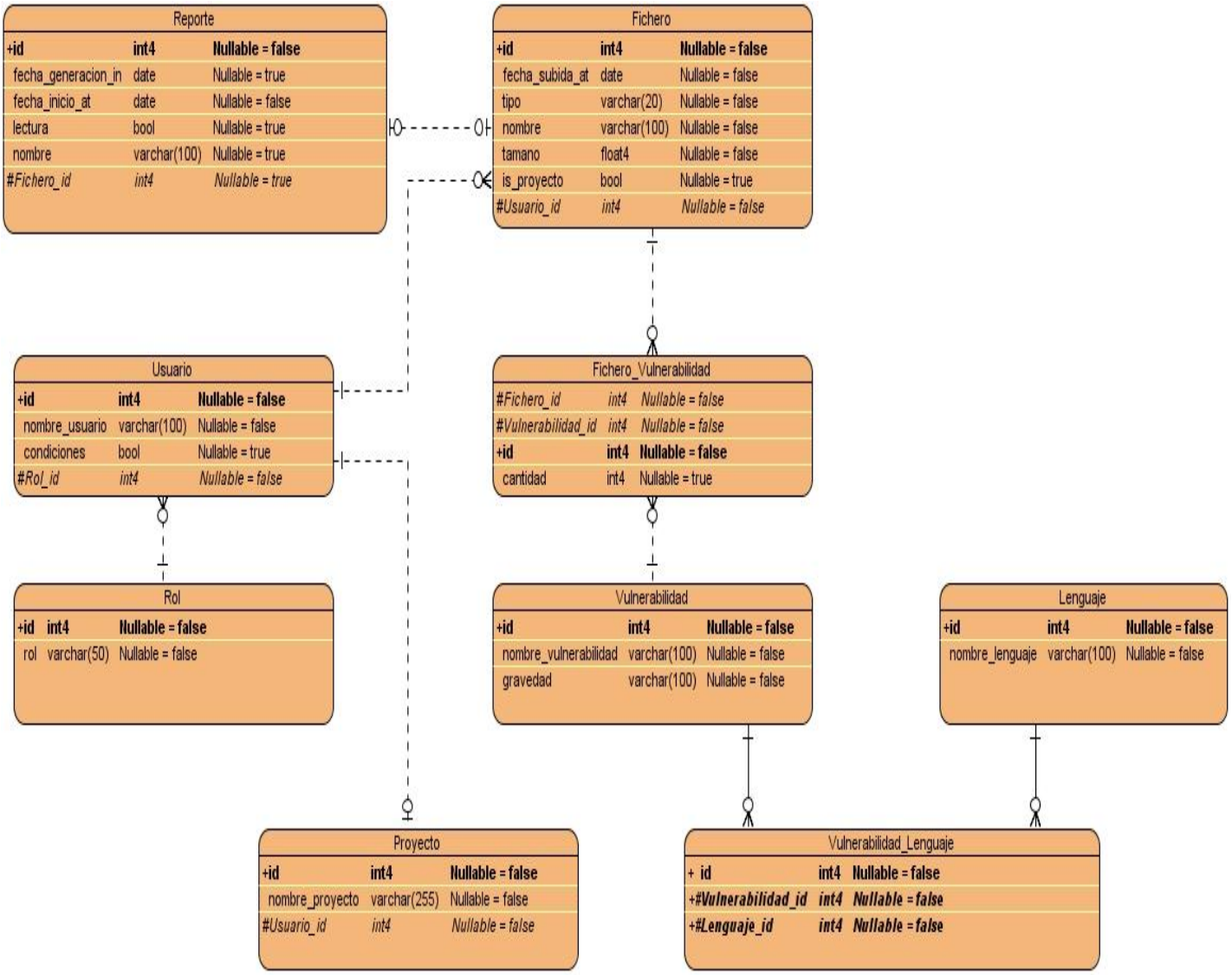


Fig. 3.14 Modelo Físico de Datos

3.5. Patrones Utilizados

¿Qué es un Patrón?

Un patrón es un modelo a seguir para realizar algo. Es además, una solución a un problema en un contexto y codifica conocimiento específico acumulado por la experiencia en un dominio; pareja de problema / solución con un nombre, que estandariza buenos principios y sugerencias relacionados frecuentemente con la asignación de responsabilidades. Los patrones surgen de la experiencia de seres humanos al tratar de lograr ciertos objetivos y capturan la experiencia existente y probada para promover buenas prácticas.

Categorías de patrones

Según la escala o nivel de abstracción:

- **Patrones arquitecturales**: Aquellos que expresan un esquema organizativo estructural fundamental para sistemas *software*.
- **Patrones de diseño**: Aquellos que expresan esquemas para definir estructuras de diseño (o sus relaciones) con las que construir sistemas *software*.
- **Idiomas**: Patrones de bajo nivel específicos para un lenguaje de programación o entorno concreto.

3.5.1. Patrones de Diseño

Un patrón de diseño es una descripción de clases y objetos comunicándose entre sí adaptada para resolver un problema de diseño general en un contexto particular.

Un patrón de diseño identifica: Clases, Instancias, Roles, Colaboraciones y la distribución de responsabilidades.

Ventajas

- Los patrones de diseño proponen una forma de reutilizar la experiencia de los desarrolladores, para ello clasifican y describen formas de solucionar problemas que ocurren de forma frecuente en el desarrollo.
- Están basados en la recopilación del conocimiento de los expertos en desarrollo de *software*.
- Es una experiencia real, probada y que funciona. Es Historia y nos ayuda a no cometer los mismos errores.

Patrones *GOF*

Los patrones *GOF* se clasifican en tres tipos**:**

- **De Creación:** abstraen el proceso de creación de instancias.
- **Estructurales:** se ocupan de cómo clases y objetos son utilizados para componer estructuras de mayor tamaño.
- **De Comportamiento:** atañen a los algoritmos y a la asignación de responsabilidades entre objetos.

Entre los patrones *GOF* que implementa *KumbiaPHP*, se pueden encontrar:

Patrones Creacionales

***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.

***Singleton* (Instancia única):** Este patrón asegura que una clase tendrá solo una instancia, y la creación de un mecanismo de acceso global a dicha instancia. El constructor es privado y el método instance () es el que devuelve la única instancia de esta clase o la crea si no existe.

Existen otros patrones implementados por *KumbiaPHP,* los cuales son:

Active Record: proporciona la capa objeto-relacional que sigue rigurosamente el estándar ORM: Tablas en Clases, Campos en Atributos y Registros en Objetos. Facilita el entendimiento del código asociado a base de datos y encapsula la lógica específica, haciéndola más fácil de usar para el programador. (17)

Template View: permite utilizar un sistema de plantillas y vistas que son reutilizables para no repetir código y darle más poder a nuestra presentación. (18)

Patrones *GRASP*

GRASP es un acrónimo que significa *General Responsibility Asignment Software Patterns* (patrones generales de software para asignar responsabilidades). El nombre se eligió para indicar la importancia de captar (*grasping*) estos principios, si se quiere diseñar eficazmente el *software* orientado a objetos.

En el proceso de diseño del sistema se utilizaron los patrones básicos *GRASP*, estos son:

- Experto
- Creador
- Alta Cohesión
- Bajo Acoplamiento
- Controlador

Experto: Experto es un patrón que se usa más que cualquier otro al asignar responsabilidades; es un principio básico que suele utilizarse en el diseño orientado a objetos. El patrón Experto (como tantas otras cosas en la tecnología de objeto) ofrece una analogía con el mundo real. Este se utiliza para asignar una responsabilidad al experto en información: la clase que cuenta con la información necesaria para cumplir la responsabilidad.

Creador: El patrón Creador guía la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos. El propósito fundamental de este

patrón es encontrar un creador que debemos conectar con el objeto producido en cualquier evento. Al escogerlo como creador, se da soporte al bajo acoplamiento.

Alta Cohesión: En la perspectiva del diseño orientado a objetos, la cohesión (o, más exactamente, la cohesión funcional) es una medida de cuán relacionadas y enfocadas están las responsabilidades de una clase. Una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme. Una clase con baja cohesión hace muchas cosas no afines o un trabajo excesivo. Por tanto, este patrón se utiliza para asignar una responsabilidad de modo que la cohesión siga siendo alta.

Bajo Acoplamiento: El acoplamiento es una medida de la fuerza con que una clase está conectada a otras clases, con que las conoce y con que recurre a ellas. Una clase con bajo (o débil) acoplamiento no depende de muchas otras. Una clase con alto (o fuerte) acoplamiento recurre a muchas otras. Este patrón se utiliza para asignar una responsabilidad para mantener bajo acoplamiento. El Bajo Acoplamiento estimula asignar una responsabilidad de modo que su colocación no incremente el acoplamiento tanto que produzca los resultados negativos propios de un alto acoplamiento.

Creador: La mayor parte de los sistemas reciben eventos de entrada externa, los cuales generalmente incluyen una interfaz gráfica para el usuario operado por una persona. Otros medios de entrada son los mensajes externos (entre ellos un conmutador de telecomunicaciones para procesar llamadas), o las señales procedentes de sensores como sucede en los sistemas de control de procesos. En todos los casos, si se recurre a un diseño orientado a objetos, hay que elegir los controladores que manejen esos eventos de entrada. Este patrón ofrece una guía para tomar decisiones apropiadas que generalmente se aceptan.

3.5.2. Patrón Arquitectural

Un estilo arquitectónico o variante arquitectónica define a una familia de sistemas informáticos en términos de su organización estructural. Además, describe componentes y las relaciones entre ellos con las restricciones de su aplicación, la composición asociada y el diseño para su construcción. Existen varios

patrones de arquitectura, dentro de los cuales encontramos el Modelo Vista Controlador (MVC). La mayoría de los *framworks web* que existen actualmente en el mundo lo utilizan, y claro, Kumbia no constituye la excepción.

Modelo Vista Controlador (MVC)

El patrón de arquitectura de *software* MVC es un patrón que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos. El patrón MVC se ve frecuentemente en aplicaciones *web*, donde la vista es la página *HTML* y el código que provee de datos dinámicos a la página, el modelo es el Sistema de Gestión de Base de Datos y el controlador representa la lógica de negocio.

Descripción del patrón

Modelo: Esta es la representación específica de la información con la cual el sistema opera. La lógica de datos asegura la integridad de estos y permite derivar nuevos datos. Muchas aplicaciones utilizan un mecanismo de almacenamiento persistente (como puede ser una base de datos) para almacenar los datos.

Vista: Este presenta el modelo en un formato adecuado para interactuar, usualmente la interfaz de usuario. Ejemplo un Formulario.

Controlador: Este responde a eventos, usualmente acciones del usuario e invoca cambios en el modelo y probablemente en la vista.

Entre las ventajas del patrón MVC están las siguientes:

Soporte de múltiples vistas: Dado que la vista se halla separada del modelo y no hay dependencia directa del modelo con respecto a la vista, la interfaz de usuario puede mostrar múltiples vistas de los

mismos datos simultáneamente. Por ejemplo, múltiples páginas de una aplicación *web* pueden utilizar el mismo modelo de objetos mostrado de maneras diferentes.

Adaptación al cambio: Los requerimientos de interfaz de usuario tienden a cambiar con mayor rapidez que las reglas de negocio. Los usuarios pueden preferir distintas opciones de representación, o requerir soporte para nuevos dispositivos como teléfonos celulares. Dado que el modelo no depende de las vistas, agregar nuevas opciones de presentación generalmente no afecta al modelo.

¿Cómo funciona este patrón en *Kumbia*?

En *Kumbia* los controladores están separados en partes, llamadas *front controller* y en un conjunto de acciones. Cada acción sabe cómo reaccionar ante un determinado tipo de petición. Las vistas están separadas en *layouts*, *templates* y *partials*. El modelo ofrece una capa de abstracción de la base de datos utilizada, además, dan funcionalidad agregada a datos de sesión y validación de integridad relacional. (19)

Este modelo ayuda a separar el trabajo en lógica de negocio (modelos) y la presentación (Vistas). Por ejemplo, si se tiene una aplicación que corra tanto en equipos de escritorio y en dispositivos de bolsillo entonces podría crear dos vistas diferentes compartiendo las mismas acciones en el controlador y la lógica del modelo.

El controlador ayuda a ocultar los detalles de protocolo utilizados en la petición (*HTTP*, modo consola, etc.) para el modelo y la vista. Finalmente, el modelo abstrae la lógica de datos, que hace a los modelos independientes de las vistas.

La implementación de este modelo es muy liviana, mediante pequeñas convenciones se puede lograr mucho poder y funcionalidad.

Para una mayor comprensión, en el **Anexo 4** se encuentra reflejado este funcionamiento.

3.6. Conclusiones

En el desarrollo de este capítulo se realizó el modelado de la solución propuesta, desarrollando el Modelo de Análisis y el Modelo de Diseño, generándose todos los artefactos necesarios para ello, dígase diagramas de clases del análisis y diseño, diagramas de secuencia, además de los diagramas de Clases persistentes y el Físico de Datos obtenidos de la base de datos. De esta forma quedan creadas las condiciones para comenzar la implementación del sistema.

Capítulo 4
IMPLEMENTACIÓN DEL SISTEMA

4.1. Introducción

Este capítulo tiene como objetivo abordar el flujo de trabajo de implementación donde se describe cómo se implementan en términos de componentes los elementos del modelo de diseño. Se modelan los diagramas de componentes y de despliegue, dando una visión de cómo quedará desarrollada y distribuida la aplicación.

4.2. Modelo de Implementación

El modelo de implementación especifica cómo los elementos del modelo de diseño, fundamentalmente las clases, se implementan en términos de componentes como ficheros de código fuente, ejecutables, librerías, entre otros. También describe cómo se organizan los componentes de acuerdo con los mecanismos de estructuración y modularización disponibles en el entorno de implementación y en el lenguaje o lenguajes de programación utilizados, y cómo dependen los componentes unos de otros, así como su organización de acuerdo a los nodos específicos en el modelo de despliegue.

4.2.1. Diagrama de Despliegue

Un diagrama de despliegue es un grafo de nodos unidos por conexiones de comunicación. Un nodo puede contener instancias de componentes *software*, objetos, procesos (caso particular de un objeto). Las instancias de componentes *software* pueden estar unidas por relaciones de dependencia, posiblemente a interfaces (ya que un componente puede tener más de una interfaz).

Los diagramas de despliegue muestran la disposición física de los distintos nodos que componen un sistema y el reparto de los componentes sobre dichos nodos. La vista de despliegue representa la

disposición de las instancias de componentes de ejecución en instancias de nodos conectados por enlaces de comunicación.

Un nodo es un objeto físico en tiempo de ejecución que representa un recurso computacional, generalmente con memoria y capacidad de procesamiento. Pueden representarse instancias o tipos de nodos en forma de cubos en 3D en los diagramas de implementación.

A continuación, se representa el diagrama de despliegue correspondiente a nuestro sistema:

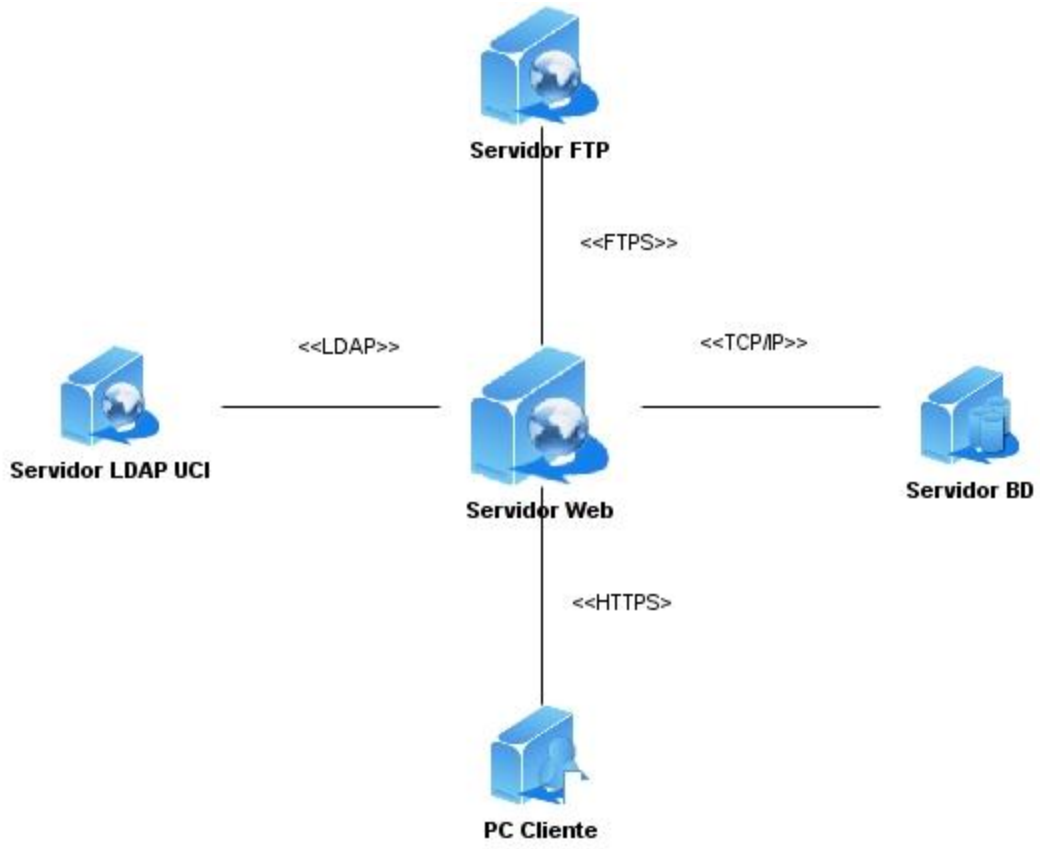


Fig. 4.1 Diagrama de Despliegue

A continuación se explican los recursos presentes en cada nodo del diagrama:

- ***PC* Cliente.** A través de esta *PC*, los clientes se conectarán al sistema y podrán operar el mismo. Esta *PC* deberá contar con el sistema operativo *Windows XP* o *GNU*/*Linux* además del navegador *Mozilla Firefox* que le permitirán al cliente tener acceso a la aplicación.
- **Servidor *Web*.** El mismo deberá contar con el software base *GNU*/*Linux* Distribución *Ubuntu* 9.10. Además, como servidor *Web* Apache *2.2*, *framework* de desarrollo *KumbiaPHP* 1.0 *stable* y el lenguaje de programación *PHP5*.
- **Servidor Base de Datos.** Este servidor deberá utilizar como *software* base *GNU/Linux* Distribución *Ubuntu* 9.10 y como Sistema Gestor de Base de Datos *PostgreSQL* 8.4.
- **Servidor FTP**. Mediante este servidor serán subidos los proyectos de los usuarios. Se utilizara *VSFTP* 2.2.2.
- **Servidor LDAP UCI.** Mediante este servidor se comprobará la validez de los datos de los usuarios a la hora de identificarse en el sistema.

4.2.2. Diagrama de Componentes

Un diagrama de componentes es un grafo de componentes unidos a través de relaciones que pueden ser de compilación o de ejecución. Los diagramas de componentes muestran los componentes de *software* que constituyen una parte reusable, sus interfaces y sus interrelaciones, en muchos aspectos se puede considerar que un diagrama de componentes es un diagrama de clases a gran escala. Estos son usados para estructurar el modelo de implementación en términos de subsistemas de implementación y mostrar las relaciones entre estos elementos. Se utilizan además para modelar la vista estática de un sistema y muestran la organización y las dependencias lógicas entre un conjunto de componentes de *software*, sean éstos componentes de código fuente, librerías, binarios o ejecutables. A continuación, se muestran los diagramas de componentes:

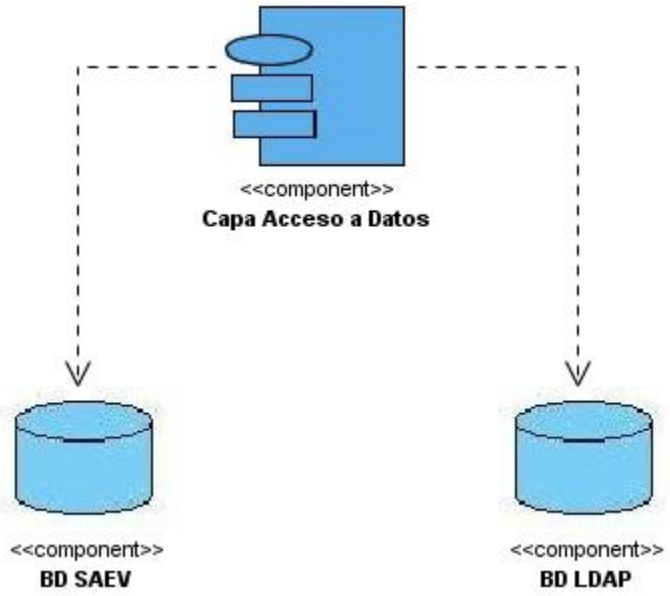


Fig. 4.2 Diagrama de componentes de la Base de Datos.

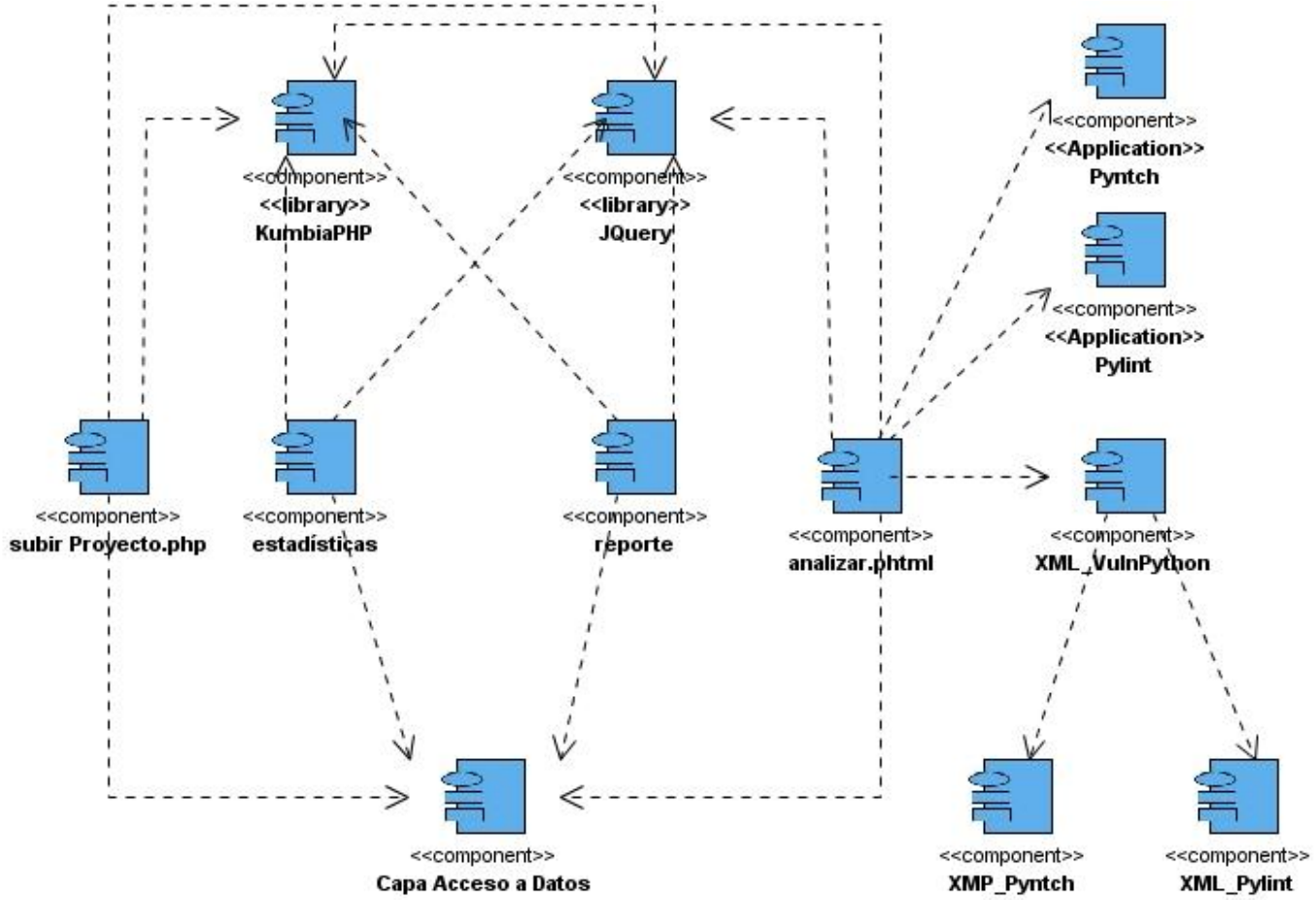


Fig. 4.3 Diagrama de componentes del código fuente.

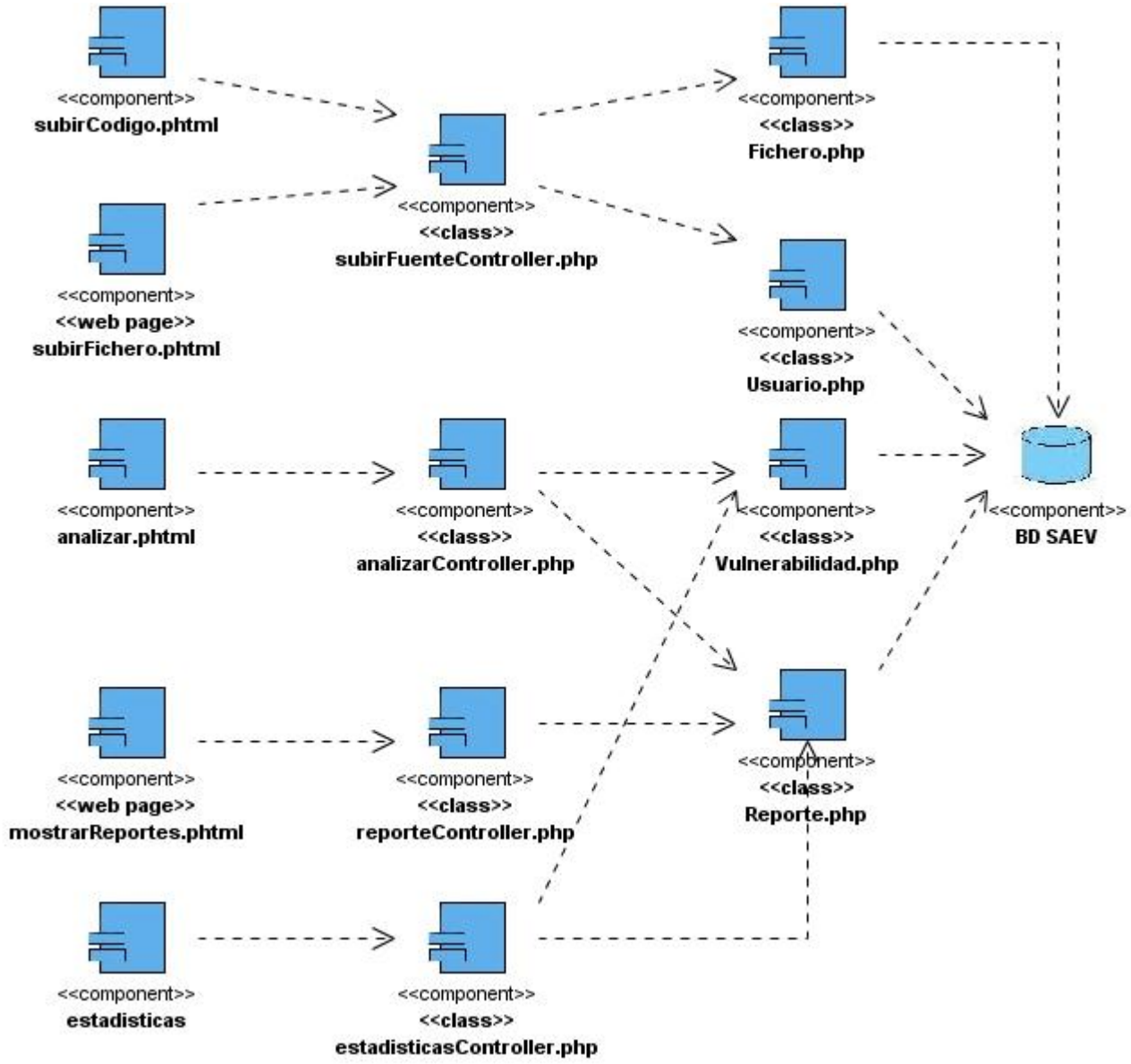


Fig. 4.3 Diagrama de componentes Web (ejecutables).

4.3. Conclusiones

En este capítulo se presentaron artefactos necesarios para la comprensión de cómo el *software* fue implementado, estos fueron el diagrama de despliegue y los de componentes. Los diagramas de despliegue y componentes presentados, describen los componentes a construir y su organización y dependencia entre los nodos físicos.

Capítulo 5
FACTIBILIDAD DEL SISTEMA

5.1. Introducción

La estimación en sentido general de un proyecto, constituye un paso importante, pues permite tener una visión del costo, beneficios, duración y la complejidad del producto, por solo mencionar algunas características observables. Vale destacar que esta solución no pertenece a un proyecto, pero si a medida de que se siga avanzando en el desarrollo, de que se itere el resultado obtenido, de que surjan nuevas necesidades y de que se complete de forma mejorada la solución que hoy se da, podría convertirse en un proyecto con todas las condiciones que este requiere. La estimación también permite a los desarrolladores de la solución una buena planificación personal.

Como la especificación de los requerimientos mediante casos de uso ha probado ser uno de los métodos más efectivos para capturar la funcionalidad de un sistema y la metodología que se utiliza acata este hecho, el presente trabajo de diploma escoge una estimación del esfuerzo basada en casos de uso.

5.2. Método de estimación por Casos de Uso

La estimación mediante el análisis de Puntos de Casos de Uso es un método propuesto originalmente por *Gustav Karner*[3] y posteriormente refinado por muchos otros autores. Se trata de un método de estimación del tiempo de desarrollo de un proyecto mediante la asignación de "pesos" a un cierto número de factores que lo afectan, para finalmente, contabilizar el tiempo total estimado para el proyecto a partir de esos factores.

5.2.1. Cálculo de Puntos de Casos de Uso sin ajustar

[3] *Gustav Karner* trabajó para *Rational Software Corporation* en la década de los 90, donde creó un proyecto de *software* basado en la estimación técnica de utilizar puntos de la sentencia, el cual se conoce hoy como Puntos por Casos de Uso.

$$UUCP = UAW + UUCW$$

UUCP: Puntos de casos de uso sin ajustar.

UAW: Factor de peso de los actores sin ajustar.

UUCW: Factor de peso de los casos de uso sin ajustar.

Tipo de Actor	Descripción	Factor de Peso	Actores	Total
Simple	Otro sistema que interactúa con el sistema a desarrollar mediante una interfaz de programación (*API*, *Application Programming Interface*)	1		
Medio	Otro sistema que interactúa con el sistema a desarrollar mediante un protocolo o una interfaz basada en texto	2		
Complejo	Una persona que interactúa con el sistema mediante una interfaz gráfica	3	1	3
Total				**3**

Tabla 1. Factor de peso de los actores sin ajustar

$$UAW = \sum cantactores * peso$$

UAW = 3

Tipo de CU	Descripción	Factor de Peso	Cantidad de CU	Total
Simple	El Caso de Uso contiene de 1 a 3 transacciones	**5**	**5**	**25**
Medio	El Caso de Uso contiene de 4 a 7	**10**	**1**	**10**

	transacciones			
Complejo	El Caso de Uso contiene más de 8 Transacciones.	**15**		

Tabla 2. Factor de peso de los casos de uso sin ajustar

$$UUCW=\sum cantCU* peso$$

UUCW = 35

Por tanto:

UUCP = UAW + UUCW

UUCP = 3 + 35

UUCP = 38

5.2.2. Cálculo de Puntos de Casos de Uso ajustados

$$UCP=UUCP*TCF*EF$$

Donde:

UCP: Puntos de Casos de Uso ajustados

UUCP: Puntos de Casos de Uso sin ajustar

TCF: Factor de complejidad técnica

EF: Factor de ambiente

El Factor de complejidad técnica (TCF) se calcula mediante la cuantificación de un conjunto de factores que determinan la complejidad técnica del sistema. Cada factor se pondera con un valor desde 0 (aporte no considerable) hasta 5 (aporte muy apreciable).

Significado de los valores

0: No presente o sin influencia.

1: Influencia incidental o presencia incidental.

2: Influencia moderada o presencia moderada.

3: Influencia media o presencia media.

4: Influencia significativa o presencia significativa.

5: Fuerte influencia o fuerte presencia.

Factor	Descripción	Peso	Valor Asignado	Comentario	Total
T1	Sistema distribuido	**2**	**4**	El sistema está distribuido.	**8**
T2	Objetivos de *performance* o tiempo de respuesta	**1**	**5**	El sistema presenta rapidez de respuesta.	**5**
T3	Eficiencia del usuario final	**1**	**2**	La eficiencia del usuario final no es buena.	**2**
T4	Procesamiento interno complejo	**1**	**4**	El procesamiento interno es algo complejo	**4**
T5	El código debe ser reutilizable	**1**	**4**	El código es reutilizable	**4**
T6	Facilidad de instalación	**0.5**	**3**	La instalación no es fácil de realizar.	**1.5**
T7	Facilidad de uso.	**0.5**	**5**	El sistema es fácil de utilizar	**2.5**
T8	Portabilidad.	**2**	**0**	El sistema no es portable	**0**
T9	Facilidad de cambio.	**1**	**5**	Los cambios son fáciles de realizar.	**5**
T10	Concurrencia	**1**	**5**	La concurrencia es buena	**5**
T11	Incluye objetivos especiales de seguridad	**1**	**5**	El sistema incluye varios objetivos de seguridad.	**5**
T12	Provee acceso directo a terceras	**1**	**3**	Tienen acceso directo a	**3**

	partes			terceras partes	
T13	Se requieren facilidades especiales de entrenamiento a los usuarios	**1**	**2**	No se requieren facilidades especiales.	**2**
Total					**47**

Tabla 3. Factor de complejidad técnica

$$TCF = 0.6 + 0.01 * \sum (\text{peso}_i * valorasignado_i)$$

TCF = 0.6 + 0.01 (47)

TCF = 0.6 + 0.47

TCF = 1.07

Factor	Descripción	Peso	Valor Asignado	Comentario	Total
T1	Familiaridad con el modelo de proyecto utilizado	**1.5**	**2**	El equipo de desarrollo está poco familiarizado con el modelo utilizado.	**3**
T2	Experiencia en la aplicación	**0.5**	**2**	La experiencia en la aplicación es baja.	**1**
T3	Experiencia en la orientación a objetos	**1**	**5**	Existe buena experiencia en la orientación a objetos.	**5**
T4	Capacidad de analista líder	**0.5**	**4**	El analista líder está capacitado	**2**
T5	Motivación	**1**	**1**	El equipo no está motivado.	**1**
T6	Estabilidad de requerimientos	**2**	**3**	Los requerimientos son estables	**6**
T7	Personal *Part-Time*	**-1**	**0**	No existe personal *par-time*	**0**
T8	Dificultad del lenguaje de	**-1**	**1**	El lenguaje de programación	**-1**

	programación			no es complejo	
Total					**17**

Tabla 4. Factor ambiente

$$EF = 1.4 - 0.03 * \sum (\text{pesoi} * valor_i)$$

EF = 1.4 − 0.03 (17)

EF = 1.4 − 0.51

EF = 0.89

Por tanto:

UCP = UUCP*TCF*EF

UCP = 38*1.07*0.89

UCP = 36.19

5.2.3. Estimación de Esfuerzo

$$E = UCP * CF$$

E: Esfuerzo estimado en horas hombres.

UCP: Punto de casos de usos ajustados.

CF: Factor de conversión.

Para calcular el factor de conversión (CF), se contabilizan cuántos factores de los que afectan al Factor de ambiente están por debajo del valor medio (3), para los factores E1 a E6 y se contabilizan cuántos factores de los que afectan al Factor de ambiente están por encima del valor medio (3), para los factores E7 y E8.

- Si el total es 2 o menos, se utiliza el factor de conversión 20 horas-hombre/Punto de Casos de Uso, es decir, un Punto de Caso de Uso toma 20 horas-hombre.

- Si el total es 3 o 4, se utiliza el factor de conversión 28 horas-hombre/Punto de Casos de Uso, es decir, un Punto de Caso de Uso toma 28 horas-hombre.
- Si el total es mayor o igual que 5, se recomienda efectuar cambios en el proyecto, ya que se considera que el riesgo de fracaso del mismo es demasiado alto.

Total $_{EF}$ = Cant EF < 3 (entre E1 –E6) + Cant EF > 3 (entre E7, E8)

Total $_{EF}$ = 3

Por tanto:

CF= 28 horas-hombres/Puntos de CU

Luego:

E =UCP*CF

E =36.19 * 28

E = 1013.32 horas-hombres

Calcular esfuerzo de todo el proyecto

Actividad	Porcentaje %	Valor Esfuerzo
Análisis	10%	253.33 horas-hombres
Diseño	20%	506.66 horas-hombres
Implementación	40%	1013.32 horas-hombres
Pruebas	15%	380.00 horas-hombres
Sobrecarga (Otras Actividades)	15%	380.00 horas-hombres
Total	100%	**2533.30 horas-hombres**

Tabla 5. Esfuerzo del proyecto

El valor de esfuerzo calculado **E** representa el esfuerzo del FT Implementación, por comparación salen el resto de los esfuerzo y la suma de ellos es el **esfuerzo total (E_T)**.

Si **E_T** = **2533.30 horas-hombres** y se estima que una persona trabaje 6 horas por día, y un mes tiene como promedio 25 días laborables; la cantidad de horas que puede trabajar una persona en 1 mes es 150 horas. Si **E_T** = **2533.30 horas-hombres** y una persona trabaja 150 horas al mes, entonces quedaría **E_T = 16.89 mes-hombres.** Es decir, un solo hombre puede realizar el proyecto en 16 meses y 28 días aproximadamente.

Para calcular la duración del proyecto entre 2 personas, tenemos que:

Tiempo total del proyecto:
TIEMPO = ET /CH
TIEMPO = 16.89 meses-hombre /2 hombres = 8.45 meses.

5.2.4. Costo

COSTO = CHM ***ET**
CHM = CH * **Salario**

CH: Cantidad de Hombres = 2
El salario mensual es de $100.00 por cada persona, por tanto:
Salario = $100.00
ET: Tiempo total del proyecto

Luego:
CHM = 2.100= 200 $/mes

Por tanto:
COSTO = $200 * 8.45 mes-hombres
COSTO = $1690

De lo obtenido se interpreta que con 2 hombres, el proyecto tiene un tiempo de duración de aproximadamente 8 meses y medio y su costo total se estima en $ 1690.

5.3. Beneficios tangibles e intangibles

Teniendo en cuenta que el Sistema de Análisis Estático de Vulnerabilidades desarrollado, no es un producto elaborado con fines comerciales, ya que es un producto que surge de la necesidad de crear una aplicación que pueda mejorar la seguridad de los proyectos realizados en el lenguaje de programación *Python* y que una gran parte de estos son comercializados con otros países, no se puede hablar de beneficios económicos.

Como beneficios tangibles se mencionan los siguientes:

- La Universidad contará con una aplicación que pueda mejorar la seguridad informática de los proyectos realizados en el lenguaje de programación *Python*.
- Ahorro de recursos, tiempo y mantenimiento del mercado internacional.
- Reusabilidad del código.

Como beneficios intangibles se mencionan los siguientes:

- Motivar a los estudiantes a desarrollar nuevas aplicaciones con analizadores estáticos para otros lenguajes de programación.
- Mayor y mejor aprovechamiento de las tecnologías de la información.

5.4. Análisis de costos y beneficios

El desarrollo del Sistema de Análisis Estático de Vulnerabilidades para el lenguaje de programación *Python*, no conlleva a grandes gastos de recursos. La aplicación es fácil de usar, la navegabilidad y

entorno de la misma se ve favorecido con un diseño muy intuitivo que le permite al usuario navegar sin problemas en la aplicación, la misma presenta una interfaz amigable al usuario.

Es factible desarrollar un sistema para analizar los proyectos que se desarrollan en la Universidad en el lenguaje de programación *Python*, pues no se requiere para la realización de dicho sistema muchos gastos, además la seguridad informática de estos proyectos se ve mejorada, entregándole al cliente un producto de *software* en buen estado. Todo esto trae como repercusión, la creación de una buena reputación a nivel mundial, asegurándole a la Universidad futuras peticiones de proyectos.

5.5. Conclusiones

Se puede concluir que contando con la presencia de 2 hombres el sistema tiene un tiempo de duración de 8 meses y medio, con un costo total estimado en $1690. El desarrollo del producto que se propone es factible, aún teniendo en cuenta los costos de producción, pues se espera la obtención de una solución completa para el ambiente de la seguridad informática que ayude a mejorar la calidad de los proyectos productivos que se realizan en el lenguaje de programación *Python*.

Capítulo 6
PYNTCH Y PYLINT

6.1 Introducción

El presente capítulo tiene como objetivo realizar un estudio sobre los analizadores de código estático que fueron seleccionados para vincularos al sistema a desarrollar. Para ello, se tratarán temas muy importantes como: características, funcionamiento, utilización, forma de instalación, así como otras informaciones que serán de gran ayuda para lograr un mejor entendimiento sobre la utilización de dichas herramientas.

6.2 Pyntch

Antecedentes

Una de las grandes ventajas de los lenguajes de *script* como *Python* es su dinamismo. Puede definir funciones, variables y estructuras de datos siempre que lo desee sin dar más detalles. Sin embargo, esta característica viene con algún costo: a veces es difícil de encontrar posibles errores que son causados por la no coincidencia de tipos antes de que realmente se ejecute el programa. En un lenguaje como *Python*, siempre hay un riesgo de excepciones en tiempo de ejecución no detectada que el programador no puede prever cuando está escribiendo el código; lo cual causa la muerte súbita del programa. Este tipo de comportamiento es particularmente desfavorable para aplicaciones de misión crítica, por lo que se quiere detectar estos errores de antemano. Por desgracia, como el programa se va haciendo cada vez más largo, es complicado realizar un seguimiento de estos tipos de errores y es aún más difícil prevenirlos por entendimiento de cuales tipos o valores pueden ser aprobados o devueltos por cada función.

Pyntch pretende ayudar a reducir estas cargas por inferir qué clase de tipos pueden ser asignados a las variables / miembros / argumentos de función y qué clase de tipos pueden ser devueltos por una función en cualquier momento de la ejecución, y qué tipos de excepciones pueden ser planteadas. Esto se hace

mediante el análisis de código sin ejecutarlo. El objetivo de *Pyntch* es tratar de analizar cada posible ruta de ejecución y todas las posibles combinaciones de datos.

Suena imposible, pero por lo menos esto es parcialmente posible, utilizando una técnica llamada "análisis *typeflow*" (Para más detalles, véase la sección: 6.2.5 Funcionamiento). Sin embargo, también hay un par de inconvenientes, el primero es sobre el propósito de *Pyntch*, quien busca atrapar el mayor número posible de los errores ocultos antes de que el código sea usado en una producción y se centra en la cobertura de los análisis a expensas de su exactitud y el otro inconveniente sería sobre los falsos positivos que trae *Pyntch* en sus resultados, por lo que deben ser examinados por los programadores humanos.

6.2.1 Características

Pyntch (originalmente significa: Verificador de Tipo *Python*), es un analizador de código estático para el lenguaje de programación *Python*. Este detecta posibles errores en tiempo de ejecución antes de que se ejecute el código. *Pyntch* examina el código fuente para deducir todos los tipos posibles de variables, atributos, argumentos de funciones y valores de retorno de cada función o método. Además, detecta las posibles excepciones causadas por falta de coincidencia de tipos, atributos que no se encuentran, u otros tipos de excepciones planteadas por cada función. A diferencia de otros analizadores de código de lenguaje de programación *Python* (como *Pychecker* o *Pyflakes*), *Pyntch* no se ocupa de cuestiones de estilo. *Pyntch* normalmente funciona muy rápido; puede realizar la comprobación de decenas de miles de líneas de código en solo un minuto. Otra característica importante es que *Pyntch* soporta las versiones de *Python* 2.x.

6.2.2 Instalación

La instalación de este analizador estático es bastante sencilla, a continuación, se enumeran los pasos a seguir para lograr dicha instalación:

1. Instalar la versión de Python 2.4 o superior.

2. Descargar el archivo fuente de *Pyntch*.
3. Descompactarlo.
4. Ejecutar el fichero "*setup*.py" para instalar utilizando este comando:

   ```
   # python setup.py install
   ```

5. Y ya está hecho.

6.2.3 ¿Cómo se utiliza?

El principal programa de chequeo que tiene *Pyntch* es: "*tchecker*.*py*". Este viene con dos diferentes modos de funcionamiento: modo resumen y modo comentario.

En el modo de resumen, *Pyntch* sólo muestra los tipos de cada variable y las excepciones en un formato de texto abreviado. En el modo de comentario, hay dos etapas. En primer lugar, el programa analizador genera un archivo *XML* que contiene toda la información sobre un código fuente, seguidamente, otro programa (*annot*.*py*), utiliza esa información para comentar los textos fuentes. A continuación, se muestra un ejemplo de cada modo de funcionamiento:

Modo Resumen

Código Fuente

```
$ cat -n sample.py
     1  import sys, os
     2
     3  # Count the total number of characters for each file in directory.
     4  def countchars(directory):
     5    n = 0
     6    for name in os.listdir(directory):
     7      fp = open(name)
     8      for line in fp:
     9        n += line
    10      fp.fclose()
    11    return n
    12
    13  countchars(sys.argv[1])
```

Fig. 6.1 Fragmento de código del fichero *sample*.py

Corrida del código con el programa de chequeo "*tchecker.py*":

```
$ tchecker.py sample.py
loading: 'sample.py' as 'sample'
...
total files=9, lines=1435 in 0.38sec .....................................[A]
[sample (sample.py)]
  os = <Module os (/home/euske/work/pyntch/pyntch/stub/os.pyi)>
  sys = <Module sys (/home/euske/work/pyntch/pyntch/stub/sys.pyi)>
  ### sample(4) ..........................................................[B]
  # called at sample(13) ..................................................[C]
  def countchars(directory=<str>): ......................................[D]
    fp = <file> .........................................................[D]
    line = <str> ........................................................[D]
    n = <int> ...........................................................[D]
    name = <str> ........................................................[D]
    return = <int> ......................................................[E]
    raises TypeError: not supported operand Add(int, str) at sample:9 ....[F]
    raises AttributeError: attribute not found: @fp.fclose at sample:10 ..[G]
    raises AttributeError: attribute not found: <file>.fclose at sample:10 ...[H]
```

Fig. 6.2 Corrida del fichero *sample*.py utilizado el programa *tchecker.py* en el modo Resumen

El resultado muestra varios detalles:

[A]: El analizador leyó 9 módulos de *Python* (1(objetivo) + 8 importados) que tienen 1.435 líneas en total. El análisis tomó 0,38 segundos.

[B]: La función "countchars" parte de la línea 4 en *sample.py*.

[C]: La función es llamada desde la línea 13.

[D]: Muestra el tipo posible de cada variable.

[E]: La función devuelve un entero.

[F]: La función puede plantear una excepción *TypeError* en la línea 10, tratando de sumar un entero y una cadena.

[G]: Referencia al atributo "*fp.fclose*" falla para cualquier tipo de objetos que "*fp*" podría haber tenido.

[H]: Un objeto "*file*" no tiene atributo "*fclose*".

Modo Comentario

En el modo de anotación (comentario), en primer lugar se necesita ejecutar el programa analizador para producir el resultado en formato *XML,* después, se le da el archivo *XML* a otro programa llamado "*annot.py*." y por último este programa inserta los resultados del análisis en los lugares apropiados en el código fuente, produciendo un texto fuente comentado. A continuación, se ejemplifican estos pasos:

Paso1:

```
$ tchecker -o output.xml sample.py
loading: 'sample.py' as 'sample'
...
total files=9, lines=1435 in 0.40sec
```

Fig. 6.3 Código para ejecutar el programa *tchecker.py* en el modo Comentario

Salida del *XML*

```
<output>
<module src="sample.py" name="sample">
  <var name="os"><compound id="0"><module name="os" /></compound></var>
  <var name="sys"><compound id="0"><module name="sys" /></compound></var>
  <function loc="sample:4:5" name="countchars">
    <caller loc="sample:13" />
    <arg name="directory"><compound id="0"><str /></compound></arg>
    <var name="fp"><compound id="0"><file /></compound></var>
    <var name="line"><compound id="0"><str /></compound></var>
    <var name="n"><compound id="0"><int /></compound></var>
    <var name="name"><compound id="0"><str /></compound></var>
    <return><compound id="0"><int /></compound></return>
    <raise msg="TypeError: not supported operand Add(int, str)" loc="sample:9" type="TypeError" />
    <raise msg="AttributeError: attribute not found: @fp.fclose" loc="sample:10" type="AttributeError" />
    <raise msg="AttributeError: attribute not found: &lt;file&gt;.fclose" loc="sample:10" type="AttributeError" />
  </function>
</module>
</output>
```

Fig. 6.4 Salida en formato *XML* de la corrida al fichero *sample.py* en el modo Comentario

Paso 2:

```
$ annot.py output.xml sample.py
# os = <module os>
# sys = <module sys>
import sys, os

# Count the total number of characters for each file in directory.
def countchars(directory):
  # directory = <str>
  # fp = <file>
  # line = <str>
  # n = <int>
  # name = <str>
  # return <int>
  n = 0
  for name in os.listdir(directory):
    fp = open(name)
    for line in fp:
      # raise TypeError: not supported operand Add(int, str)
      n += line
    # raise AttributeError: attribute not found: @fp.fclose
    # raise AttributeError: attribute not found: <file>.fclose
    fp.fclose()
  return n

countchars(sys.argv[1])
```

Fig. 6.5 Salida comentariada utilizando el programa *annot.py*.

6.2.4 Programas de chequeo

Como se ha visto anteriormente *Pyntch* tiene dos programas principales para realizar el análisis, esta sección está dedicada a indagar aun más sobre estos programas.

Tchecker.py

Como se expresó en el epígrafe anterior, *tchecker.py* es la herramienta de control principal de *Pyntch*, esta permite escanear códigos fuente de *Python* y muestra el resultado del análisis en formato *XML*. La salida puede ser usada más tarde para los propósitos de anotación.

Sintaxis

tchecker.py [*options*] *file.py*...

Opciones

-a:

Muestra el resultado de todos los módulos que se importan, en lugar de los especificados en los argumentos de la línea de comandos.

-c *config_file*

Especifica un archivo de configuración. Un archivo de configuración es un *script* en *Python* que define los parámetros para los ajustes de "*ErrorConfig*".

-C *key = value*

Especifica un parámetro de configuración.

-d

Aumenta el nivel de depuración.

-D

Ignora todas las rutas de acceso de los módulos por defecto de *Python*.

-f *type*

Especifica el formato de salida. Actualmente son soportados *TXT* (modo resumen) o *XML* (modo comentario).

-o *filename*

Especifica un nombre de archivo de salida. Cuando se termina con "*.xml*" se asume automáticamente el formato de salida *XML*.

`-p >python_path`

Agrega una ruta de búsqueda para los módulos de *Python*. Se admiten múltiples opciones (-p).

`-q`

Modo silencioso.

Annot.py

Annot.py como su nombre lo indica, es una herramienta de anotación. Esta herramienta toma una salida *XML* de *tchecker.py* y la combina con el código fuente para producir una salida anotada. Un archivo fuente puede ser especificado por una nombre de ruta real o por un nombre de módulos de *Python* (tal como se utiliza una sentencia "*import*"). Cuando se utiliza un nombre de ruta, la cadena del nombre de ruta debe ser exactamente la que figura en el atributo "*src*" en una salida *XML*.

Sintaxis

```
annot.py [options] output_xml file.py...
                   O
annot.py [options] output_xml module_name...
```

Opciones

`-p basepath`

Especifica el nombre de la ruta de acceso base de ficheros fuente que ha de añadirse a relativas rutas contenidas en el archivo *output_xml*.

`-d`

Aumenta el nivel de depuración.

6.2.5 Funcionamiento Interno

El mecanismo básico de *Pyntch* se basa en la idea del "análisis *typeflow* (análisis de flujo de datos)". Este análisis de flujo de datos da el conjunto máximo de datos posibles que se almacenan en cada lugar en un programa. En primer lugar, este construye un extenso grafo conectado que representa todo el programa de *Python*. Toda expresión o declaración se convierte en un "nodo", que es un lugar abstracto donde cierto tipo de datos es almacenado. Luego, se trata de averiguar qué tipo de datos va desde un nodo a otro. He aquí un pequeño ejemplo:

```
A = 'xyz'
B = 2
C = a*b
```

Teniendo en cuenta las declaraciones anteriores, *Pyntch* construye una gráfica que se muestra en la Fig. 6.6. Un nodo en forma de cuadrado es un nodo "hoja" y representa un solo tipo de objeto de *Python*. Un nodo en forma redonda es un nodo "compuesto", donde uno o más tipos de objetos pueden ser potencialmente almacenados. Ahora, los datos almacenados en la parte superior de los dos nodos hoja, que son un objeto cadena y entero respectivamente, se desprenden a los nodos inferiores y cada nodo pasa los datos de acuerdo a la flecha. Ambos objetos son "mezclados" en el nodo signo “*”, que produce un objeto de cadena. Eventualmente, el objeto va a la variable C, que es el nodo en la parte inferior. De esta manera, se puede inferir el tipo posible de cada variable.

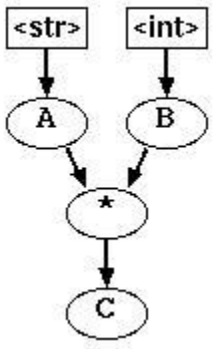


Fig. 6.6 Gráfico que muestra el mecanismo básico de *Pyntch*.

6.2.6 Limitaciones

Uno de los mayores inconvenientes de los análisis "*typeflow*" es su incapacidad para tener en cuenta el orden de ejecución. La secuencia de estados es simplemente ignorada y todo el orden posible es considerado. Esto es como considerar todas las permutaciones de las declaraciones en un programa y su combinación en una sola. Esto a veces trae la inexactitud de su resultado a cambio de una amplitud del chequeo. Un ejemplo esta en los siguientes dos comandos consecutivos:

```
x = 1
x = 'a'
```

Después de ejecutar estas dos declaraciones, es evidente que la variable x tiene siempre un objeto de tipo cadena, no un objeto entero. Sin embargo, debido a la falta de conocimiento del orden de ejecución, *Pyntch* reporta que esta variable puede tener dos tipos posibles: un entero y una cadena.

Otra limitación es que *Pyntch* asume el alcance de cada "*namespace*" definido estáticamente, es decir, todos los nombres (variables, funciones, clases y atributos) están escritos debajo en el código fuente. Por lo tanto, un programa que define o modifica el espacio de nombres de forma dinámica durante la ejecución no puede ser analizado correctamente.

6.2.7 Resumen del Análisis Estático

La siguiente información puede ser recogida en el chequeo estático:

- Posibles tipos de objetos que cada variable, argumento de función o atributo de clase pueda tener.
- Funciones o instancia de métodos que pueden ser llamadas actualmente en cada llamada a la función (teniendo en cuenta el polimorfismo).
- Llamada a lugares para cada función o método.
- Excepciones no detectada tales como:
    - Falta de coincidencia (por ejemplo, la suma de un entero y una cadena).

- Acceso a atributos no definidos (por ejemplo, "*obj*.*attr*" donde "*obj*" no tiene atributo "*attr*").
- Llamar a algo que no se puede llamar (por ejemplo, "*func* (1)" donde "*func*" no es ninguna función, método o clase).
- Acceso a subíndices para objetos que no presentan subíndices (por ejemplo, "a [1]", donde "a" no es una secuencia)
- Iteración sobre objetos no iterables (por ejemplo, "ordenar (x)" donde "x" no es un objeto iterable).

6.3 Pylint

Características

Pylint es una herramienta que todo programador en *Python* debe considerar en su proceso de integración continua (Ej. *Bitten*[4]), básicamente su misión es analizar código en *Python* en busca de errores o síntomas de mala calidad en el código fuente. Cabe destacar que por omisión, la guía de estilo a la que se trata de apegar *Pylint* es la descrita en el *PEP*[5]-8. (20) Es importante resaltar que *Pylint* no sustituye las labores de revisión continua de alto nivel en el proyecto, esto quiere decir, su estructura, arquitectura, comunicación con elementos externos como bibliotecas, diseño, entre otros.

Pylint además, muestra una serie de mensajes cuando analiza el código, así como algunas estadísticas sobre el número de advertencias y errores encontrados en diferentes archivos. Los mensajes se clasifican en distintas categorías, tales como errores y advertencias. Si *Pylint* es ejecutado dos veces, se mostrarán las estadísticas de la ejecución anterior, junto con los de la ejecución actual, de modo que el usuario puede ver si el código ha mejorado o no.

[4] Bitten es un *framework* basado en *Python* para la recogida de diferentes métricas de *software* mediante la integración continua.

[5] *Python Enhancement Proposals*: Propuestas de Mejora de *Python*.

Una de las grandes ventajas de *Pylint* es que es totalmente configurable, además, se pueden escribir complementos o *plugins* para agregar funcionalidades que pueden ser útiles. (20) Por último, pero no menos importante, el código recibe una nota global (ver epígrafe 6.3.2), basado en el número de la gravedad de las advertencias y errores, lo cual ha demostrado ser muy motivador para los programadores.

6.3.1 Instalación

Pylint requiere los paquetes más recientes de *logilab-astng* y *logilab-common*. Este debe ser compatible con cualquier versión superior a *python* 2.2.0 (para la versión 2.2 hay que instalar el paquete *optik*). *Pylint* está disponible en la distribución estándar de *Debian*, así como para *Gentoo*, *Fedora* 4, *Ubuntu*, *FreeBSD*, *Darwin*.

Pasos para la Instalación:

1. Extraer el archivo .*tar* desde la distribución de código fuente.
2. Ir al directorio donde se ha extraído la distribución y ejecutar:
   "*python setup.py install*"
3. Y el programa se habrá instalado.

Para instalar las dependencias se logra de manera similar.

6.3.2 Invocando Pylint

La ejecución de *Pylint* es fácil, sencillamente se debe correr la siguiente línea de comando:

"*pylint [options] module_or_package*"

Se le debe dar a *Pylint* el nombre de un paquete o módulo de *python*. *Pylint* importará este paquete o módulo, por lo que se deberá prestar atención a la ruta de *python*, ya que es un error común analizar una versión instalada de un módulo en lugar de la versión de desarrollo.

También es posible analizar los archivos en *python*, con unas pocas restricciones. Lo que hay que tener en cuenta es que *Pylint* tratará de convertir el nombre de archivo a un nombre de módulo, y sólo será capaz de procesar el archivo si tiene éxito.

En el caso de *Ubuntu*, si el fichero a analizar se encuentra en el directorio /*home*/nombre-usuario solo es necesario correr la siguiente sentencia:

```
pylint mymodule.py
```

En caso de que este en otro lugar, se le debe dar la dirección donde se encuentre:

```
pylint directorio / mymodule.py
```

Fichero de Salida

El formato predeterminado para la salida es texto simple (*.txt*). Pero pasándole a *Pylint* la opción – *html*, este generará el reporte en un documento *HTML*.

Ejemplo:

TXT: `pylint ejemplo.py > reporte.txt`
HTML: `pylint –f html ejemplo.py > reporte.html`

Opciones

En primer lugar, tenemos dos básicas pero útiles opciones.

`--version`

Muestra el número de la versión del programa y sale.

`-h,--help`

Muestra la ayuda sobre las opciones de la línea de comandos.

Pylint presenta una arquitectura ordenada en cuanto a sus analizadores. Por defecto, todos están habilitados. Para deshabilitar o habilitar alguno se muestran las siguientes opciones:

`--enable-<checker> = n` Desactivar un analizador especifico.
`--disable-all` Desactivarlos todos.
`--enable-<checker> = y` Habilitar un analizador en especifico.

Especificar todas las opciones adecuadas para su instalación y las normas de codificación puede ser tedioso, por lo que es posible utilizar un archivo "*rc*" para especificar los valores predeterminados. *Pylint* busca en */etc/pylintrc* y *~ /.pylintrc*. La opción *--generate-rcfile* generará un archivo de configuración comentado de acuerdo con la configuración actual en la salida estándar y la salida del programa.

`--generate-rcfile >.pylintrc`

Se genera un archivo de configuración que queda oculto (por el punto delante del nombre de archivo) en uno de los lugares donde *pylint* va a buscar por defecto un archivo de configuración.
Otras opciones globales útiles son:

`--ignore = file`

Añadir <*file*> (puede ser un directorio) a la lista negra. Debe ser un nombre de base, no una ruta. Esta opción puede configurarse múltiples veces.

`--statistics = y_or_n`

Calcular las estadísticas de los datos recogidos.

`--persistent=y_or_n`

Recogida de datos para comparaciones posteriores.

`--comment = y_or_n`

Añadir un comentario de acuerdo a su nota de evaluación.

`--parseable = y_or_n`

Utilizar un formato de salida analizable.

`--html = y_or_n`

Utilizar *HTML* como formato de salida en lugar de texto.

`--enable-msg = msgids`

Habilitar los mensajes dados.

`--disable-msg = msgids`

Deshabilitar los mensajes dados.

`--enable-msg-cat = cats`

Activar todos los mensajes de la misma categoría.

`--disable-msg-cat = cats`

Desactivar todos los mensajes de la misma categoría.

`--errors-only`

Habilitar sólo los analizadores de la categoría de error.

6.3.3 Análisis de Código

Entre la serie de revisiones que hace *Pylint* al código fuente se encuentran las siguientes:

- Revisiones básicas:
  - Presencia de cadenas de documentación (*docstring*).
  - Nombres de módulos, clases, funciones, métodos, argumentos, variables.
  - Número de argumentos, variables locales, retornos y sentencias en funciones y métodos.
  - Atributos requeridos para módulos.
  - Valores por omisión no recomendados como argumentos.
  - Redefinición de funciones, métodos, clases.
  - Uso de declaraciones globales.
- Revisión de variables:
  - Determina si una variable o *import* no está siendo usado.
  - Variables indefinidas.
  - Redefinición de variables proveniente de módulos *builtins* o de ámbito externo.
  - Uso de una variable antes de asignación de valor.
- Revisión de clases:
  - Métodos sin *self* como primer argumento.
  - Acceso único a miembros existentes vía *self*
  - Atributos no definidos en el método *__init__*
  - Código inalcanzable.
- Revisión de diseño:
  - Número de métodos, atributos, variables locales, entre otros.
  - Tamaño, complejidad de funciones, métodos, entre otros.
- Revisión de *imports*:
  - Dependencias externas.
  - *Imports* relativos o importe de todos los métodos, variables vía * (*wildcard*).
  - Uso de *imports* cíclicos.
  - Uso de módulos obsoletos.
- Conflictos entre viejo/nuevo estilo:
  - Uso de *property*, *__slots__*, *super*.

  - Uso de *super*.
- Revisiones de formato:
  - Construcciones no autorizadas.
  - Sangrado estricto del código.
  - Longitud de la línea.
  - Uso de <> en vez de "!=".
- Otras revisiones:
  - Notas de alerta en el código como *FIXME*, *XXX*.
  - Búsqueda por similitudes o duplicación en el código fuente.
  - Revisión de excepciones.
  - Revisiones de tipo, haciendo uso de inferencia de tipos.

Mientras *Pylint* hace el análisis estático del código, muestra una serie de mensajes así como también algunas estadísticas acerca del número de advertencias y errores encontrados en los diferentes ficheros. Estos mensajes son clasificados bajo ciertas categorías, entre las cuales se encuentran:

Refactorización [R]

Asociado a una violación en alguna buena práctica.

Convención [C]

Asociada a una violación al estándar de codificación.

Advertencia [W]

Asociadas a problemas de estilo o errores de programación menores.

Error [E]

Asociados a errores de programación importantes, es probable que se trate de un *bug*.

Fatal [F]

Asociados a errores que no permiten a *Pylint* avanzar en su análisis.

Un ejemplo de este reporte se puede ver a continuación:

```
Messages by category
--------------------

+-----------+-------+---------+-----------+
|type       |number |previous |difference |
+===========+=======+=========+===========+
|convention |969    |1721     |-752.00    |
+-----------+-------+---------+-----------+
|refactor   |267    |182      |+85.00     |
+-----------+-------+---------+-----------+
|warning    |763    |826      |-63.00     |
+-----------+-------+---------+-----------+
|error      |78     |291      |-213.00    |
+-----------+-------+---------+-----------+
```

Fig. 6.7 Reporte de mensajes por categoría.

Reportes

Pylint ofrece una sección dedicada a los reportes, esta se encuentra inmediatamente después de la sección de mensajes de análisis, cada uno de estos reportes se enfocan en un aspecto particular del proyecto (20), como los que se muestran a continuación:

- Número de mensajes por categorías (mostrado arriba).
- Dependencias internas y externas de los módulos.
- Número de módulos procesados.
- Porcentaje de errores y advertencias encontradas por módulo.
- Total de errores y advertencias encontradas.
- Porcentaje de clases, funciones y módulos con *docstrings* y su respectiva comparación con un análisis previo (si existe).
- Porcentaje de clases, funciones y módulos con nombres correctos (de acuerdo al estándar de codificación).
- Otros

Al final del reporte arrojado por *Pylint* se podrá observar una puntuación general por su código, basado en el número y la severidad de los errores y advertencias encontradas a lo largo del código fuente. Estos resultados suelen motivar a los desarrolladores a mejorar cada día más la calidad de su código fuente.

Como se dijo anteriormente, si *Pylint* se ejecuta varias veces sobre el mismo código, se podrá ver el puntaje de la corrida previa junto al resultado de la corrida actual, de esta manera se puede saber si se ha mejorado la calidad del código o no.

```
Global evaluation
-----------------
Your code has been rated at 7.74/10 (previous run: 4.64/10)
If you commit now, people should not be making nasty comments about you on
c.l.py
```

Falsos Positivos

Pylint puede arrojar falsos positivos, esto puede entenderse al recibir una alerta por parte de *Pylint* de alguna cuestión que se realizó conscientemente. Ciertamente algunas de las advertencias encontradas por *Pylint* pueden ser peligrosas en algunos contextos, pero puede que en otros no lo sea, o simplemente *Pylint* haga revisiones de cosas que al usuario realmente no le importen. El usuario puede ajustar la configuración de *Pylint* para no ser informado acerca de ciertos tipos de advertencias o errores.

6.3.4 Otras Informaciones

Integración con *IDEs*

Pylint se integra en los siguientes editores / *IDEs*:

- *Emacs*
- *Eric3*

- *Eclipse* (usando el plugin *PyDev* [6])

Proyectos que utilizan *Pylint*

Los siguientes proyectos son conocidos por el uso de *Pylint* para mejorar su código:

- *OSAF Chandler*
- *Xen*
- *CPS*
- *ERP5*
- *pyxmpp*
- *mercurial*
- *eXe*
- *PrimaGIS*
- *python-cdd*
- *CDSWare*
- *ASE*
- *RunJob*
- *Slugathon*
- *Topographica* (por lo menos la intención de hacerlo)

6.3.5 Modificaciones

En esta sección se encuentran reflejadas algunas funciones que pueden ser riesgosas para las aplicaciones *web* implementadas en *Python* y para el Sistema Operativo. Con el objetivo de detectarlas antes de que ocurran males mayores se le realizaron un grupo de modificaciones a la herramienta de

[6] *PyDev* es un *IDE* de *Python* para *Eclipse*, el cual puede ser usado para el desarrollo en *Python*, *Jython* e *IronPython*.

análisis estático *Pylint*. En total fueron insertadas en la herramienta 38 funciones, pero solo se mencionan las más importantes teniendo en cuenta el nivel de gravedad.

Validar la entrada para evitar Inyecciones

Cualquier programa que interactúa con una aplicación fuera de sí puede ser aprovechado si los datos proporcionados por el usuario son incorrectos. Esto puede llevar a dos tipos diferentes de inyección. En primer lugar, las inyecciones *SQL* surgen cuando se envían datos incorrectos a una aplicación, que serán insertados en una base de datos; los datos maliciosos contienen comandos *SQL* y por lo tanto, la ejecución ocurre con privilegios de nivel de aplicación, con el objetivo de corromper, robar o borrar información. En segundo lugar, cualquier usuario que facilite datos que no son chequeados, se pueden utilizar para pasar parámetros o comandos maliciosos al sistema operativo, al intérprete de *Python*, o ambas cosas.

En este sentido, cualquier llamada al sistema *Python* es sospechosa, ya que interactúa con el sistema operativo.

Módulo *os*

El módulo donde se encuentran las funciones que realizan llamadas al sistema *Python* o al sistema operativo es llamado: os**.** Los métodos de *Python* de los cuales se debe tener especial cuidado en este modulo son:

- `exec()`
- `eval()`
- `system()`
- `popen()`
- `execfile()`
- `input()`
- `compile()`

Existen funciones que realizan llamadas con el objetivo de recibir ciertos resultados, pero dichos resultados pueden ser falsificados fácilmente por los atacantes y llegan a ser inseguros. Ejemplo de estas funciones son:

- `getlogin()`
- `ttyname()`

Muchas llamadas para generar nombres de archivos temporales no son seguros (sensibles a las condiciones de corrida). Algunos de estos son:

- `tmpnam()`
- `tmpfile()`

Módulo *socket*

Los resultados de las llamadas a funciones que involucran a *DNS* pueden ser fácilmente falsificados por un atacante y no deben se confiados. Las funciones relacionadas con este módulo son:

- `gethostbyaddr()`
- `gethostbyname()`
- `gethostbyname_ex()`

Módulo *Random*

Los generadores estándar de números aleatorios de *Python* no deben ser utilizados para generar aleatoriedad por razones de seguridad, debido a que los métodos criptográficos que utiliza *Python* no son muy buenos. Algunas funciones generadoras de números aleatorios son:

- `random()`
- `randint()`
- `randrange()`
- `setstate()`

- *getstate()*
- *jumpahead()*
- *shuffle()*
- *choice()*
- *gauss()*
- *gammavariate()*
- *normalvariate()*
- *paretovariate()*

6.4 Conclusiones

Con el estudio realizado en este capítulo se pudieron ver las potencialidades de estos analizadores estáticos de código, que a pesar de tener algunas limitaciones y arrojar falsos positivos, son realmente útiles para lograr un código limpio. Además, se mostraron las funciones que pueden causar vulnerabilidades a la aplicación o directamente al sistema operativo, para lo cual se realizaron modificaciones en el analizador *Pylint* con el objetivo de eliminar dichas vulnerabilidades.

.

CONCLUSIONES GENERALES

Al concluir este trabajo, con el desarrollo del sistema de análisis estático de código para el lenguaje de programación *Python*, se solucionó satisfactoriamente el problema existente en cuanto a mejorar la seguridad de los proyectos realizados en este lenguaje y la utilización de buenas prácticas de programación, dando cumplimiento al objetivo general de la investigación presentada.

Para el desarrollo de dicho sistema se realizó un proceso de investigación basado en el estado del arte relativo a las herramientas de análisis de código estático existentes en el mundo, resultando elegidas *Pyntch* y *Pylint*, con el propósito de realizar un mejor análisis. Todo el proceso de desarrollo del *software* ha estado sustentado en la utilización de metodologías y herramientas eficientes definidas con anterioridad, tales como *RUP*, para guiar el trabajo del equipo de desarrollo a través de un ciclo que permita de manera organizada traducir los requisitos de *software* a un sistema que garantice la solución a la problemática existente. Entre las herramientas escogidas figuran el *framework KumbiaPHP* versión 1.0 *stable* y la herramienta para el modelado *UML*: *Visual Paradigm* debido entre otros factores a su soporte multiplataforma. Durante la fase de inicio del proyecto fue necesario realizar un modelo de dominio que permitiera ofrecer una visión clara de los principales conceptos empleados asociados a las entidades implicadas; realizando además descripciones detalladas de todos los casos de uso del sistema, sirviendo de base para los posteriores flujos de Análisis y Diseño. Con la culminación de la fase de Elaboración del *software* quedó definida una arquitectura Orientada a Objetos; dando paso a la implementación del sistema, todo lo cual es expresado mediante diagramas de componentes que reflejan la relación entre clases y paquetes. El sistema realizado presenta un conjunto de funcionalidades básicas que le permiten al usuario realizar el análisis de un proyecto que se ha subido, ya sea insertando su código en un área de texto o mediante determinados formatos. Además, se le permite revisar los reportes generados del análisis de sus proyectos, dándole la posibilidad de eliminar el que desee. Por último, se le brinda un conjunto de estadísticas referentes a las vulnerabilidades encontradas en los proyectos realizados, donde se puede observar con que calidad han sido realizados dichos proyectos.

Finalmente, con este sistema la Universidad podrá mejorar la seguridad informática que presentan los proyectos realizados en el lenguaje de programación *Python*, permitiéndole de esta manera mantener una

buena reputación a nivel mundial para con los que solicitan sus servicios; así como, erradicar en gran medida las vulnerabilidades que presentan dichos proyectos.

RECOMENDACIONES

Como resultado del presente trabajo se obtuvo un sistema informático que aunque resuelve la problemática planteada inicialmente, puede ser refinado con el objetivo de lograr un incremento en su calidad, a continuación se exponen algunas recomendaciones a tener en cuenta para futuras versiones del producto:

- Incluir nuevas herramientas de análisis estático o realizar modificaciones a las que se utilizan actualmente.
- Adicionar un nuevo módulo para el análisis estático utilizando otras herramientas para otros lenguajes de programación.
- Adicionar nuevas funcionalidades al sistema implementado.
- Analizar después de un período de tiempo el impacto que ha tenido el *software* en la Universidad.
- Permitir al usuario configurar el análisis de código estático.
- Tener en cuenta el proyecto *Owasp-python-static-analysis* para futuros análisis.

Referencias Bibliográficas

1. **Duque, Raúl González.** *Python para todos.* España : s.n., 1995. pág. 7.

2. —. *Python para todos.* España : s.n., 1995. pág. 9.

3. **Seguridad Informática.** *Sistemas de prevención de incidentes de seguridad.* Ciudad Habana : s.n., 2009.

4. **Peña, Javier Fernández-Sanguino.** Programación Segura. *Administración y seguridad en entornos GNU/Linux.* 6 de junio de 2006.

5. **Rodríguez, Luis.** Causas de Ataques y Analizador. *Herramientas para análisis estático de seguridad: estado del arte.* 2009.

6. **Ivar Jacobson, Grady Booch y James Rumbaugh.** *El Proceso Unificado de Desarrollo de Software.* Madrid : Pearson Education S.A, 2000.

7. **Schmuller, Joseph.** *Aprendiendo UML en 24 horas.* México : Pearson Educación, 2000.

8. **slideshare.** slideshare. [En línea] [Citado el: 2 de febrero de 2010.] http://www.slideshare.net/vanquishdarkenigma/visual-paradigm-for-uml.

9. **UTN-FRLP(Facultad Regional La Plata).** UTN-FRLP(Facultad Regional La Plata). [En línea] [Citado el: 1 de febrero de 2010.] http://www.frlp.utn.edu.ar/materias/info2/bases_de_datos.htm.

10. **Ciberaula.** Ciberaula. [En línea] 1996. [Citado el: 1 de febrero de 2010.] http://linux.ciberaula.com/articulo/linux_apache_intro/.

11. **Valdés, Damián Pérez.** Maestros del web. [En línea] 19 de julio de 1997. [Citado el: 7 de noviembre de 2009.] http://www.maestrosdelweb.com/principiantes/los-diferentes-lenguajes-de-programacion-para-la-web/.

12. **S, Christian Van Der Henst.** Maestros del web. [En línea] 19 de julio de 1997. [Citado el: 7 de noviembre de 2009.] http://www.maestrosdelweb.com/editorial/phpintro/.

13. **Gutiérrez, Javier J.** ¿Qué es un framework web? [En línea] [Citado el: 7 de noviembre de 2009.] http://www.cssblog.es/guias/Framework.pdf.

14. **Gutierrez, Andrés Felípe.** *Kumbia PHP Framework. Porque Programar debería ser más fácil.* pág. 18.

15. **KumbiaPHP Framework.** KumbiaPHP Framework. [En línea] [Citado el: 30 de enero de 2010.] http://www.kumbiaphp.com/blog/2009/03/27/benchmark_frameworks_kumbiaphp_vs_symfony_zend_cakephp/.

16. **El Directorio.** El Directorio. [En línea] [Citado el: 18 de noviembre de 2009.] http://el-directorio.org/vsftpd.

17. **Gutierrez, Andrés Felípe.** *Kumbia PHP Framework. Porque Programar debería ser más fácil.* pág. 42.

18. —. *Kumbia PHP Framework. Porque Programar debería ser más fácil.* pág. 113.

19. —. *Kumbia PHP Framework. Porque Programar debería ser más fácil.* pág. 31.

20. **Mazzarri, Milton.** milmazz. [En línea] [Citado el: 3 de marzo de 2010.] http://blog.milmazz.com.ve/archivos/2010/03/13/pylint-analisis-estatico-del-codigo-en-python.

Bibliografía

1. **Oscar M. Fernández Carrasco, Delba García León y Alfa Beltrán Benavides.** Un enfoque actual sobre la calidad del software. [Online] 1995. http://bvs.sld.cu/revistas/aci/vol3_3_95/aci05395.htm.

2. **Duque, Raúl González.** *Python para todos.* España : s.n.

3. **Rodríguez, Luis.** *Herramientas para análisis estático de seguridad: estado del arte.* 2009

4. **Seguridad Informática.** Programación Segura de Aplicaciones. *Sistemas de prevención de incidentes de seguridad.* 2009

5. **Peña, Javier Fernández-Sanguino.** Programación Segura. *Administración y seguridad en entornos GNU/Linux.* 2006.

6. **Muller, Neil.** Code analysis for Python.Pyfakes, Pychecker & Pylint, marzo 21, 2009.

7. **Shinyama, Yusuke.** Pyntch.Static code analyzer for Python. [Online] 2009. [Citado el: 3 de Febrero de 2010.] http://www.unixuser.org/~euske/python/pyntch/index.html.

8. **Fortify.** Fortify Software. [Online] [Citado el: 3 de Febrero de 2010.] http://www.fortify.com/security-resources/rats.jsp.

9. **OWASP.** OWASP. [Online] [Citado el: 3 de Febrero de 2010.] http://www.owasp.org/index.php/Category:OWASP_Python_Static_Analysis_Project.

10. **slideshare.** slideshare. [Online] http://www.slideshare.net/vanquishdarkenigma/visual-paradigm-for-uml.

11. **Facultad Regional La Plata.** UTN-FRLP(Facultad Regional La Plata). [Online] http://www.frlp.utn.edu.ar/materias/info2/bases_de_datos.htm.

12. **Ciberaula.** Ciberaula. [Online] 1996. [Citado el: 10 de Febrero de 2010.] http://linux.ciberaula.com/articulo/linux_apache_intro/.

13. **The Apache Software Foundation.** The Apache Software Foundation. [Online] [Citado el: 10 de Febrero de 2010.] http://www.apache.org/.

14. **Valdés, Damián Pérez.** Maestros del web. [Online] julio 19, 1997. [Citado el: 12 de Febrero de 2010.] http://www.maestrosdelweb.com/principiantes/los-diferentes-lenguajes-de-programacion-para-la-web/.

15. **S, Christian Van Der Henst.** Maestros del web. [Online] julio 19, 1997. [Citado el: 12 de Febrero de 2010.] http://www.maestrosdelweb.com/editorial/phpintro/.

16. **Gutierrez, Andrés Felípe.** *Kumbia PHP Framework. Porque Programar debería ser más fácil.*

17. **KumbiaPHP Framework.** KumbiaPHP Framework. [Online] [Citado el: 14 de Febrero de 2010.] http://www.kumbiaphp.com/blog/2009/03/27/benchmark_frameworks_kumbiaphp_vs_symfony_zend_cakephp/.

18. **El Directorio.** El Directorio. [Online] [Citado el: 14 de Febrero de 2010.] http://el-directorio.org/vsftpd.

19. **vsftpd.** vsftpd. [Online] [Citado el: 14 de Febrero de 2010.] http://vsftpd.beasts.org/.

20. **Ivar Jacobson, Grady Booch y James Rumbaugh.** *El Proceso Unificado de Desarrollo de Software.* Madrid : Pearson Education S.A, 2000.

21. **Schmuller, Joseph.** *Aprendiendo UML en 24 horas.* México : Pearson Educación, 2000.

22. **Larman, Craig.** *UML y patrones.* s.l. : Prentice Hall Iberoamericana.

23. **Escobal, M.** Código Atado con Alambre. [Online] [Citado el: 15 de Diciembre de 2009.] http://conalambre.wordpress.com/2009/02/04/python-en-apache-2-para-tarados/.

24. **Logilab.** Logilab. [Online] [Citado el: 15 de Diciembre de 2009.] http://www.logilab.org/card/pylint_manual.

25. **Mazzarri, Milton.** milmazz. [Online] [Citado el: 15 de Diciembre de 2009.] http://blog.milmazz.com.ve/archivos/2010/03/13/pylint-analisis-estatico-del-codigo-en-python.

26. **Bitten.** Bitten. [Online] [Citado el: 5 deMarzo de 2010.] http://bitten.edgewall.org/.

27. **Pydev.** Pydev. [Online] [Citado el: 5 deMarzo de 2010.] http://pydev.org/index.html.

28. **Carroll, Ed.** Iron Speed. [Online] [Citado el: 10 deMarzo de 2010.] http://www.ironspeed.com/articles/Estimating%20Software%20via%20Use%20Cases/article.aspx.

29. **Lukaszewski, Al.** About.com: Python. [Online] [Citado el: 10 deMarzo de 2010.] http://python.about.com/od/cgiformswithpython/ss/ProgramSecurity.htm.

30. **masadelante.com.** masadelante.com. [Online] [Citado el: 11 deMarzo de 2010.] http://www.masadelante.com/faqs/host.

31. Ayuda de Python 2.5.

32. **SAEV.** Manual de Despliegue de SAEV. 2010.

ANEXOS

Anexo 1. Análisis Estático

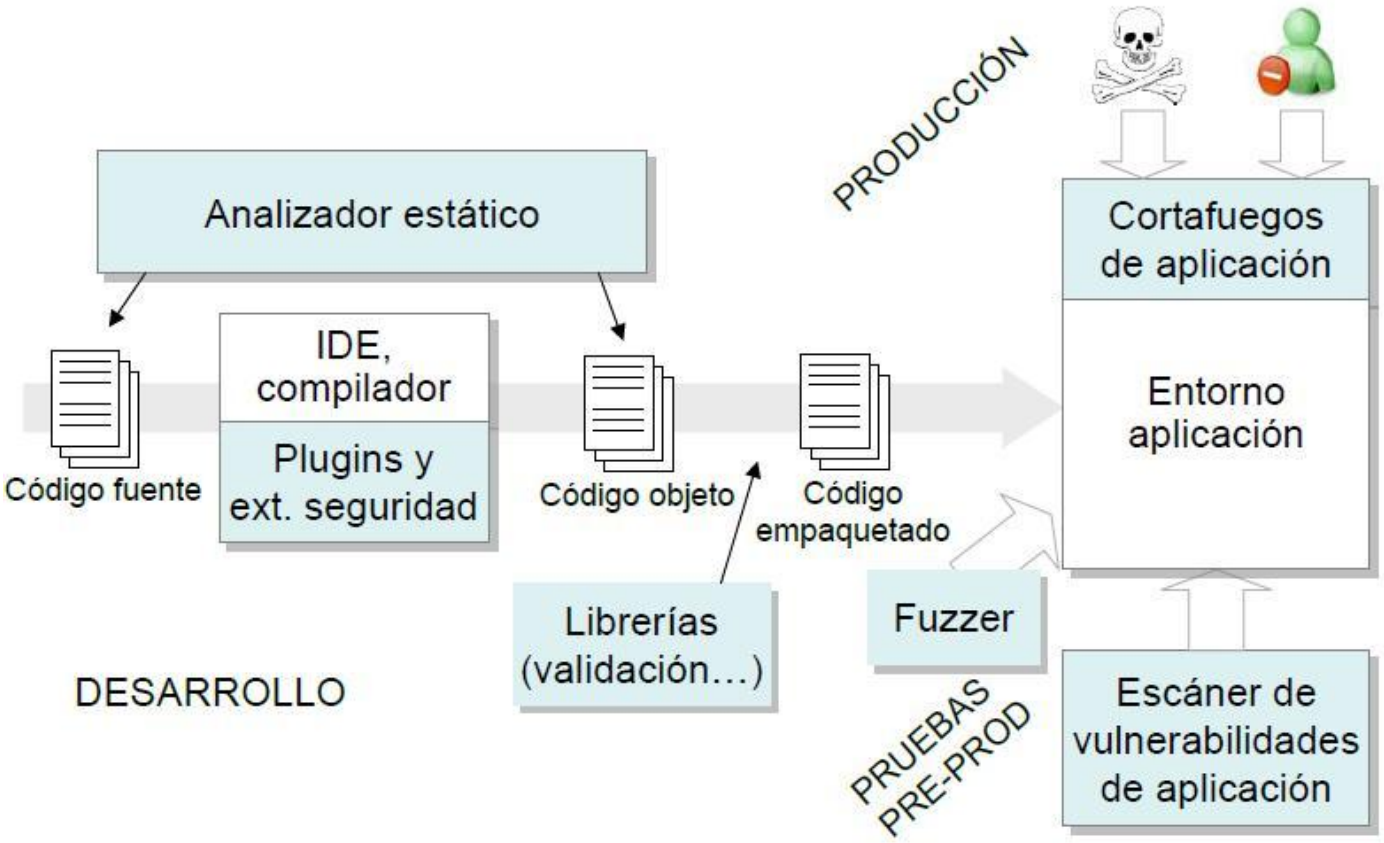


Fig. 8 Herramienta de análisis estático.

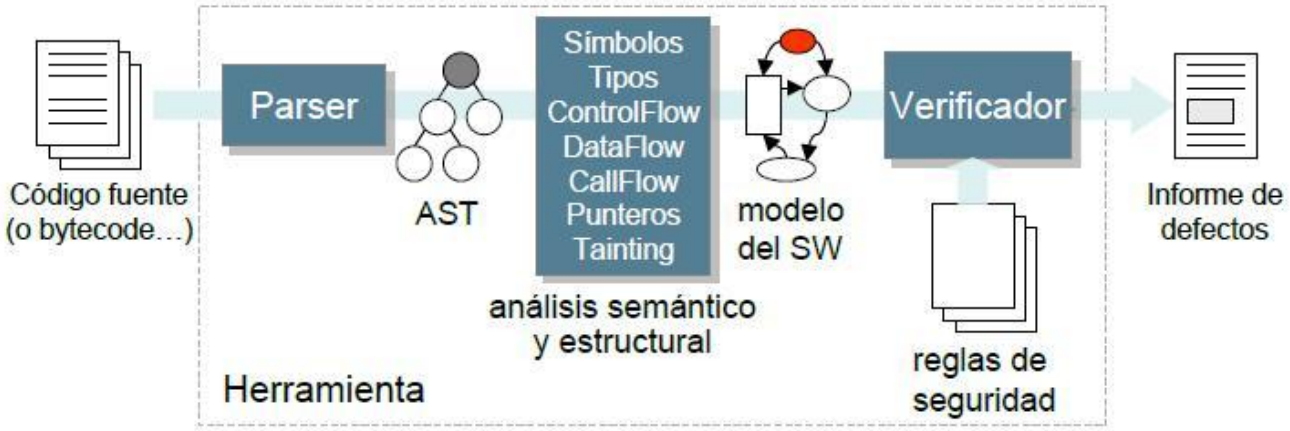


Fig. 9 Interioridades del análisis estático.

Anexo 2. Benchmark realizado a diferentes frameworks.

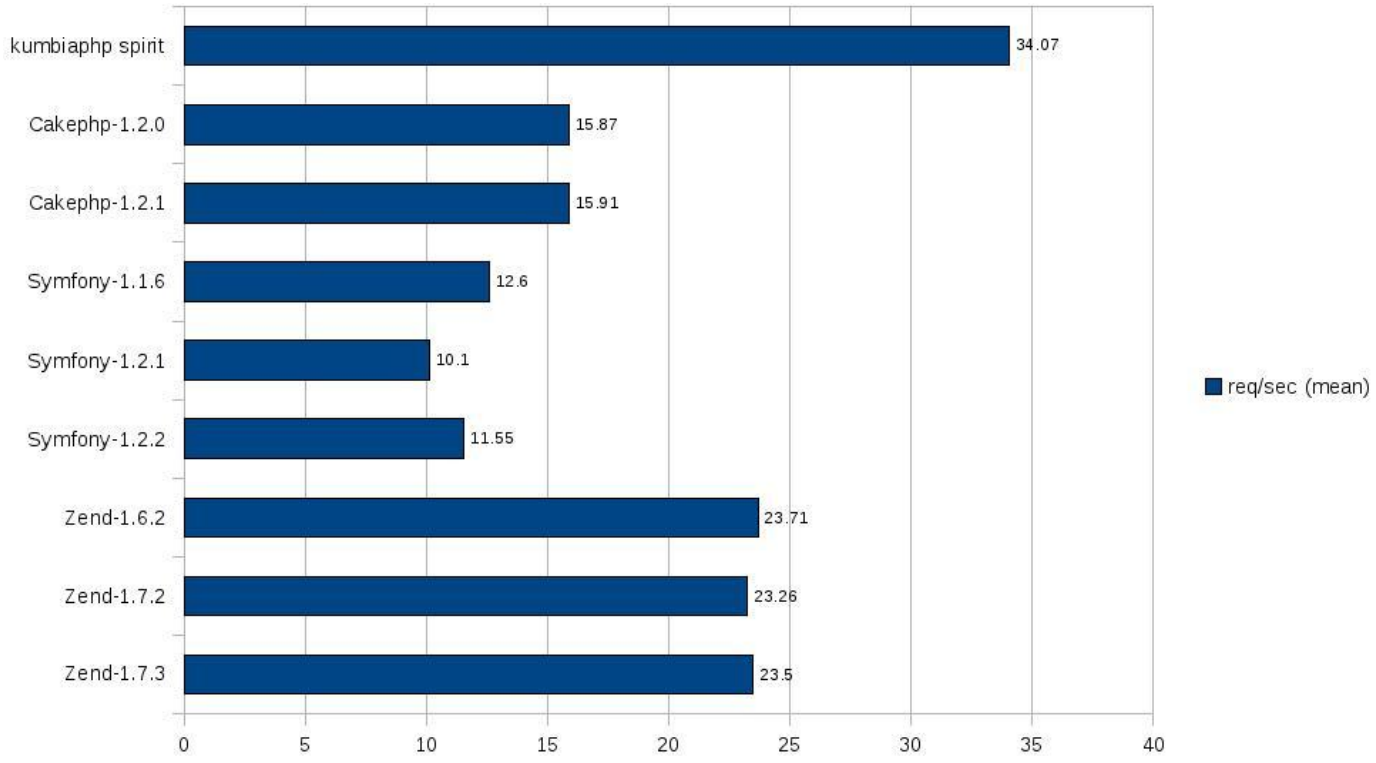


Fig. 10 Benchmark realizado a diferentes frameworks.

Anexo 3. Diagramas de Secuencia

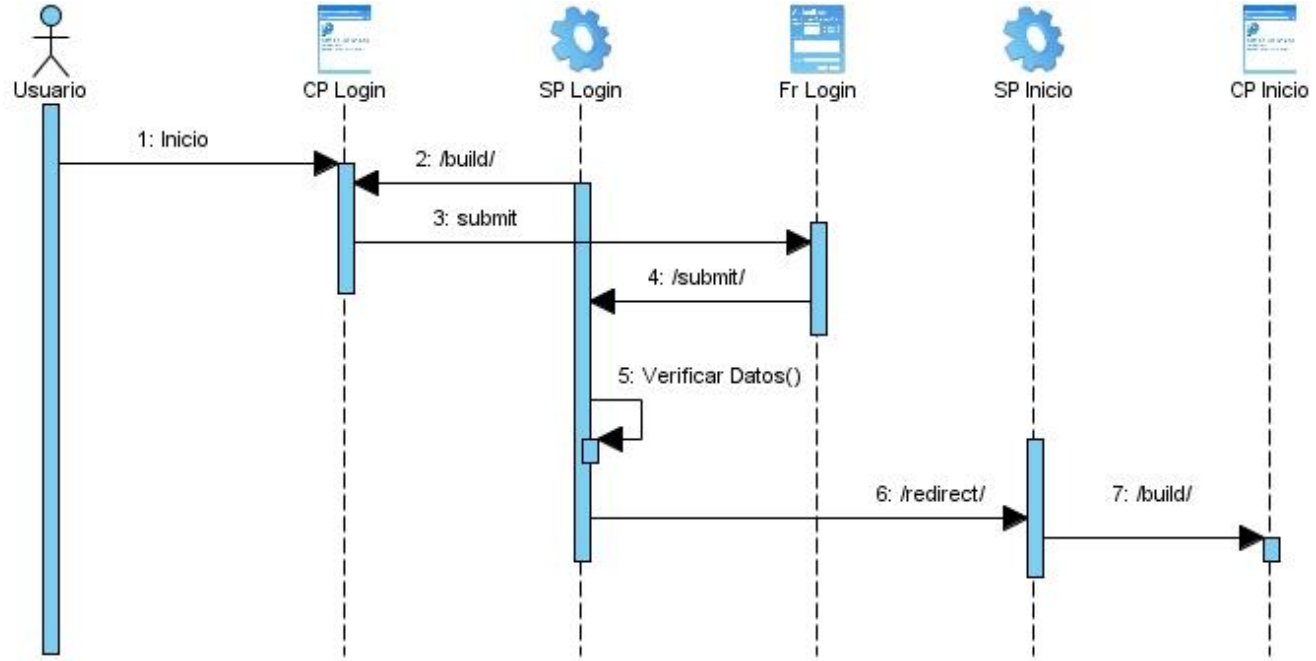


Fig. 11 Diagrama de Secuencia Autenticar Usuario.

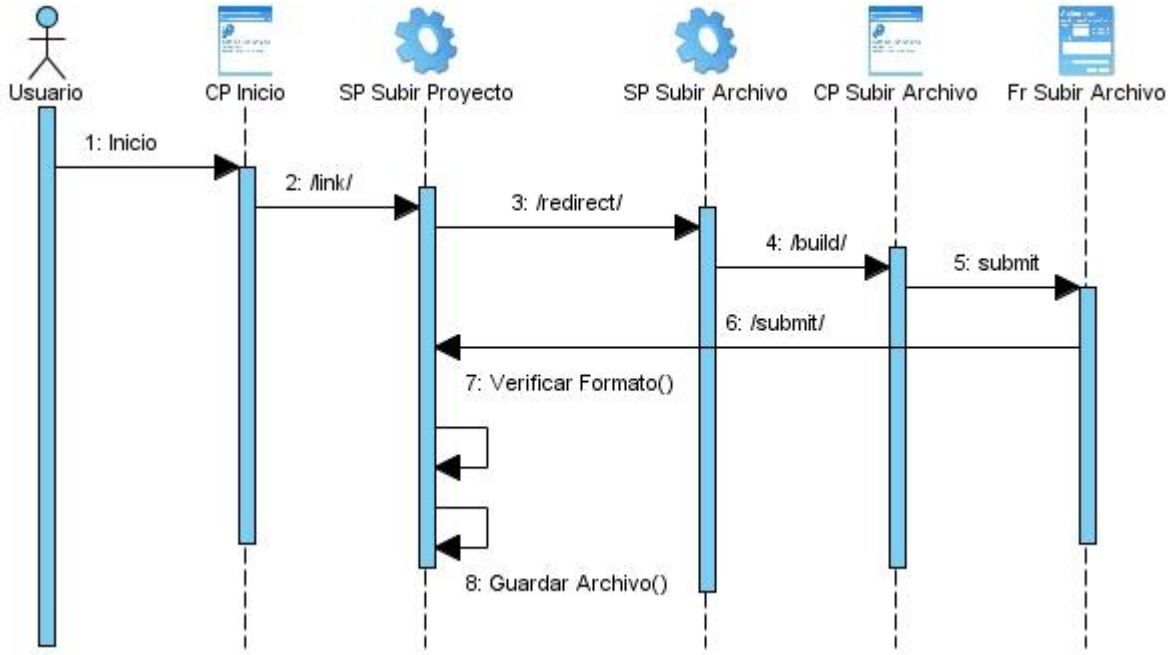


Fig. 12 Diagrama de Secuencia Subir Proyecto.

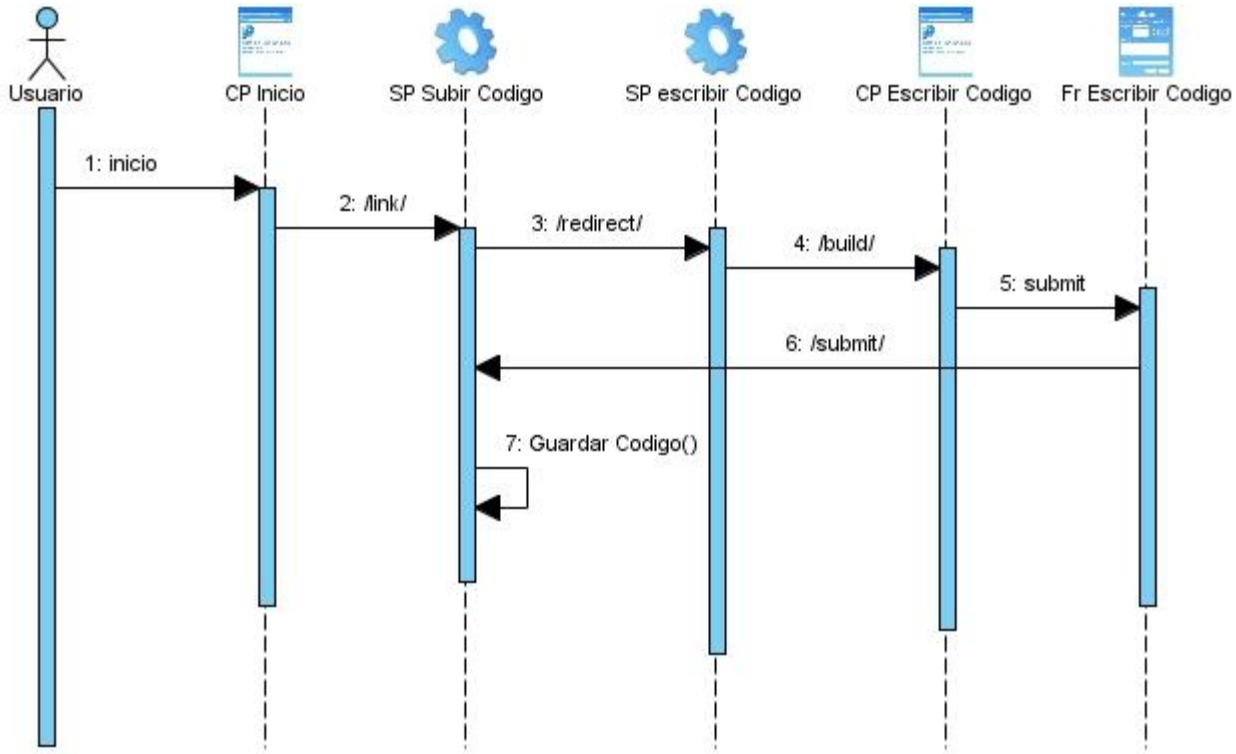


Fig. 13 Diagrama de Secuencia Subir Código.

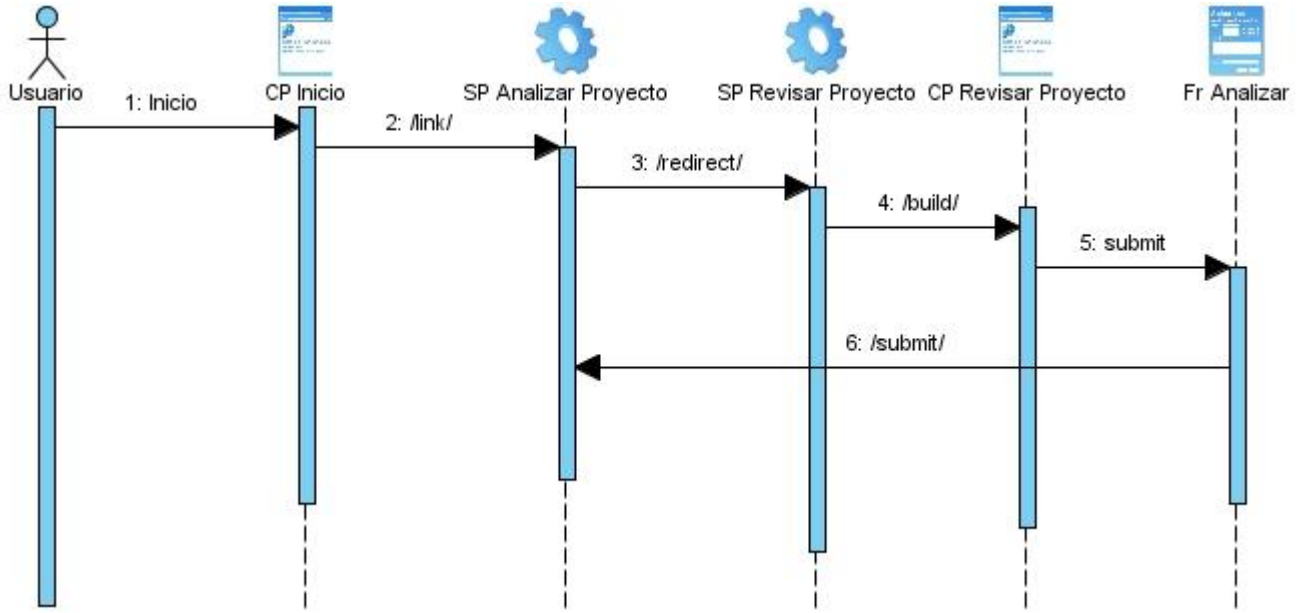


Fig. 14 Diagrama de Secuencia Revisar Proyecto.

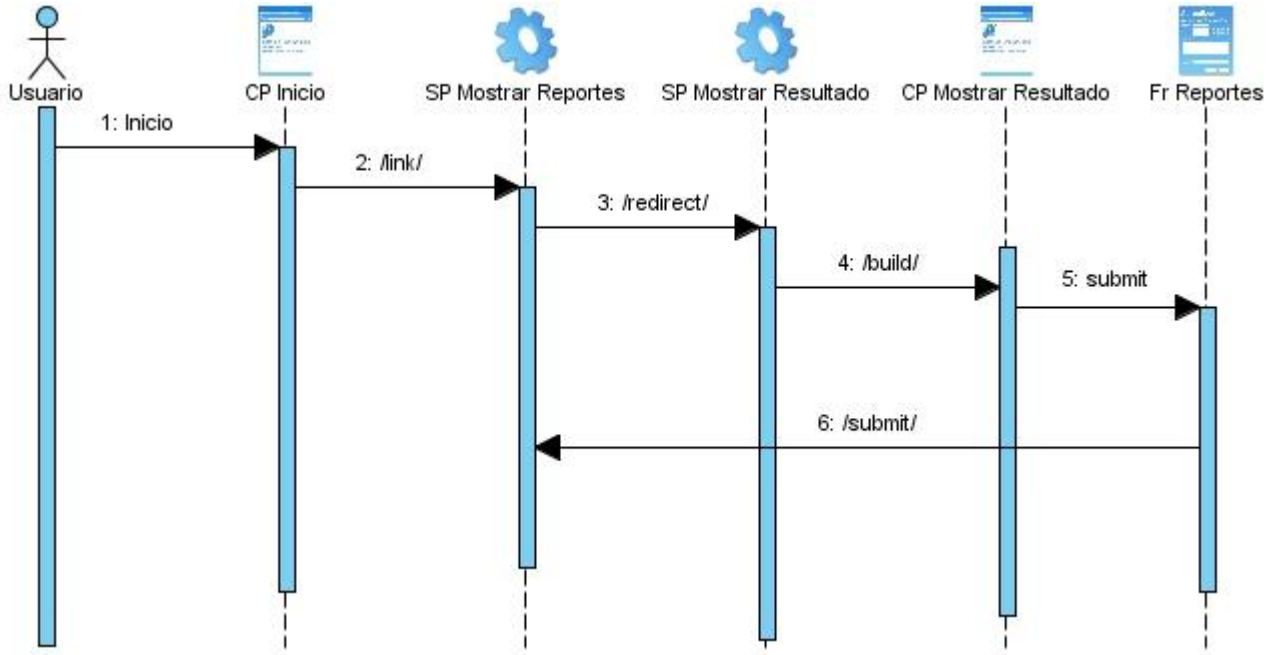


Fig. 15 Diagrama de Secuencia Mostrar Reportes.

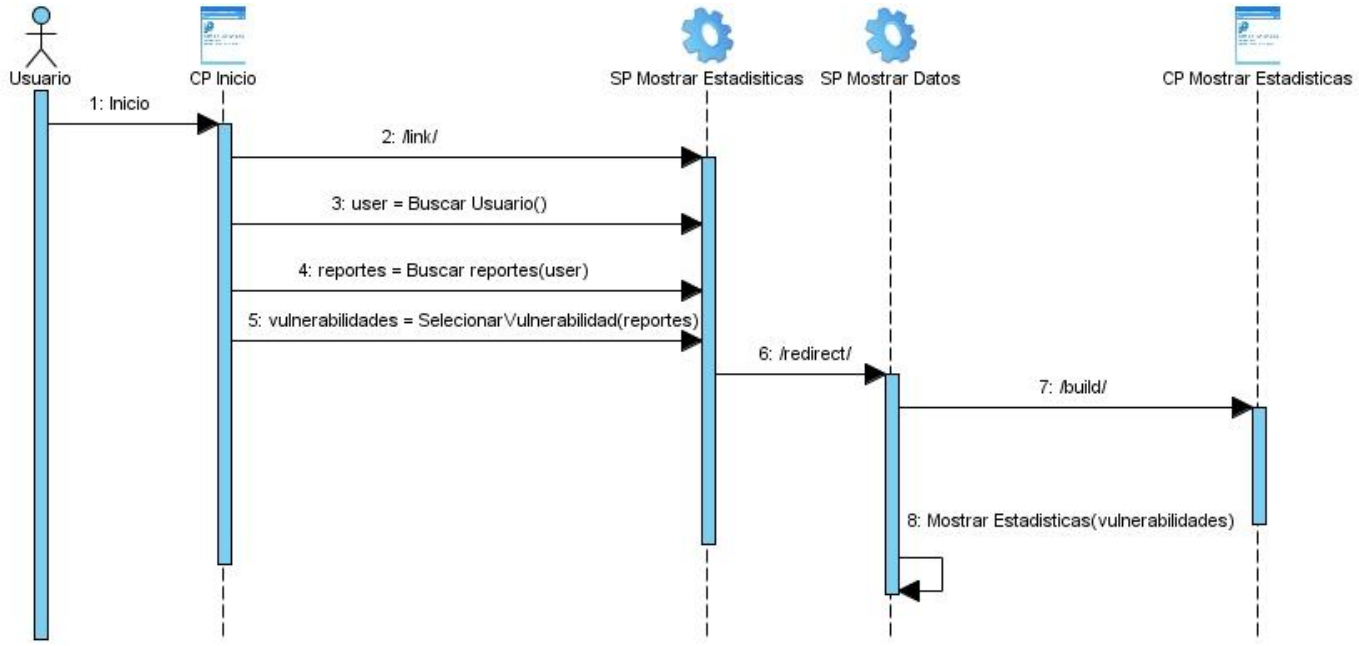


Fig. 16 Diagrama de Secuencia Mostrar Estadísticas.

Anexo 4. Modelo Vista Controlador (MVC) de Kumbia

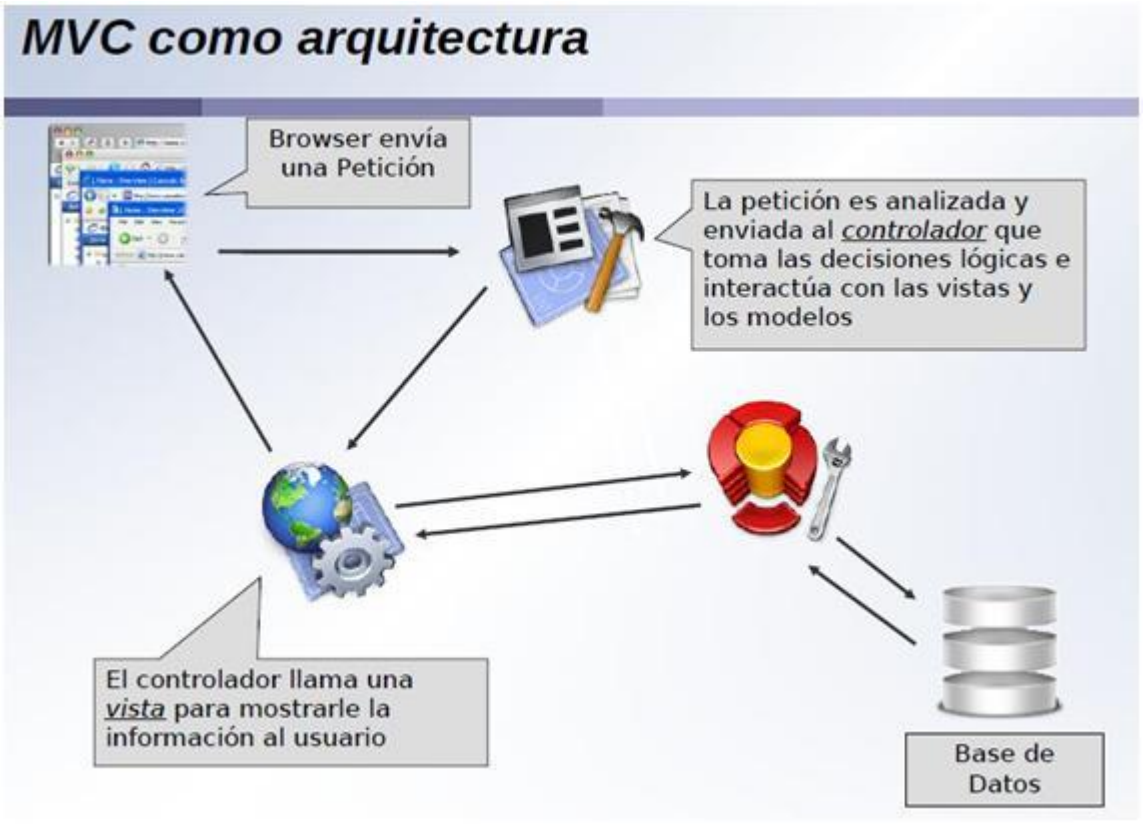


Fig. 17 Modelo Vista Controlador (MVC) Clásico.

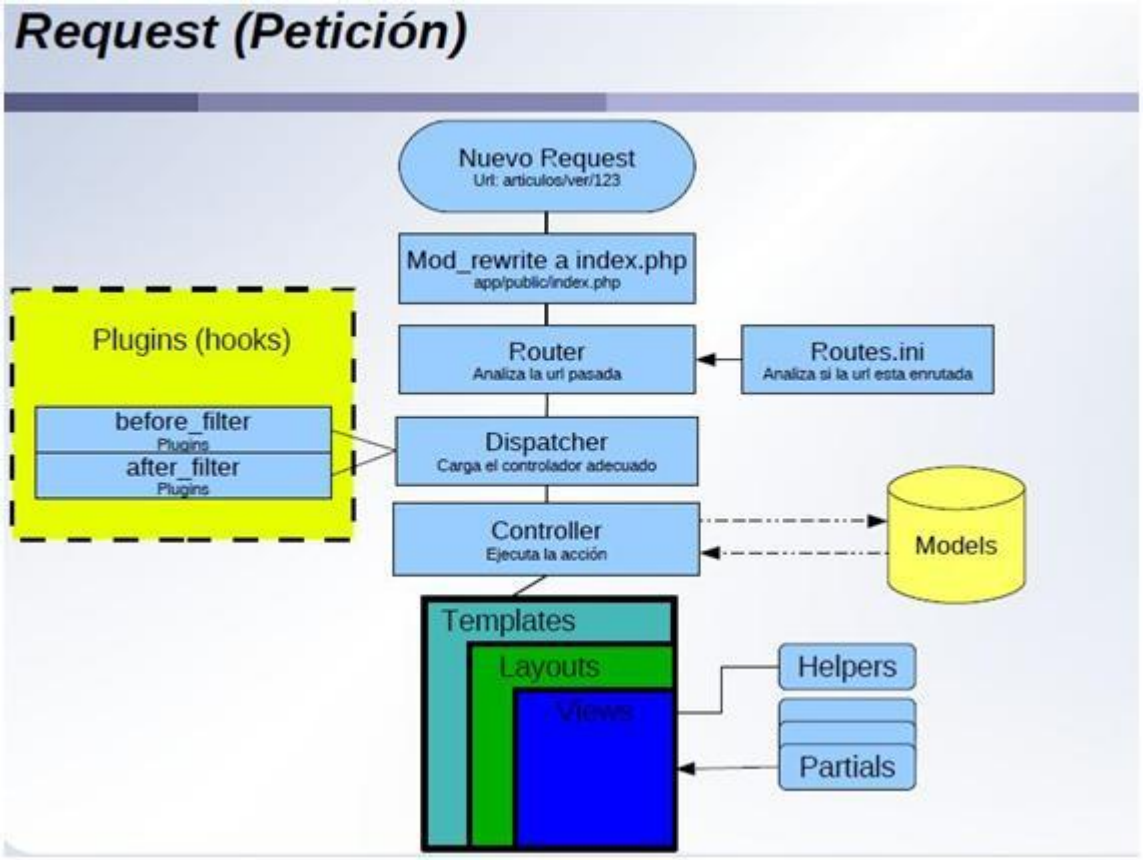


Fig. 18 Forma en que *KumbiaPHP* atiende las peticiones.

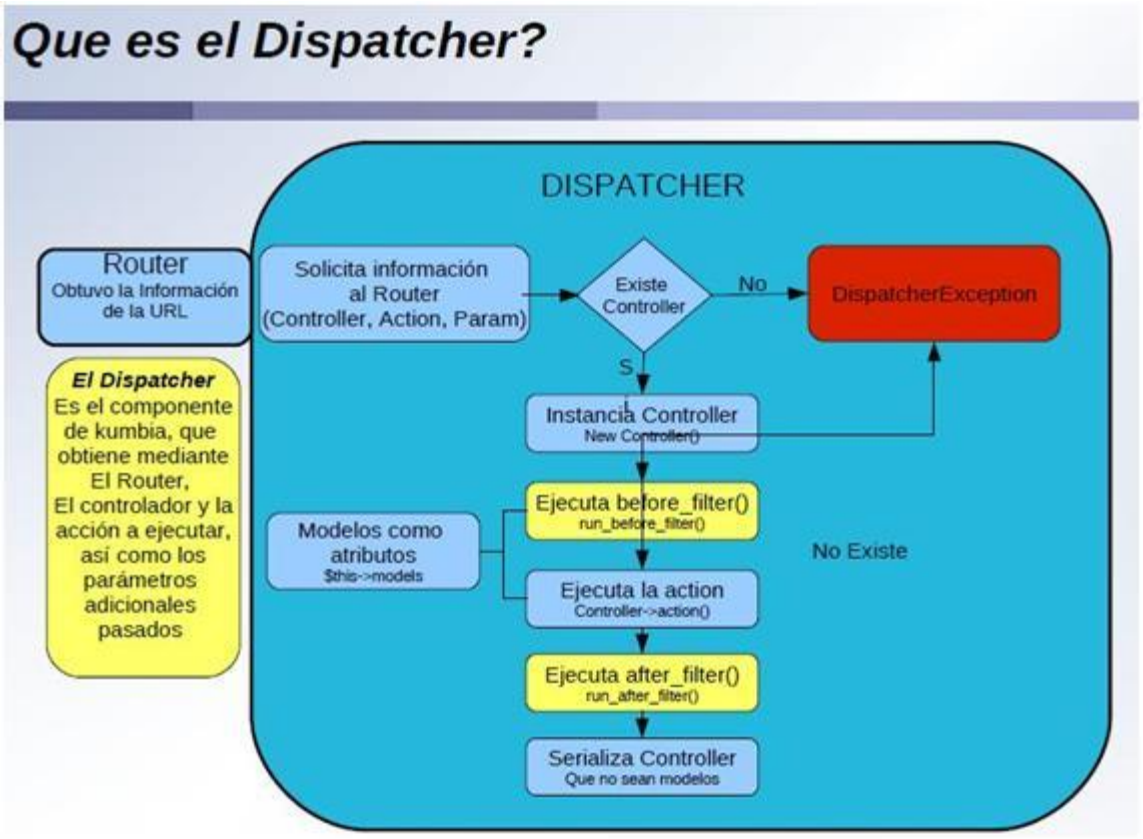


Fig. 19 Funcionamiento del *Dispatcher*.

GLOSARIO DE TÉRMINOS

Programación Orientada a Objetos (POO)

Es un paradigma de programación que usa objetos y sus interacciones, para diseñar aplicaciones y programas de ordenador. Está basado en varias técnicas, incluyendo herencia, abstracción, polimorfismo y encapsulamiento. En la actualidad, existe variedad de lenguajes de programación que soportan la orientación a objetos.

Software libre

Es la denominación del *software* que brinda libertad a los usuarios sobre un producto adquirido y por tanto, una vez obtenido, puede ser usado, copiado, estudiado, modificado y redistribuido libremente.

Lenguajes interpretados o de script

Un lenguaje interpretado o de *script* es un lenguaje de programación que está diseñado para ser ejecutado por medio de un intérprete, en contraste con los lenguajes compilados. En estos lenguajes las instrucciones se traducen o interpretan una a una siendo mucho más lentos que los programas compilados.

Lenguajes compilados

Un programa escrito en un lenguaje "compilado" se traduce a través de un programa anexo llamado compilador que, a su vez, crea un nuevo archivo independiente que no necesita ningún otro programa para ejecutarse a sí mismo. Este archivo se llama ejecutable. Los lenguajes compilados son lenguajes de alto nivel en los que las instrucciones se traducen del lenguaje utilizado a código máquina para una ejecución rápida.

Tipado dinámico

Un lenguaje de programación es dinámicamente tipado si una misma variable puede tomar valores de distinto tipo en distintos momentos. Esta nos indica que no es necesario declarar el tipo de dato que va a contener una variable, sino que el tipo se declara en tiempo de ejecución y está determinado por su asignación.

Falsos Positivos

Consisten en aquellas alarmas que tienen lugar cuando en realidad no existe ninguna vulnerabilidad. Existen códigos, algunas de cuyas partes coinciden con patrones de ataque de desbordamiento de búfer, que son detectados como intrusiones, cuando en realidad no lo son.

Falsos Negativos

Son uno de los tipos de falsas alarmas, y se producen cuando no se emite el correspondiente aviso cuando existe realmente una vulnerabilidad. Este tipo de situaciones, también representa un problema.

MULTIHILO

Una aplicación multihilo es aquella que ejecuta varias instrucciones a la vez, ideal para paralelizar procesos que pasan mucho tiempo en espera.

HTML

(*HyperText Markup Language*), lenguaje de marcas hipertextuales, lenguaje de marcación diseñado para estructurar textos y presentarlos en forma de hipertexto, que es el formato estándar de las páginas *web*. Gracias a *Internet* y a los navegadores del tipo *Internet Explorer*, *Opera*, *Firefox* o *Netscape*, el *HTML* se ha convertido en uno de los formatos más populares que existen para la construcción de documentos y también de los más fáciles de aprender.

UML

(*Unified Modeling Language*) Lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de *software*.

RUP

El Proceso Unificado de Racional (*Rational Unified Process*) es un proceso de desarrollo de *software* y junto con el Lenguaje Unificado de Modelado *UML*, constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos.

GOF

(*Gang of Four*) Banda de los Cuatro. Así se les conoce a los autores del libro “Patrones de Diseño” (*Design Patterns*: *Elements of Reusable Object-Oriented Software*), referencia en el campo del diseño orientado a objeto.

API

Interfaz de programación de aplicaciones (*Applications Programming Interface*): una serie de funciones que están disponibles para realizar programas para un cierto entorno.

XML

(*Extensible Markup Language*) Lenguaje de Marcas Extensible. Metalenguaje extensible de etiquetas desarrollado por el *W3C*. Es una tecnología sencilla que tiene a su alrededor otras que la complementan y la hacen mucho más grande y con unas posibilidades mucho mayores. Tiene un papel muy importante en la actualidad ya que permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil.