Análisis y diseño de una herramienta para la evaluación de estilos de código para la asignatura Práctica Profesional 1 de la Universidad de las Ciencias Informáticas.

Autor:

Oksana Morejón Sánchez

Tutores:

Ing. Maikel Pereira Ojeda

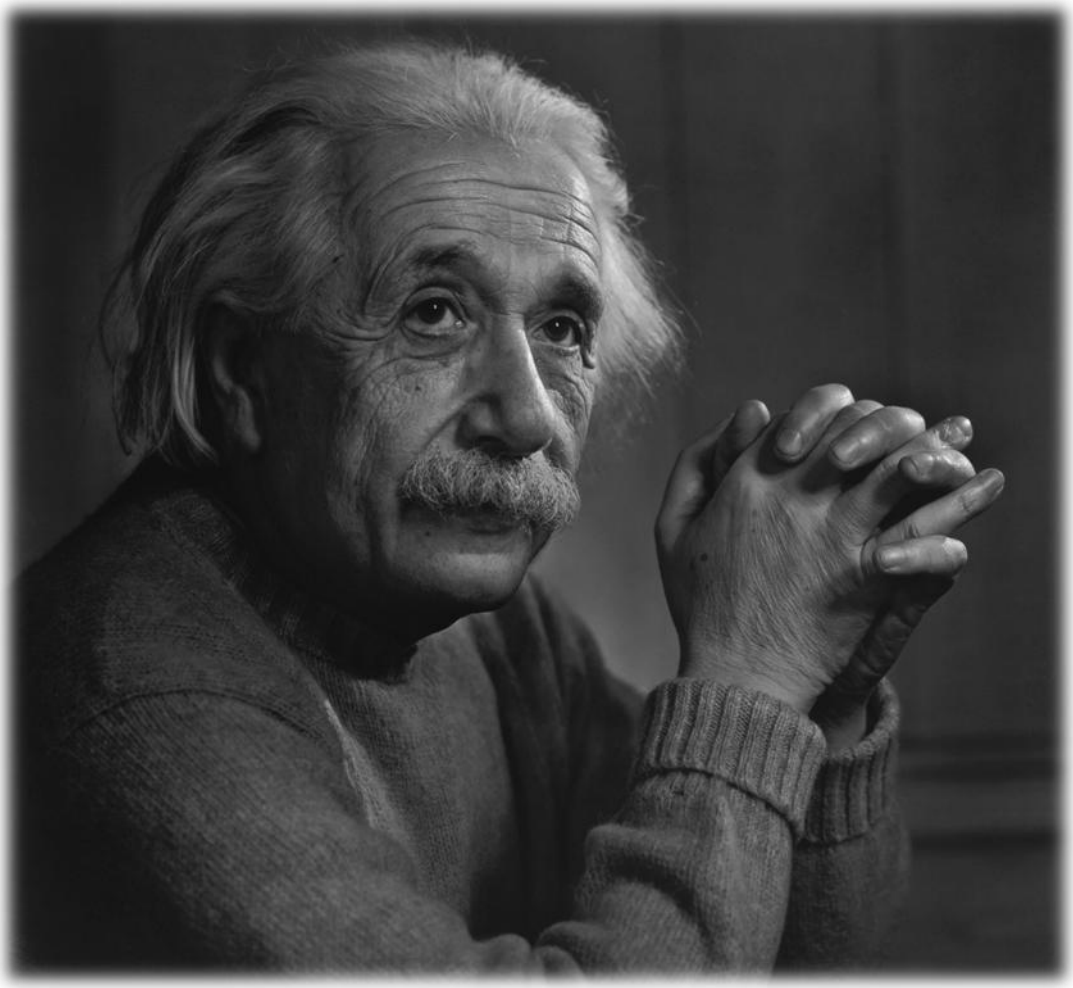
Ing. Yanet Casado Maceo

Co-tutor:

Ing. José Ernesto Lara

Ciudad de La Habana, Cuba, junio 2011

"Año del 53 Aniversario del Triunfo de la Revolución"



“La verdadera muestra de inteligencia no es el conocimiento, sino la imaginación.”

Albert Einstein

A mis padres:

Quienes me dieron la vida, quienes sin esperar nada a cambio, lo dieron todo. Quienes me guiaron por un camino de rectitud y me enseñaron que es lo mejor. A un par de corazones buenos: con gratitud eterna.

Declaración de Autoría

Declaro que soy el único autor de este trabajo y autorizo a la Universidad de la Ciencias Informáticas a hacer uso del mismo en su beneficio.

Y para que así conste firmo la presente a los ____ días del mes de Mayo de 2011.

Autor:

_____
Oksana Morejón Sánchez

Tutores

_____ _____

Ing. Maikel Pereira Ojeda Ing. Yanet Casado Maceo

Agradecimiento:

Resumen

Estudios realizados demuestran la importancia de los estilos de códigos en el desarrollo de aplicaciones informáticas. La evaluación o verificación de estilos de código se realiza automáticamente en la actualidad, utilizando alguna de las numerosas herramientas disponibles. La mayoría de estas herramientas evalúan solamente el código escrito en un solo lenguaje de programación y en muchos casos dependen de algún entorno de desarrollo específico o sistema operativo. En la Universidad de las Ciencias Informáticas, en el Departamento Central de Práctica Profesional no se cuenta con una herramienta para agilizar el proceso de enseñanza – aprendizaje de estilos de codificación en la asignatura Práctica Profesional 1. El presente trabajo propone el análisis y diseño de una aplicación Web que automatice el proceso evaluación de estilos de código en diferentes lenguajes de programación.

Introducción

Las Tecnologías de la Información y las Comunicaciones (TIC) han evolucionado a una velocidad increíble en las últimas décadas, convirtiéndose en un apoyo fundamental para la sociedad en su desarrollo. La informática ha sido uno de los principales factores que influyen en la competitividad entre las empresas, que en una fuerte lucha por satisfacer las necesidades del cliente siempre desean mejores aplicaciones en menos tiempo y al menor costo posible.

A raíz del impacto que ha causado en el mundo el avance de la industria del software, el gobierno de Cuba se propuso hace algunos años desarrollar esta esfera. Uno de los pasos principales en la consecución de este objetivo fue la creación de la Universidad de las Ciencias Informáticas (UCI), que surgió con el objetivo de formar profesionales capaces de informatizar cada rincón de nuestro país y de trabajar para la exportación de software, renglón que se espera vaya ganando importancia en las exportaciones cubanas.

Un Ingeniero informático representa un papel indispensable en el ciclo de vida de un software, y puede ejecutar roles diferentes durante el desarrollo de este (1), velando siempre por la calidad, tanto del producto, como del proceso de construcción. Por esta razón la formación del profesional de la informática debe ser excelente, sobre todo en el aspecto práctico.

La práctica laboral, en los estudios superiores de las diversas ingenierías, prepara a los estudiantes para escenarios reales de producción. En la UCI los esfuerzos en esta dirección se reúnen en la disciplina integradora Práctica Profesional. Esta juega distintos roles en cada uno de los ciclos que componen la actividad formativa de la universidad: el ciclo básico y el profesional. En el primero la práctica profesional tiene como objetivo crear las habilidades básicas que tributan a las competencias transversales que debe desarrollar el Ingeniero en Ciencias Informáticas. Durante el ciclo profesional el estudiante se inserta en los proyectos productivos, en los que adquiere competencias técnicas a partir de los roles que desempeña en los equipos de proyecto.

En cada uno de los semestres de la carrera existe una asignatura integradora que forma parte de la disciplina Práctica Profesional. En el primer año de la carrera se cuenta con la Práctica Profesional 1 (PP1), asignatura de carácter anual, que durante el primer semestre, motiva a los estudiantes a partir del desarrollo de habilidades generales que le serán útiles en el ejercicio de la profesión, y en el segundo semestre crea habilidades básicas relacionadas con el rol de programador, haciendo especial énfasis en las buenas prácticas que debe seguir el estudiante para desempeñarlo con calidad en cada una de las fases del ciclo de desarrollo de un software.

Introducción.

En la asignatura PP1 dentro de los temas que se imparten en clase se abordan los diferentes estilos de codificación para diversos lenguajes de programación. Es importante que un programador conozca los estilos de codificación pues las convenciones de código mejoran la lectura del software, permitiendo entender el código mucho más rápido, haciendo más eficiente el trabajo en equipo, la documentación del software y la retroalimentación del propio programador.

Actualmente existen herramientas en el mundo que verifican, o evalúan, si un código cumple con un estilo de codificación determinado, muchas de estas son libres e implementan funcionalidades muy útiles para la verificación de convenciones, pero tienen el inconveniente de que presentan en su mayoría una interfaz compleja, en algunas inclusive es necesario definir las reglas a evaluar a través de líneas de código fuente, o utilizando como interfaz la línea de comandos, además de que cada una de estas herramientas están concebidas para un solo lenguaje de programación y muchas forman parte de un entorno de desarrollo específico o deben integrarse a uno.

La asignatura PP1 dedica un tema a las buenas prácticas de desarrollo de software y dentro de ellas los estándares de codificación o estilos de código. Actualmente los profesores dedican más tiempo que el deseable para la planificación y ejecución de estas clases debido a la variedad de herramientas que es necesario conocer, y transmitir su conocimiento a los estudiantes. El uso de algunas de estas herramientas se dificulta además por no ser multiplataforma y ser aplicaciones de escritorio, que deben estar instaladas en cada una de las computadoras de la clase. O sea, no existe una herramienta que evalúe, a través de una interfaz sencilla, fragmentos de códigos escritos siguiendo diferentes estilos de programación, que pueda soportar varios lenguajes de programación, y que además sea fácilmente accesible e independiente del sistema operativo.

Planteada la **situación problemática** se puede enunciar el **problema a resolver:** ¿Cómo garantizar la implementación de una herramienta para la evaluación de estilos de código para la asignatura Práctica Profesional 1 de la Universidad de las Ciencias Informática?

Para darle respuesta se define como **objeto de estudio** la evaluación de estilos de código para lenguajes de programación, teniendo como **campo de acción** la concepción, análisis y diseño de herramientas de evaluación de estilos de código.

Idea a defender: Con la realización del análisis y diseño de una herramienta de evaluación de estilos de código para la asignatura PP1 es posible obtener una entrada a la implementación, que pueda solucionar los problemas detectados en el desarrollo de esta actividad.

Objetivo general Realizar el análisis y diseño de una herramienta de evaluación de estilos de código para la asignatura Práctica Profesional 1 en la Universidad de las Ciencias Informática, que solucione los problemas detectados.

Objetivos específicos:

1. Elaborar el marco teórico de la investigación.
2. Definir funcionalidades y características de la nueva herramienta evaluadora de estilos de código.
3. Desarrollar los artefactos correspondientes al análisis y diseño de la nueva herramienta.

Para dar cumplimiento a los objetivos propuestos se utilizaron varios **métodos de investigación** con el objetivo de proveerla de bases solidas, que tributen a su éxito. Entre los utilizados se encuentran: **Análisis – Síntesis**, que permitió identificar los conceptos y definiciones más importantes relacionados con la evaluación de estilos de código y con la construcción de herramientas para tal fin; además permitió generar luego, una propuesta de análisis y diseño adecuada a la situación planteada y a las tecnologías estudiadas. **Histórico – Lógico,** que permitió realizar un estudio de las diferentes herramientas existentes, sus características y ventajas, así como su evolución. **Modelación,** que sirvió de base para el esbozo de los diferentes diagramas y modelos del proceso de análisis y diseño de la aplicación. También se utilizaron **métodos empíricos** como la **Observación,** pues mediante este se pudo detallar el comportamiento de los sistemas existentes y el método de **Entrevista,** que permitió lograr un acuerdo entre el cliente y el autor, y recopilar toda la información necesaria para el posterior desarrollo del trabajo.

Para dar cumplimiento al objetivo general se desglosan las siguientes **tareas:**

1. Estudiar estándares de códigos existentes en el mundo y en la UCI.
2. Analizar y comparar las herramientas evaluadoras de estilos de codificación existentes.
3. Seleccionar los lenguajes que serán soportados por la herramienta.
4. Seleccionar el lenguaje de programación, las herramientas y la metodología de desarrollo para el análisis y diseño de la herramienta.
5. Definir los requisitos de la herramienta.
6. Realizar las pruebas de concepto necesarias para garantizar el funcionamiento del sistema.
7. Generar los artefactos de análisis y diseño de la herramienta.

El presente trabajo se compone de 3 capítulos. Cada uno de ellos aborda diferentes temas estructurando el contenido. Los capítulos son los siguientes:

Capítulo I. Describe el marco teórico de la investigación. Se realiza el análisis del estado del arte. Se enuncian los principales conceptos relacionados con las herramientas de verificación de estándares de codificación, exponiendo una visión preliminar de las características deseables de dichas herramientas para la propuesta de solución. Además, se justifican las herramientas, lenguaje de modelado y metodología utilizada.

Capítulo II. Describe la propuesta del sistema que facilitará el proceso de enseñanza de estilos de codificación. Se hace referencia al Modelo de Dominio y se identifican los principales conceptos que lo integran. Se describen las características del sistema, en términos de requisitos funcionales y no funcionales y se identifican los casos de uso del sistema.

Capítulo III. Se identifican las clases del análisis y se relacionan, estática y dinámicamente, para lograr una visión más clara del sistema. Se logra una abstracción con mayor nivel de detalle del sistema, con respecto a la plataforma, lenguaje de programación, entre otros elementos. Los artefactos generados correspondientes a la metodología RUP son el Modelo de Análisis y Modelo de Diseño.

Capítulo 1. Marco teórico.

Introducción

En este capítulo se exponen los fundamentos que sustentan la presente investigación, con el objetivo de crear el marco teórico relacionado con los aspectos tratados en el objeto de estudio y campo de acción. Se brinda una panorámica de las principales definiciones y conceptos asociados al problema presentado. Se realiza el análisis de varias herramientas de análisis estático de código a fin de conocer el estado del arte en este campo y se fundamenta la selección del lenguaje de programación, herramientas, y metodología de desarrollo. Como uno de los principales hitos de este capítulo se llega a un listado de las características más importantes de los sistemas analizados con vistas a incorporarlos en la propuesta de solución.

1.1 Lenguaje de programación

Los lenguajes de programación en el ámbito de la informática son notaciones que permiten comunicar a las personas con las maquinas. Estos permiten crear programas formados por sentencias siguiendo reglas de sintaxis que se ponen a disposición del programador para comunicarse con los dispositivos hardware y software existentes (2).

Si bien no se establece ninguna conversación entre dos personas utilizando un lenguaje de programación, se dice que estos permiten la comunicación entre programadores, debido a que el código escrito por uno, debe ser de fácil entendimiento por otros programadores. En los albores de la programación de equipos de cómputo, las instrucciones que se le pasaban a las máquinas eran casi indescifrables para un humano. Actualmente es muy necesario que se comprenda lo que otro programador ha escrito utilizando algún lenguaje de programación de alto nivel, por varias razones: primera, porque cada vez más, el trabajo necesita de equipos de personas, no individuos que realicen un trabajo que solo ellos entienden (3); segundo, las personas que hoy hicieron un programa, quizás no podrán proveer soporte toda la vida, haciéndose necesaria una transferencia de conocimientos; y tercero, ya los lenguajes han evolucionado hasta utilizar estructuras muy cercanas a las del lenguaje humano, sería un desperdicio desaprovechar tales bondades.

1.2 Estilo de codificación

Es el conjunto de reglas o normas usadas para escribir código fuente y que incluye varios aspectos dentro del proceso de codificación (3). También conocido como estándares de código o convenciones de código.

Los estilos de código son convenciones, lo que los convierte en algo subjetivo, que depende de cada lenguaje y de los propios programadores, que se pueden sentir más cómodos utilizando uno u otro estilo.

1.2.1 Importancia de la utilización de estilos de código

La utilización de un buen estilo de código influye positivamente en las labores de mantenimiento de un software como lo son la corrección de errores y el desarrollo de una nueva funcionalidad. Es realmente difícil que un programador mantenga una aplicación a lo largo de toda su vida; por tanto, el cambio de personal contribuye a un profundo estudio del código fuente (3).

Un buen estilo de código debe aportar a la eficiencia del proceso de desarrollo, logrando que los programas sean más robustos y comprensibles. Las cualidades que se ven beneficiadas de forma más directa por un buen estilo son:

Extensibilidad: La facilidad con que se adapta el software a cambios de especificación. Utilizando un buen estilo de código se refleja de forma más clara la relación problema/solución. Esto tiene como efecto que muchos cambios simples en el problema, reflejen de forma obvia los cambios a hacer en el programa (3).

Verificabilidad: La facilidad con que pueden comprobarse propiedades de un sistema. Si el estilo de código hace obvia la estructura del programa, eso ayuda a verificar que el comportamiento sea el esperado (3).

Reparabilidad: La posibilidad de corregir errores sin demasiado esfuerzo gracias a la comprensión rápida del problema (3).

Capacidad de evolución: La capacidad de adaptarse a nuevas necesidades, gracias a lo evidente que se hace la estructura del programa (3).

Comprensibilidad: La facilidad con que el programa puede ser comprendido por varias personas, que pueden componer un equipo de trabajo (3).

1.2.2 Elementos generales que conforman un estilo de codificación

Indentación

Permite ver la estructura jerárquica de la codificación sin tener que observar detalles del código, eso es posible sólo mirando la distribución de espacios en blanco y espacios escritos. En un fragmento de programa cuando se busca algo o se lee rápido, se puede visualizar la distribución de espacios, aunque no se dedique tiempo a observar la estructura, el contenido o la concordancia de paréntesis, llaves, corchetes u otro separador (3).

Notación

Permite definir un estándar de identificación a la hora de escribir los nombres de métodos, variables y clases. Los nombres deben darse con intencionalidad, independientemente de la distribución de símbolos o mayúsculas que se utilicen, por ejemplo cada nombre de variable debe transmitir su función en el código (3).

Algunas Notaciones aceptadas en la docencia en la Universidad de las Ciencias Informáticas se relacionan a continuación:

Notación Camel:

Consiste en escribir los identificadores con la primera letra de cada palabra en mayúsculas y el resto en minúscula: EndOfFile. Existen dos variantes:

- UpperCamelCase, CamelCase o PascalCase: En esta variante la primera letra también es mayúscula.
- lowerCamelCase, camelCase o dromedaryCase: La primera letra es minúscula.

La notación Camel se utiliza también en publicidad y marcas comerciales: *PlayStation*, *easyJet.*

Notación C:

Durante la década del 1960, con la estandarización del código ASCII, los primeros programadores de C y UNIX utilizaron el carácter "_" como separador (3):
end_of_file. Esta notación sigue siendo la más utilizada en C y entornos UNIX, y es utilizada por muchos programadores.

Notación Húngara:

Se basa en Camel, añadiendo al principio del identificador una secuencia de letras en minúscula, que indica alguna característica del identificador, como su tipo en el caso de variables (3). Por ejemplo:

- iValue podría denotar una variable entera.
- fValue una variable float.
- frmPartner la instancia de un formulario.

Otros elementos

Además de los elementos mencionados antes, se toma en cuenta además a la hora de definir un estilo de código los siguientes:

- Cantidad máxima de caracteres en una línea de código, independientemente si esta finaliza o no.
- Espacios en blanco que se destinan a la separación de expresiones y sentencias.
- Estilo de los comentarios, o sea, la explicación de funciones o líneas de códigos dentro del programa. Pueden ser en bloque o solamente una línea explicativa.

1.3 Convenciones de código en la asignatura PP1

A continuación se relacionan los estilos fundamentales de codificación definidos en la UCI para la asignatura Practica Profesional 1:

Convenciones para los nombres: Definen la forma en que se deben nombrar las clases, las interfaces y las variables (4).

- ✓ Los nombres de las clases y de los métodos deben usar la notación Pascal.
- ✓ Las interfaces deben nombrarse con el prefijo “I” siguiendo la notación Camel.
- ✓ Para nombrar las variables y los parámetros se debe usar la notación Camel.

Indentación: Se debe realizar con un tabulado de tamaño 4, además para lograr estos 4 espacios no debe hacerse con la tecla *SPACE* sino con *TAB* (4).

Comentarios: Deben estar al mismo nivel del código y usar “//” o “///”. Se debe evitar usar los comentarios de tipo “/* …*/” (4).

Llaves y líneas en blanco (4)**:**

- ✓ Se debe usar una línea en blanco para separar agrupaciones lógicas del código.
- ✓ Debe dejarse una y solo una línea en blanco entre los métodos de una clase.
- ✓ Las llaves deben ser utilizadas sobre líneas separadas y no sobre la misma línea.
- ✓ Las llaves se deben poner al mismo nivel del código que las contiene.

1.4 Herramientas evaluadoras de estilos de código

El análisis estático de código es el que se le hace a los programas sin ejecutarlos, a diferencia del análisis dinámico que sí verifica el comportamiento del programa en funcionamiento, esta tarea está estrechamente relacionada con ciertas herramientas, también llamadas herramientas de revisión de código. La sofisticación de análisis estático varía desde las herramientas que sólo consideran las características de sentencias individuales hasta las que incluyen el análisis del programa completo con todos sus archivos. El resultado de la aplicación de estos programas se extiende desde el resaltado de las partes con posibles errores en el código, hasta reportes completos del análisis, y en su versión más

compleja se utilizan modelos que prueban matemáticamente las propiedades del código de determinado algoritmo.

El *estilo de código* como muchas de las cualidades del código fuente escrito y como parte del análisis estático de código, también es objeto de comprobación automática. En la actualidad existe un conjunto grande de herramientas, librerías o *plugin* de un entorno de desarrollo que facilita la comprobación de la convención establecida para la codificación. A continuación se listan algunas de las herramientas más conocidas en el mundo y de las cuales se ha hecho uso alguna vez en las clases de PP1.

1.4.1 *StyleCop*

StyleCop 4.3.2 analiza el código fuente escrito en C# para hacer cumplir un conjunto de reglas de estilo, erradicando las violaciones de las mismas mediante la adición de atributos de supresión. Es similar al análisis de código de Visual Studio, aunque más especializada en algunos aspectos. Lo positivo de esta herramienta es que permite su integración con Visual Studio, y que los desarrolladores tengan la posibilidad de definir nuevas normas (5).

Figura 1. *StyleCop*

Las reglas de estilo más importantes que evalúa la herramienta son las siguientes:

Reglas de Diseño: Se verifica un conjunto de requisitos como las llaves, las declaraciones, los métodos de accesos y las líneas en blanco (5).

- ✓ Las llaves y los códigos no deben compartir una misma línea.
- ✓ Las declaraciones no deben ser de una sola línea.
- ✓ Los métodos de accesos (set y get) pueden estar definidos en una misma línea o en varias líneas.
- ✓ No deben existir líneas en blanco ni antes ni después de abrir o cerrar una llave.

Reglas de Mantenimiento: Se recogen una serie de requisitos a cumplir que a continuación se listan:

- ✓ El uso de paréntesis innecesarios.
- ✓ Un archivo solo puede contener una sola clase.
- ✓ Las funciones *Debug.Asset y Debug.Fail* debe mostrar una descripción.

Requisitos para nombres, los tipos y variables: Se verán reflejados en la forma de nombrar las interfaces, las constantes y la utilización de la Notación *Húngara* para las variables:

- ✓ Los nombres de los elementos, interfaces y constantes no deben comienzan con letra mayúscula.
- ✓ Los nombres de las variables no deben ser prefijo, ni comenzar y menos contener un guión bajo.

1.4.2 *Checkstyle*

Checkstyle es una herramienta para ayudar al desarrollo de programas en Java que se deseen ajustar a un estándar de codificación. Es ideal para los proyectos en los que se quiere aplicar un estándar de codificación personalizado. Es altamente configurable y se puede utilizar para verificar casi cualquier estándar (6).

Figura 2. *Checksyle*

Se definen una gran variedad de requisitos para dar cumplimiento a los objetivos de evaluación, de los cuales los más importantes se enunciaran a continuación.

Análisis de bloques: Se evalúa la estructura del código a partir de los separadores utilizados, y debe estar correctamente indentado el bloque:

- ✓ El texto tiene que estar dentro de los bloques.
- ✓ Exige que haya una declaración dentro de los bloques.

En la inicialización de un arreglo se debe analizar que exista una coma así como se muestra en el siguiente fragmento de código:

```
int [] a=new int []
 {
  1,
  2,
  3
 }
```

Convenciones para los nombres: Se validan mediante una expresión regular, se comprueba que los identificadores miembros comiencen con una “m” seguido de una letra mayúscula.

Espacios en blancos: Se evalúa la correcta utilización de los espacios en blanco para lograr mejor legibilidad del código.

- ✓ Los espacios alrededor de los símbolos < y >.
- ✓ Se requiere un espacio después de un inicializador (;).

1.4.3 PMD

PMD analiza el código escrito en Java buscando problemas potenciales como (7):

- Bloques *try/catch/finally/switch* vacíos.
- Código muerto, o sea, variables y métodos privados no utilizados.
- Código fuente no óptimo, o sea, uso excesivo de clases como *String/StringBuffer*, entre otras.
- Expresiones complicadas innecesariamente, o sea clausulas ***if, for*** *y* ***while*** anidadas o escritas de forma poco legible.
- Código duplicado, esto potencia la reutilización de código, dentro del propio proyecto.

Figura 3. *PMD*

PMD viene integrado con herramientas como *JDeveloper*, *Eclipse*, *JEdit*, *JBuilder*, *BlueJ*, *CodeGuide*, *NetBeans/Sun* Java *Studio Enterprise/Creator*, *IntelliJ IDEA*, *TextPad*, *Maven*, *Ant, Gel, JCreator*, y *Emacs* en forma de plugins (7).

1.4.4 *FxCop*

Es una herramienta libre, de análisis de código estático de Microsoft que chequea el código CSharp en conformidad con los estándares recomendados por los fabricantes, un subconjunto de los estándares que propone la tecnología *.NET* (8).

FxCop analiza el código fuente, permitiendo validar contra un conjunto de reglas predefinidas. Las reglas están clasificadas en las siguientes categorías:

- Documentación, o sea, los distintos tipos de comentarios.
- Vista, referido a la legibilidad del código.
- Notación, nombres de variables, métodos sugerentes y siguiendo los estándares comunes.
- Espaciamiento, referido al tabulado y la indentación.

FoxCop incluye una versión con interfaz de usuario gráfica y una versión accesible desde la línea de comandos del sistema operativo. Es posible crear reglas para ser usadas posteriormente(8).

FoxCop fue liberado como software libre en abril del 2010.

1.5 Aportes del estudio de las herramientas evaluadoras a la propuesta de solución

Las herramientas analizadas evalúan fragmentos de código mediante reglas definidas, evitan las violaciones de las mismas y mejoran la forma de programar de los desarrolladores. El nivel de detalle que contempla cada una de ellas es muy alto, debido a que son dedicadas a este fin solamente.

La nueva herramienta evaluadora de estilos de código debe tener un subconjunto de las funcionalidades brindadas por las herramientas estudiadas. A continuación se delimitan cuáles son las funcionalidades que se toman en cuenta a la hora de analizar y diseñar la aplicación:

- Poner a disposición de usuario los estilos más conocidos para chequear el código según los mismos.
- Configurar los estilos de código a utilizar haciendo personalizables los valores de los elementos fundamentales.
- Determinar que parámetros se miden en cada evaluación, o sea, tomar, o no, en cuenta determinado parámetro.
- Evaluar el código fuente según un estilo de código.

1.6 Proceso de desarrollo

En este apartado se listan y analizan metodologías, herramientas y lenguajes con el fin de escoger el entorno en el cual se va a desarrollar el sistema, escogiendo un grupo de estos para la propuesta de solución. La definición de estos elementos garantiza la ejecución planificada de las tareas en el desarrollo de un software, además, de brindar una documentación extensiva y de mucha utilidad de los elementos que lo componen.

1.6.1 Metodologías de desarrollo de Software

Una metodología de desarrollo no es más que un conjunto de pasos y procedimientos que deben seguirse para el desarrollo de un software (9). Las metodologías tomadas en cuenta para la investigación son dos de las más conocidas y de carácter totalmente opuesto: Extreme Programming (XP) y el Proceso Unificado de Desarrollo (RUP).

Extreme Programming(XP)

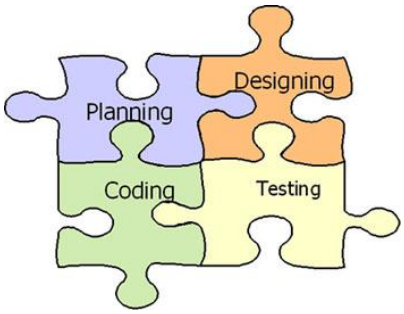


Figura 4. Metodología *XP*

XP es una metodología ágil que potencia las relaciones interpersonales, promueve el trabajo en equipo, y contempla como una parte importante el aprendizaje de los desarrolladores. Se basa en la retroalimentación continua entre el cliente y el equipo de desarrollo y promueve la simplicidad en las soluciones implementadas. Es adecuada especialmente para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico. Se centra principalmente en el desarrollo del software, dándole una menor importancia a la generación de la documentación que implica la creación de una aplicación informática (10).

Proceso Unificado de Desarrollo de Software (RUP)

RUP es el resultado de varios años de desarrollo y uso práctico en el que se han unificado diferentes técnicas, a través del UML, su lenguaje de modelado. La versión que se ha estandarizado surgió en 1998 y se conoció en sus inicios como Proceso Unificado de Rational; de ahí las siglas con las que se identifica a este proceso de desarrollo (11).

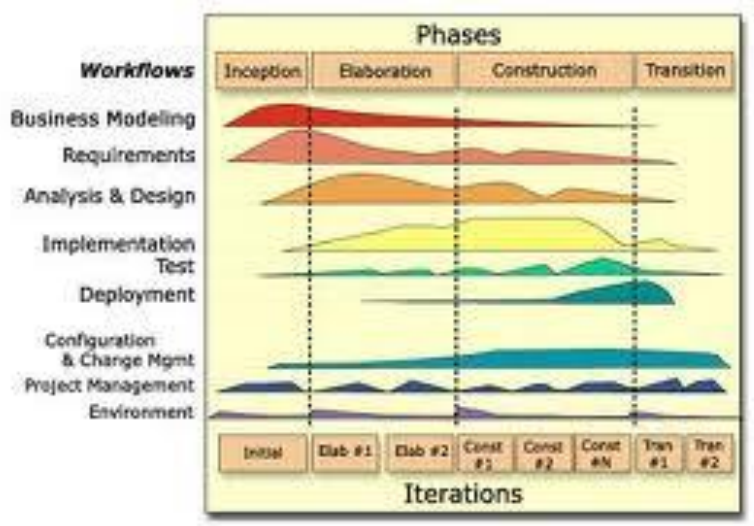


Figura 5. Metodología *RUP*

Características de RUP:

- ✓ Dirigido por Casos de Uso:

Los Casos de Uso son una técnica de captura de requisitos que fuerza a pensar en términos de importancia para el usuario y no sólo en funciones que sería bueno contemplar. Se define un Caso de Uso como un fragmento de funcionalidad del sistema que proporciona al usuario un valor añadido. Los Casos de Uso representan los requisitos funcionales del sistema.

En RUP los Casos de Uso no son sólo una herramienta para especificar los requisitos del sistema, también guían su diseño, implementación y prueba. Los Casos de Uso no sólo inician el proceso de desarrollo sino que proporcionan un hilo conductor, permitiendo establecer trazabilidad entre los artefactos que son generados en las diferentes actividades del proceso de desarrollo (11).

- ✓ Centrado en la arquitectura:

La arquitectura de un sistema es la organización o estructura de sus partes más relevantes, lo que permite tener una visión común a todos los involucrados (desarrolladores y usuarios) y una perspectiva clara del sistema completo, necesaria para controlar el desarrollo.

- ✓ Iterativo e Incremental:

El equilibrio correcto entre los Casos de Uso y la arquitectura es vital en el desarrollo del producto, lo cual se consigue con el tiempo, para esto, la estrategia que propone RUP es tener un proceso iterativo e incremental donde el trabajo se divide en partes más pequeñas o minis proyectos. Cada mini proyecto se

puede ver como una iteración (un recorrido más o menos completo a lo largo de todos los flujos de trabajo fundamentales) del cual se obtiene un incremento que produce un crecimiento en el producto.

Selección de la Metodología.

Una vez mencionadas las principales características de dos metodologías con filosofías totalmente diferentes, se decide utilizar la metodología robusta: RUP. La razón fundamental que conlleva a esta decisión es que el proceso de construcción del sistema estará dividido en dos partes, separadas por largo tiempo, esto convierte a RUP en un candidato muy deseable debido a que genera documentación mucho más formal y detallada (9). En su contra, muchas personas defienden que es una metodología para grandes proyectos, y grandes equipos de desarrollo, pero la realidad es que es muy adaptable a cualquier tipo de proyecto.

1.6.2 Herramientas de Modelado

Visual Paradigm

Figura 6. *Visual Paradigm*

La herramienta soporta el ciclo completo de desarrollo de software: análisis y diseño, implementación, pruebas y despliegue. Es muy eficiente, posibilita la construcción de aplicaciones de forma rápida y con buena calidad. Utiliza UML como lenguaje de modelado, además, posibilita la integración con sistemas de control de versiones que almacenan centralmente los artefactos, realizan un seguimiento de los cambios realizados en software y brindan un enlace seguro entre los desarrolladores del sistema haciendo el trabajo más fácil y dinámico (12).

Rational Rose

Rational Rose Enterprise Edition es una herramienta desarrollada por *Rational Corporation* basada en el Lenguaje Unificado de Modelación (UML), que permite crear los diagramas que se van generando durante el proceso de Ingeniería en el Desarrollo del Software (13).

En la definición de sistemas, esta herramienta permite que el equipo de desarrollo identifique las necesidades del cliente de forma efectiva y comunique la solución propuesta de forma clara (13). Rational permite completar una gran parte de las disciplinas (flujos fundamentales) de RUP tales como:

- ✓ Captura de requisitos.
- ✓ Análisis y diseño.
- ✓ Implementación.
- ✓ Control de cambios y gestión de configuración.

Características principales:

Entre las características principales:

- ✓ Admite como notaciones: *UML, OMT y Booch*.
- ✓ Permite desarrollo multiusuario.
- ✓ Genera documentación del sistema.

Selección de la herramienta de modelado

Después de haber realizado el estudio de ambas herramientas de modelado se llega a la conclusión de que las dos son bastante utilizadas por las ventajas que brindan, no obstante, se propone *Visual Paradigm* en su versión para *UML* 6.4 *Enterprise Edition* debido a que la empresa comercializadora de la UCI, Albet posee licencia de soporte y actualizaciones para la utilización de la misma y también por ser una herramienta libre, y estar acorde con las nuevas estrategias de la universidad para la migración a software libre.

1.6.3 Lenguajes de programación

Java

Java es un lenguaje de programación creado por *SUN Microsystems* muy parecido al estilo de programación del lenguaje “C” y basado en la programación orientada a objeto.

Figura 7. *Java*

Con la programación en Java, se pueden realizar programas como los *applets*, que son aplicaciones especiales, que se ejecutan dentro de un navegador al ser cargada una página HTML en un servidor Web. Por lo general los *applets* son programas pequeños y de propósitos específicos.

Otra de las utilidades de la programación en Java es el desarrollo de aplicaciones, que son programas que se ejecutan en forma independiente. Java permite la modularidad por lo que se pueden hacer rutinas individuales que sean usadas por más de una aplicación, por ejemplo una rutina de impresión que puede servir tanto para el procesador de palabras, como para la hoja de cálculo (14).

Java, permite el desarrollo de aplicaciones bajo el esquema de Cliente-Servidor, y de aplicaciones distribuidas, lo que es capaz de conectar dos o más computadoras u ordenadores, ejecutando tareas simultáneamente, y de esta forma logra distribuir el procesamiento (14).

C#

Es un lenguaje de programación orientado a objetos desarrollado por *Microsoft* y estandarizado, como parte de su plataforma *.NET*. Su sintaxis básica deriva de C/C++ y utiliza el modelo de objetos de la plataforma *.NET* el cual es similar al de Java aunque incluye mejoras derivadas de otros lenguajes. C#, pronunciado C Sharp, fue diseñado para combinar el control a bajo nivel de lenguajes como C y la velocidad de programación de lenguajes como *Visual Basic* (15).

C# es actualmente uno de los lenguajes de programación más populares en informática. El objetivo de *Microsoft*, que tiene todos los derechos sobre la plataforma de desarrollo *.NET Framework SDK* en la que está incluido C#, es permitir a los programadores abordar el desarrollo de aplicaciones complejas con facilidad y rapidez. Es como si se tomaran todas las cosas buenas de *Visual Basic* y se añadieran a C++, aunque recortando algunas de las tradiciones más ocultas y difíciles de conocer de C y C++ (15).

Selección del lenguaje de programación

Para seleccionar el lenguaje de programación se tomaron en cuenta los siguientes aspectos: si es *software* libre o propietario, esto se debe a que para Java existen potentes Entornos de desarrollo Integrado (*IDE*) que son libres y C# posee algunos que todavía están empezando a desarrollarse como el *MonoDevelop*. Otro factor es que las aplicaciones desarrolladas en Java pueden ejecutarse en diferentes sistemas operativos siempre que estos tengan la máquina virtual instalada, y las que están desarrolladas en C# no poseen esta característica. Por estas razones se ha elegido Java como el lenguaje para la implementación del software.

1.6.4 Sugerencias para el desarrollo de la aplicación

Cualquiera sea el lenguaje que se utilice, es necesario contar con medios para que los desarrolladores puedan editar el código, compilar, y ejecutar programas con mayor facilidad, es ahí donde radica la importancia de un *IDE*. Java es un lenguaje que posee una larga oferta de *IDE*, entre los que se encuentran: *NetBeans*, *Eclipse*, *JBuilder* y otros.

Seleccionado el lenguaje de programación para el proceso de desarrollo de la aplicación se propone el uso de los *IDEs NetBeans* o *Eclipse*, ambos son multiplataforma y muy potentes. Como se verá más adelante la aplicación que se propone es Web, por lo que debe mencionarse además que ha de seleccionarse un servidor Web, para esta decisión, se recomienda investigar *Apache* y *JBoss*.

Conclusiones

El análisis de los elementos teóricos necesarios para definir las principales características de la aplicación a partir de sistemas conocidos, garantiza un punto de partida para la propuesta de la solución. Después del estudio de las tecnologías y herramientas más utilizadas en el campo de la informática para el desarrollo de aplicaciones es posible seleccionar cuáles de ellas usar, con el fin de crear el entorno apropiado para la elaboración del sistema. De esta manera se asegura la creación de una herramienta en menor tiempo y una mayor calidad posible.

Capítulo 2. Descripción del sistema.

Introducción

En el presente capítulo se describe la propuesta de un sistema enfocado a facilitar el trabajo independiente de los estudiantes y agilizar el proceso de enseñanza aprendizaje guiado por el profesor de PP1 en el tema de estilos de código. Se hace referencia al Modelo de Dominio con los principales conceptos que lo integran. Se exponen las características del sistema, se describe cómo este debe funcionar, se presentan los requisitos del sistema, tanto Funcionales como No Funcionales que darán solución a los problemas existentes y se muestran quiénes son los actores y los casos de uso del sistema.

2.1 Objeto de automatización

El Departamento Central de Práctica Profesional de la UCI, a partir de la retroalimentación por parte de los profesores de sus clases, determinó que había que agilizar el proceso de enseñanza-aprendizaje de estilos de codificación dentro de las clases. Los estudiantes deberían tener la forma de practicar lo aprendido, cuestión un tanto difícil en las condiciones actuales y con las herramientas existentes, tan variadas como dependientes de la plataforma. Los elementos a automatizar no quedaron claros desde un principio, pero la necesidad fundamental detectada fue que el estudiante como el profesor debía tener la oportunidad de verificar los errores de estilo de codificación, en un entorno cómodamente accesible, fácil de utilizar, independiente de la plataforma y que se pudiera evaluar el estilo para el lenguaje escogido por el usuario.

2.2 Propuesta de sistema

Debido a las características deseadas que se mencionaron previamente, se propone una aplicación Web, para garantizar la accesibilidad y la independencia de la plataforma, cuya interfaz sea sencilla y navegable intuitivamente. Dicha aplicación debe estar disponible en un servidor central acorde con el modelo Cliente–Servidor, con acceso al servicio de directorio disponible en la UCI, a través del protocolo LDAP (Protocolo ligero de acceso a directorio). Este requisito se sostiene en la necesidad de definir diferentes roles, para varios tipos de usuario. Para la implementación del sistema se determinó crear una aplicación Web que utilice herramientas libres existentes de forma transparente al usuario, sirviendo de puente entre éste y las funcionalidades expuestas por los software seleccionados del conjunto que se estudió. Esta decisión se tomó debido a dos razones fundamentales: la primera es que ya existen estas herramientas con años de desarrollo y probada efectividad, no es necesario implementar el análisis de

estilos de código nuevamente si estas cumplen con los requisitos que se necesitan para la herramienta de apoyo a la docencia que se desea implementar, la otra razón es que la propuesta sería una solución escalable al poder incrementar el número de lenguajes soportados paulatinamente, o hacer cambios de lenguaje provocados por cambios de planes de estudio.

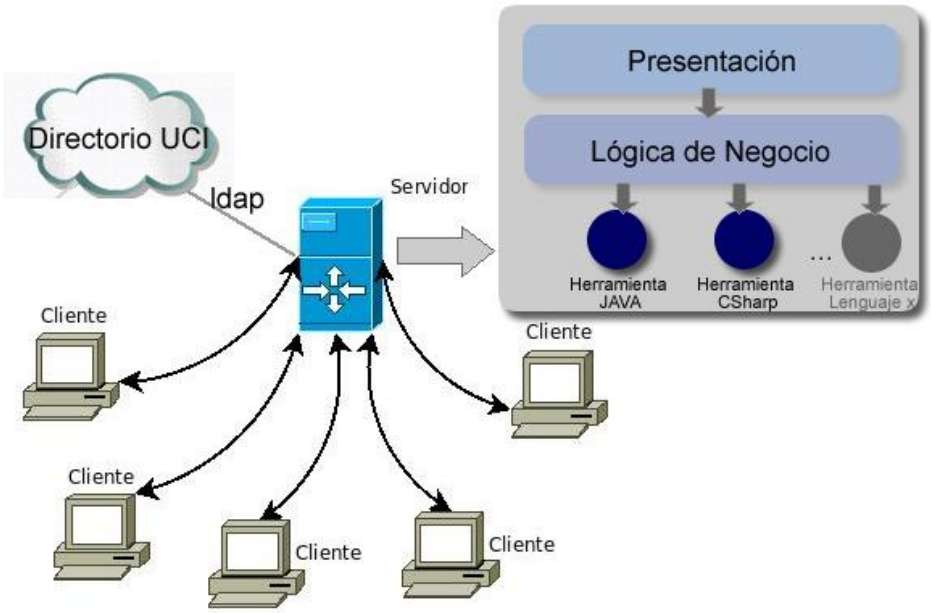


Figura 8. Modelo de alto nivel de la propuesta de solución.

2.3 Pruebas de concepto de las herramientas seleccionadas.

Las herramientas seleccionadas fueron FxCop para el lenguaje CSharp y Checkstyle para Java, estas herramientas del estudio realizado cuentan con la mayor cantidad de funcionalidades y además tienen una característica que las convierten en candidatas perfectas, que ambas tienen acceso al núcleo de las funcionalidades a través de la línea de comandos que a su vez pueden ser fácilmente accedida desde cualquier lenguaje de alto nivel, como Java, lenguaje de implementación de la herramienta.

Antes de decidir si esta solución era posible se le hicieron un conjunto de pruebas a las herramientas, en su integración con Java, con el fin de determinar si era técnicamente factible utilizar las herramientas existentes. Para valorar el resultado de estas pruebas que se tomó en cuenta que las herramientas admitieran la configuración por archivos de configuración o por comandos, que aceptara más de una configuración por instancia del software, y que utilizara para su configuración archivos en un formato procesable sin tanta dificultad, todo esto se comprobó una vez que se tuviera la certeza de que las herramientas son accesibles desde el código Java.

Los resultados en general arrojaron que ambas herramientas cumplen con los requisitos necesarios y que son accesibles desde Java y el resultado de la evaluación se puede obtener apropiándose de la salida estándar.

2.4 Modelo de Dominio

Debido a la poca estructuración de los procesos de negocio, o a la poca claridad que se tenían de ellos por la variedad de herramientas utilizadas actualmente, se plantea un Modelo de Dominio, con el objetivo de ayudar a comprender los conceptos que utilizan los usuarios, que serán los mismos a utilizar para el desarrollo de la aplicación. Este artefacto de RUP, es un Diagrama de Clases de UML, que define los principales conceptos del dominio y permite representar un diagrama con los eventos reales relacionados a este. Algunos de estos objetos pueden ser: conceptos, atributos, asociaciones y roles. A continuación se identifican los principales conceptos que se utilizarán en el diagrama del Modelo Dominio:

Estudiante: Persona responsable de los contenidos para el aprendizaje de las asignaturas que cursa los estudios en la UCI en primer año de la carrera.

Profesor: Persona encargada de tutorar al estudiante a configurar y utilizar las herramientas evaluadoras definidas para la asignatura.

Jefe de la asignatura PP del Departamento Central: Persona responsable de la definición de los estilos de código a utilizar a nivel de universidad en la parte docente.

Jefe de la asignatura PP1 de facultad: Persona responsable de la definición de los estilos de codificación a utilizar a nivel de facultad.

Práctica Profesional: Actividad formativa, se desempeña a lo largo del proceso de enseñanza aprendizaje del estudiante durante los cinco años de la carrera. Esta actividad está comprendida por cinco asignaturas de carácter anual.

Práctica Profesional 1: Forma parte de la actividad formativa, es una asignatura que se imparte en primer año de la carrera informática en la UCI.

Herramienta Evaluadora: Herramientas actuales utilizadas en la asignatura.

Lenguaje de programación: Lenguaje en el que se escribe un código fuente que más tarde será verificado, según su estilo.

Estilo de codificación: Es el conjunto de reglas o normas usadas para escribir código fuente.

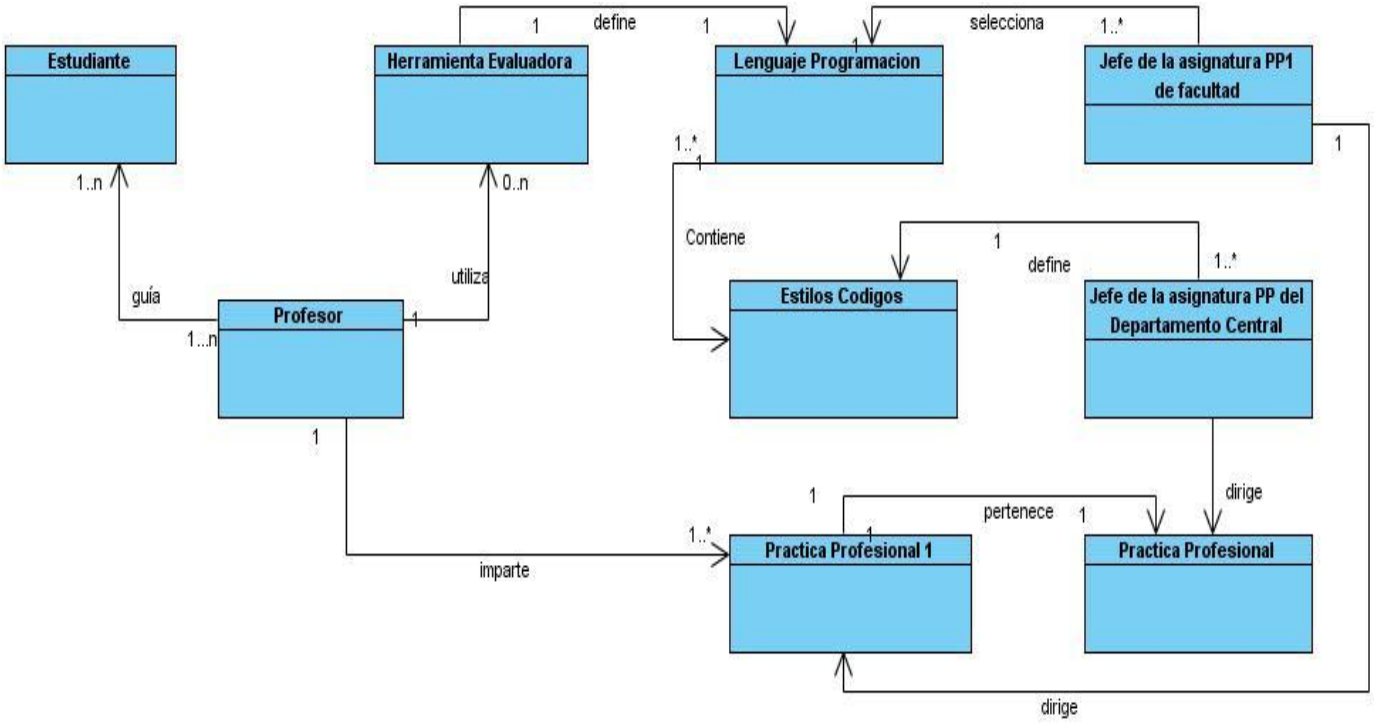


Figura 9. Modelo de dominio

2.5 Especificación de los requisitos de software

La especificación de los requisitos es uno de los flujos de trabajo más importantes, mediante él se establece qué tiene que hacer exactamente el sistema. Los requisitos especificados son el contrato que se debe cumplir la aplicación y que el usuario final debe entender y aceptar (11).

2.5.1 Requisitos Funcionales

RF1. Autenticar usuarios.

RF2. Permitir la selección del lenguaje de programación.

RF3. Listar estilos de codificación.

RF4. Selección del estilo de código.

RF5. Escoger elementos de un estilo de código a evaluar.

RF6. Evaluar un código fuente según un estilo de código.

RF7. Reportar el estado de errores del código evaluado para que cumpla con un estilo determinado.

RF8. Eliminar un estilo de código.

RF9. Modificar un estilo de código.

RF10. Crear un estilo de código.

RF11. Asignar privilegio.

RF12. Desasociar privilegio.

RF13. Listar usuarios con privilegios.

2.5.2 Requisitos No Funcionales.

Los requisitos No Funcionales responden a características que debe tener en sistema, a diferencia de los funcionales que corresponden a funcionalidades, estos se clasifican en una gran cantidad de grupos, de los cuales a continuación se muestran los más importantes para la aplicación a implementar.

Usabilidad

La usabilidad es la capacidad con la que pueda ser aprendida y operada la aplicación por los usuarios a los que va destinada, en este caso a los estudiantes y profesores de la UCI. Para su correcto uso los usuarios deben tener conocimientos básicos de navegación solamente pues la interfaz será sencilla y con pocas interfaces que navegar.

Rendimiento

La aplicación Web debe tener un tiempo de respuesta rápido. Después de instalada la aplicación, la misma debe ser capaz de soportar gran cantidad de usuarios conectados simultáneamente, por lo cual se recomienda que se analice como un riesgo arquitectónico a la hora de implementar.

Soporte

Se requiere que la aplicación tenga mantenimiento ante cualquier fallo que pueda ocurrir. La aplicación Web tiene solo un rasgo que puede dificultar su mantenimiento y es el acceso al directorio de usuarios UCI, que se realizará consumiendo un servicio Web. El resto lo constituiría el mantenimiento del servidor escogido y la instalación de un navegador en las maquinas clientes.

Confiabilidad

Todas las respuestas de salida del sistema tienen que tener un cien por ciento de precisión y veracidad. Las reglas sobre las que se evaluará el código, estarán protegidas del acceso no autorizado, solo el personal acreditado podrá gestionar las reglas.

Seguridad

La aplicación permitirá el uso de un usuario anónimo para los entornos en los que no haya conectividad con el servicio de autenticación, intentando respetar el desenvolvimiento de la clase por encima de todo. Las funcionalidades administrativas y de gestión de estilos y configuraciones solo serán accesibles cuando esté disponible el mencionado servicio.

Interfaz

El diseño de la interfaz debe ser sencillo y amigable. No debe estar cargado de imágenes, para acelerar las respuestas del sistema. La interfaz solo debe mostrar las funcionalidades del rol de usuario que esté utilizando la aplicación. El software debe garantizar una claridad y correcta organización de la información para permitir una correcta interpretación.

Ayuda y documentación en línea

La aplicación debe tener incluida la documentación necesaria para que los usuarios puedan, sin asistencia ninguna, utilizar las funcionalidades del sistema.

2.6 Modelo de Casos de Uso

Está compuesto por todos los casos de uso que guían al proceso de desarrollo (diseño, implementación y prueba). El Modelo de Casos de Uso representa todos los requisitos que dan solución a los problemas presentes. Ayuda a los que están vinculados con el sistema ya sea el cliente, los usuarios o los desarrolladores a llegar un acuerdo sobre cómo utilizar el sistema (11).

2.6.1 Determinación y justificación de los actores

Actor	Descripción
Administrador	Persona que Administra el sistema en caso de cambio de estilo, etc.
Jefe Equipo	Persona encargada de administrar estilo. Esta persona puede ser jefe de proyecto como jefe de departamento.

Usuario	Estudiante o profesor, beneficiarios fundamentales del sistema.

Tabla 1. Descripción de los actores del sistema

2.6.2 Diagrama de Casos de Uso

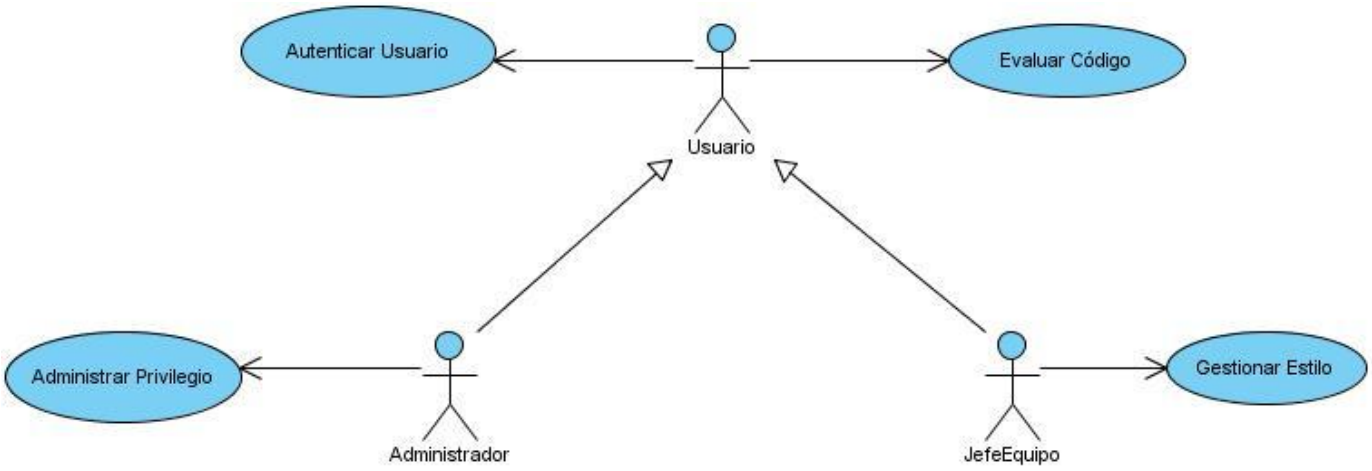


Figura 10. Diagrama de Caso de Uso del Sistema

2.6.3 Descripciones de Casos de Usos del Sistema

Caso de Uso	**Autenticar**
Actores	Usuario
Resumen	El caso de uso inicia cuando el usuario (estudiante, profesor, jefe de equipo, jefe de departamento central y administrador) se autentica en el sistema. El sistema le brinda los privilegios definido a cada usuario.
Referencias	RF1
Precondiciones	
Flujo normal de eventos	
Acción del Actor	**Acción del Sistema**

1. Introduce su identificador y su contraseña en el sistema para acceder a las funcionalidades.	2. El sistema valida la entrada de los datos y muestra su interfaz principal.
Flujos alternos	
	2.1 El sistema muestra un mensaje de error al usuario “Introduzca sus datos”.
Pos-condiciones	El usuario accederá a las funcionalidades definidas.

Tabla 2. Descripción del Caso de Uso del Sistema “Autenticar”

Caso de Uso	**Evaluar código fuente**
Actores	Usuario
Descripción	El caso de uso inicia cuando el estudiante o el profesor introducen un código para evaluar. Opcionalmente puede seleccionar un conjunto de requisitos a evaluar.
Referencias	RF2, RF3, RF4, RF5, RF6, RF7
Precondiciones	El usuario debe estar registrado en el sistema.
Flujo normal de eventos	
Acción del Actor	**Acción del Sistema**
1. El estudiante selecciona en el menú la opción en la página principal “Evaluar código”.	2. El sistema muestra una nueva interfaz para la evaluación de códigos permitiendo: • Introducir el código fuente a evaluar. • Seleccionar el estilo de codificación a utilizar. • Opcionalmente escoge los parámetros del estilo de codificación:

	○ Notación ○ Tamaño de tabulado ○ Tipo de comentarios ○ Espacios de la indentación Y permite: • Evaluar • Cancelar
3. El estudiante introduce el código fuente y selecciona el estilo de codificación.	
4. Selecciona la opción de evaluar.	5. El sistema registra la configuración realizada por el estudiante.
	6. Ejecuta las operaciones de chequeo y muestra el resultado de la evaluación.
Flujos alternos	
4.1 Selecciona la opción de cancelar.	4.2 Vuelve a la página principal.
Pos-condiciones	Se evaluó el código según las reglas activadas exitosamente.

Tabla 3. Descripción del Caso de Uso del Sistema “Evaluar código fuente”

Caso de Uso	**Gestionar Estilo**
Actores	Jefe de Equipo
Descripción	El caso de uso inicia cuando el Jefe de Equipo decide crear un nuevo estilo si no existe previamente. También permite eliminar, modificar los estilos de codificación que han sido definidos por él mismo y consultarlos todos.
Referencias	RF3, RF8, RF9, RF10

<table>
<tr><td>Precondiciones</td><td colspan="2">El Jefe de Equipo debe estar autenticado previamente y tener los roles asignados.</td></tr>
<tr><td colspan="3">Flujo normal de eventos</td></tr>
<tr><td>Acción del Actor</td><td colspan="2">Acción del Sistema</td></tr>
<tr><td>1. Selecciona la opción de “Gestionar estilos” en la página principal.</td><td colspan="2">2. Muestra una nueva interfaz con un listado de todos los estilos creados, permitiendo realizar las siguientes acciones:
Sección 1: Crear estilo
Sección 2: Modificar estilo
Sección 3: Eliminar estilo</td></tr>
<tr><td>3. El Jefe de Equipo selecciona una de las tres opciones mostradas en la aplicación.</td><td colspan="2"></td></tr>
<tr><td colspan="3">Sección 1 “Crear estilo” Flujo normal de eventos</td></tr>
<tr><td>Acción del Actor</td><td colspan="2">Acción del sistema</td></tr>
<tr><td></td><td colspan="2">1. Muestra los parámetros que conforman un estilo de codificación, permite seleccionar e introducir:
• Notación
• Tamaño del tabulado
• Tipo de comentarios
• Espacios de la indentación
Y permite:
• Incluir
• Cancelar</td></tr>
</table>

2. Selecciona o introduce los valores del nuevo estilo.	
3. Selecciona la opción Incluir.	4. El sistema valida la entrada de los datos y se crea el nuevo estilo.
Flujos alternos	
3.1 Selecciona la opción cancelar.	3.2 Regresa a la interfaz principal del caso de uso.
Sección 3 "Modificar estilo" Flujo normal de eventos	
Acción del Actor	**Acción del sistema**
	1. Muestra los valores de los parámetros que conforman el estilo de codificación a modificar, permite seleccionar e introducir: • Notación • Tamaño del tabulado • Tipo de comentarios • Espacios de la indentación Y permite: • Actualizar • Cancelar
2. Selecciona o introduce los valores que quiere cambiar del estilo.	
3. Selecciona la opción Actualizar.	4. El sistema valida la entrada de los datos y se actualiza el estilo.

	5. Se actualiza el listado de estilos existentes.
Sección 4 "Eliminar estilo" Flujo normal de eventos	
Acción del Actor	**Acción del sistema**
1. Selecciona el estilo a eliminar.	2. Elimina el estilo seleccionado.
	3. Se actualiza el listado de estilos existentes.
Flujos alternos	
Pos-condiciones	Se listan estilos, se crea, modifica o elimina un estilo.

Tabla 4. Descripción del Caso de Uso del Sistema "Gestionar Estilo"

Caso de Uso	**Administrar privilegios**
Actores	Administrador
Descripción	El caso de uso inicia cuando el Administrador necesita modificar los permisos de un usuario.
Referencias	RF11, RF12, RF13
Precondiciones	El Administrador debe estar autenticado previamente.
Flujo normal de eventos	
Acción del Actor	**Acción del Sistema**
1. Selecciona la opción de "Administrar privilegios" en la página principal.	2. Muestra una nueva interfaz con un listado de todos los usuarios de dominio que tienen permisos de administrador o de jefe de equipo, permitiendo realizar las siguientes acciones:

	Sección 1: Crear usuario **Sección 2:** Modificar usuario **Sección 3:** Eliminar un usuario
3. Selecciona una de las tres opciones mostradas en la aplicación.	
Sección 1 "Crear usuario" Flujo normal de eventos	
Acción del Actor	**Acción del sistema**
	1. Permite introducir el nombre de usuario de dominio y seleccionar opcionalmente si es administrador, jefe de equipo o ambos. Y permite: • Incluir • Cancelar
2. Selecciona o introduce los valores del nuevo estilo.	
3. Selecciona la opción Incluir.	4. El sistema valida la entrada de los datos y se crea el nuevo usuario con los permisos correspondientes.
Flujos alternos	
3.1 Selecciona la opción Cancelar.	3.2 Regresa a la interfaz principal del caso de uso.
Sección 3 "Modificar usuario" Flujo normal de eventos	
Acción del Actor	**Acción del sistema**

	1. Muestra los valores de los permisos del usuario a modificar, permite marcar o desmarcar los permisos correspondientes: administrador y jefe de equipo Y permite: • Actualizar • Cancelar
2. Selecciona o introduce los valores que quiere cambiar del usuario.	
3. Selecciona la opción Actualizar.	4. El sistema valida la entrada de los datos y se actualiza el usuario.
	5. Se actualiza el listado de usuarios existentes.
Flujos alternos	
3.1 Selecciona la opción Cancelar.	3.2 Regresa a la interfaz principal del caso de uso.
Sección 4 "Eliminar usuario" Flujo normal de eventos	
Acción del Actor	**Acción del sistema**
1. Selecciona el usuario a eliminar.	2. Elimina el usuario seleccionado.
	3. Se actualiza el listado de usuarios existentes.
Flujos alternos	
Pos-condiciones	Se listan usuarios, se crea, modifica o elimina un estilo.

Tabla 5. Descripción del Caso de Uso del Sistema "Administrar privilegios"

Conclusiones

Los profesores emplean diversas vías de demostración en el tema de los diferentes estilos de código, unos utilizan herramientas que son difíciles de manipular y los demás simplemente se apoyan en contenidos elaborados. Por la poca claridad de los procesos del negocio, se realizó el Modelo de Dominio. Se definieron los requisitos de automatización y la propuesta de solución a nivel general expuesta permite dar una visión de lo que se quiere lograr más claramente. La formalización de los requisitos Funcionales y No Funcionales como artefactos de RUP, y de actores y casos de uso del sistema, con su correspondiente descripción, permite una entrada más detallada y clara a los flujos siguientes: Análisis y Diseño.

Capítulo 3. Análisis y diseño del sistema.

Introducción

En el presente capítulo se exponen los resultados fundamentales de los flujos de análisis y diseño. Se muestran todos los diagramas de clases de análisis donde se tratan las clases interfaz, las controladoras, las entidades y las relaciones entre ellas. Se presentan los diagramas de diseño donde se utilizan los estereotipos como son las clases servidoras, cliente, formulario y las clases que permitirá la conexión con las herramientas externas para modelar la aplicación Web.

3.1. Modelo de Análisis

Para la construcción del Modelo de Análisis se identifican las clases que interactúan en la realización de los casos de uso, definidas bajo los estereotipos: clase de interfaz, clase controladora y clase entidad. El objetivo de este modelo es refinar y estructurar los requisitos obtenidos en el flujo anterior, profundizando en el dominio de la aplicación y precisando cómo es que se implementará la solución, sin un alto nivel de detalle (16).

3.1.1 Diagrama de clases del análisis

El diagrama de clases del análisis proporciona una visión estructural de las clases que componen la solución, en este se plasman, por cada Caso de Uso los actores y las clases que interactúan para lograr las funcionalidades deseadas, así como sus relaciones (16).

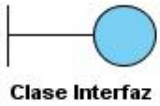
Clase	Estereotipo	Función
Interfaz		Modelan la interacción: Actor-Sistema. Ejemplos: Ventanas, Formularios, comunicación con otros sistemas o dispositivos.
Controladora		Coordinan el trabajo de las clases. Encapsulan comportamiento de un CU. Funciones complejas.
Entidad		Modelan la información del Sistema. Modelan el comportamiento asociado a una información.

Tabla 6. Estereotipos del Modelo de Diseño

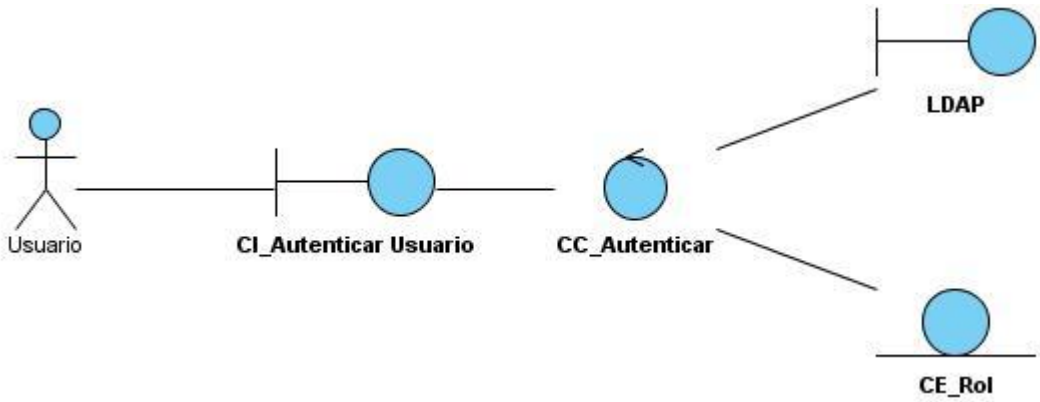


Figura 11. Diagrama de clases del Análisis del Caso de Uso “Autenticar”

Figura 12. Diagrama de clases del Análisis del Caso de Uso “Evaluar Código”

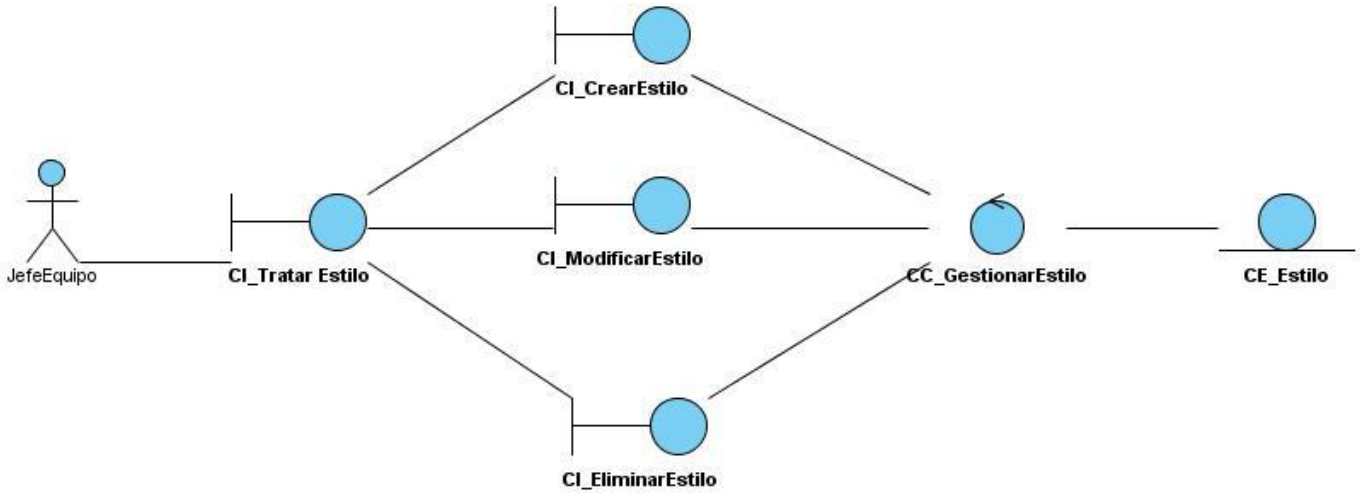


Figura 13. Diagrama de clases del Análisis del Caso de Uso “Gestionar Estilo”

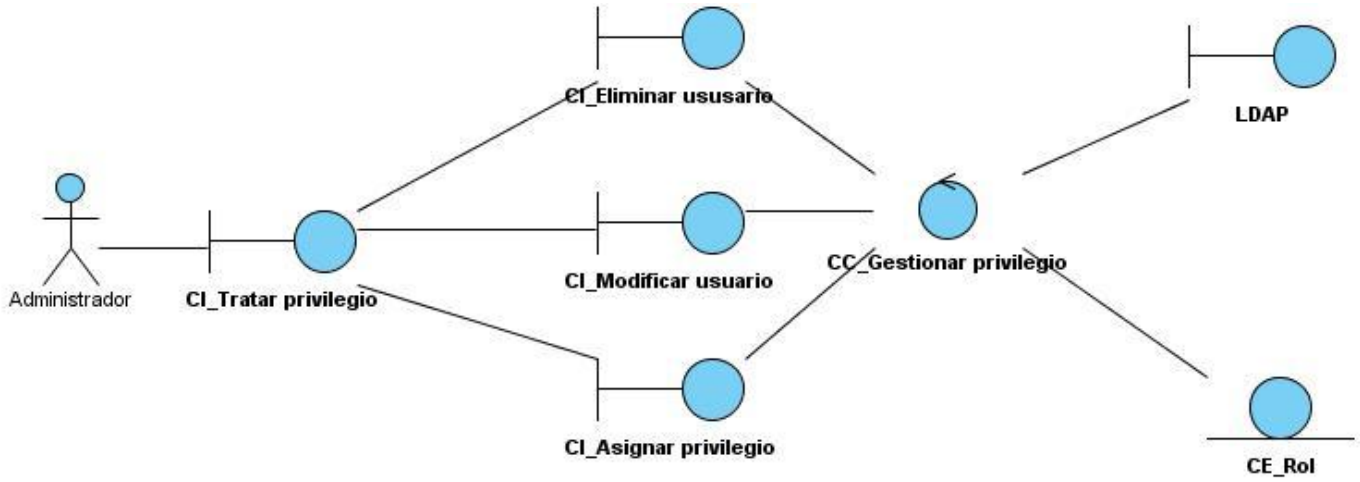


Figura 14. Diagrama de clases del Análisis del Caso de Uso "Gestionar Privilegio"

3.1.2 Diagramas de colaboración del Análisis.

A continuación se listan los diagramas de colaboración del Análisis para los casos de uso del sistema, estos muestran la interacción entre las clases para resolver cada Caso de Uso, el nivel de detalle es el correspondiente al flujo de Análisis de RUP, y los mensajes y clases presentes solamente sirven de acercamiento al diseño final y como documentación del proceso.

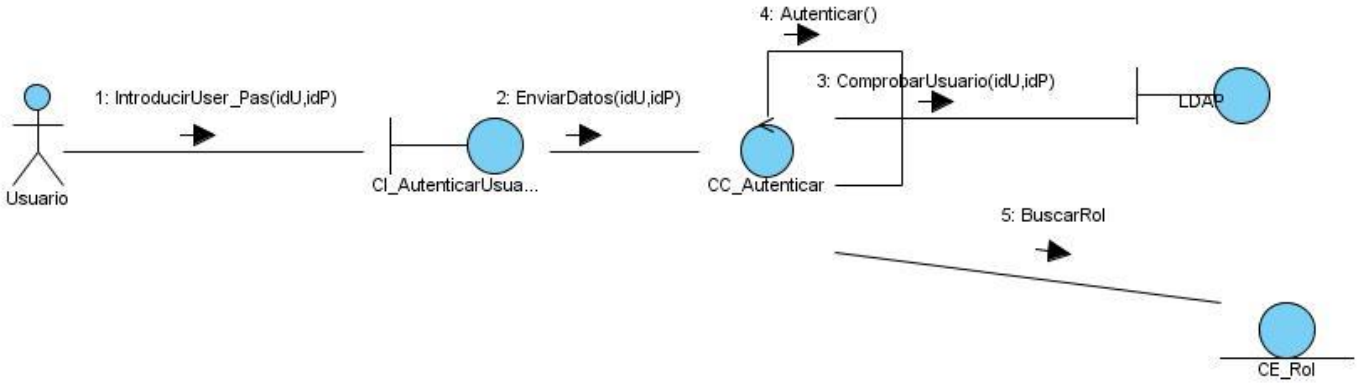


Figura 15. Diagrama de colaboración del Caso de Uso "Autenticar"

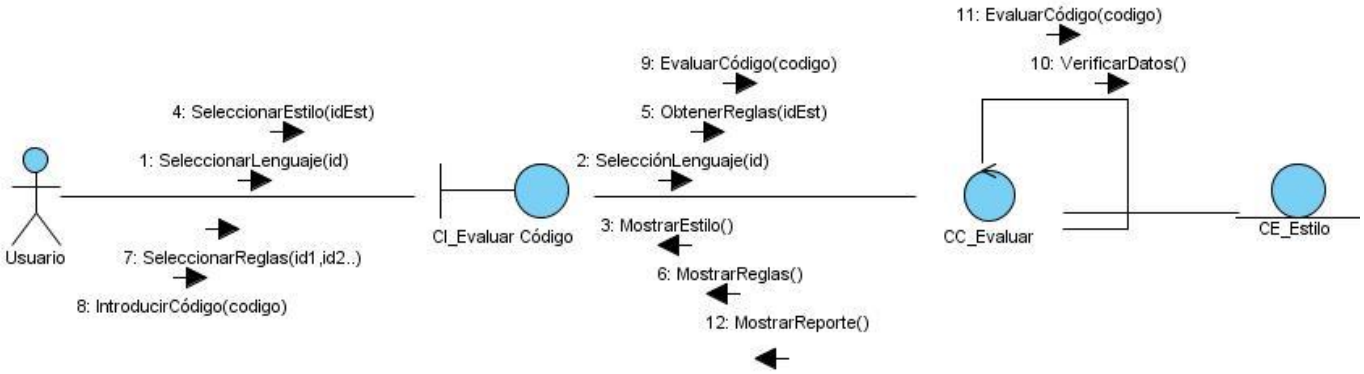


Figura 16. Diagrama de colaboración del Caso de Uso "Evaluar Código"

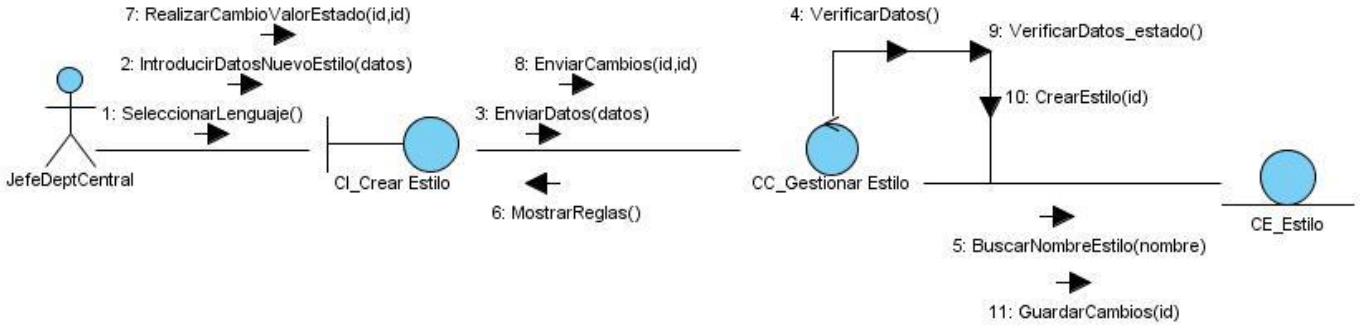


Figura 17. Diagrama de colaboración del escenario "Crear Estilo" del Caso de Uso "Gestionar Estilo"

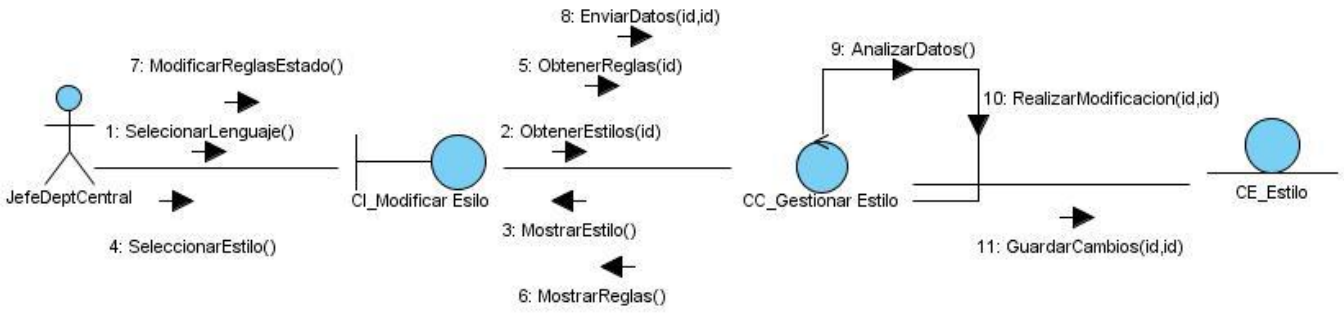


Figura 18. Diagrama de colaboración del escenario "Modificar Estilo" del Caso de Uso "Gestionar Estilo"

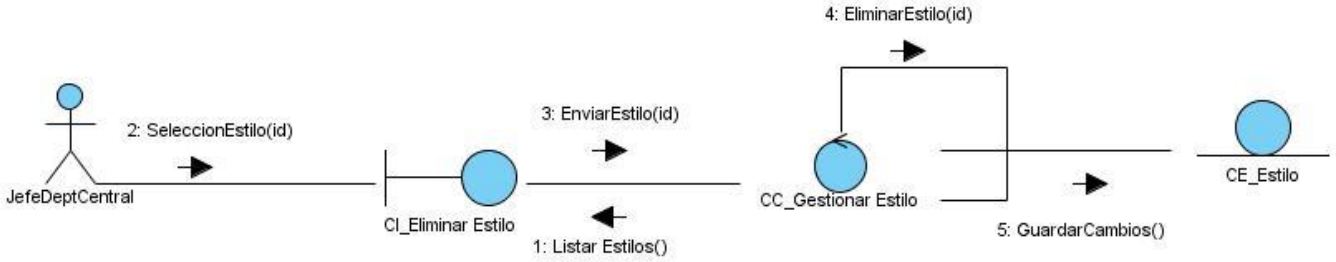


Figura 19. Diagrama de colaboración del escenario "Eliminar Estilo" del Caso de Uso "Gestionar Estilo"

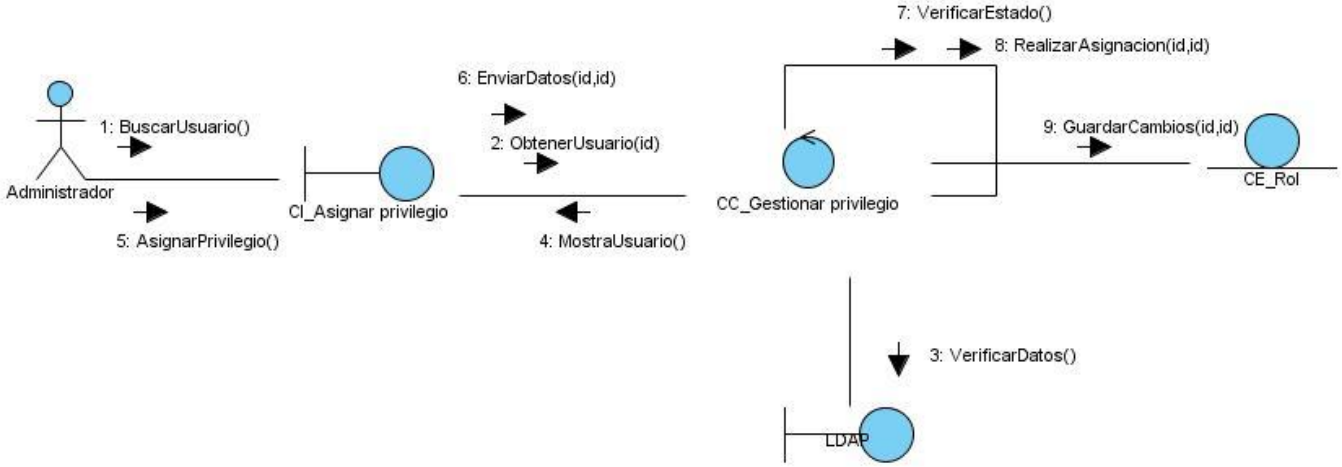


Figura 20. Diagrama de colaboración del escenario "Asignar Privilegio" del Caso de Uso "Gestionar Privilegio"

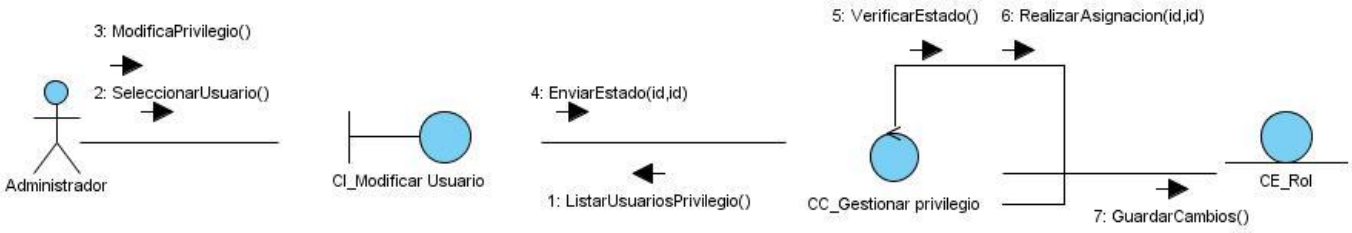


Figura 21. Diagrama de colaboración del escenario "Modificar Privilegio" del Caso de Uso "Gestionar Privilegio"

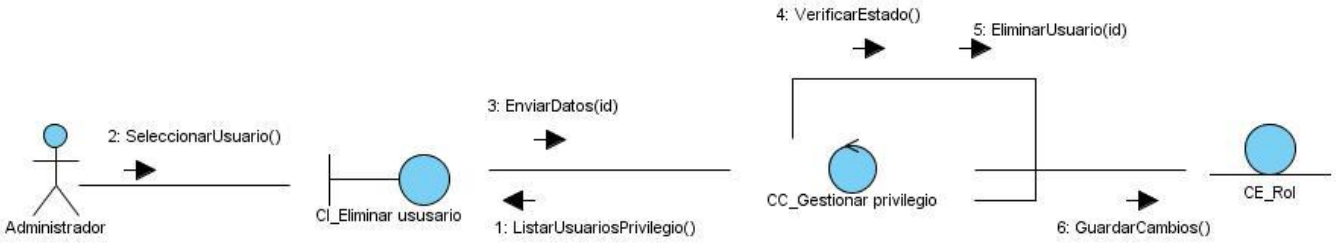


Figura 22. Diagrama de colaboración del escenario "Eliminar Privilegio" del Caso de Uso "Gestionar Privilegio"

3.1.3 Diagramas de secuencia del Análisis.

A continuación se detallan los diagramas de secuencia del Análisis para los casos de uso del sistema, estos muestran la interacción entre las clases para resolver cada Caso de Uso desde un punto de vista diferente de los anteriores, cada tomando en cuenta el tiempo de vida de cada objeto y la secuencia de mensajes como su nombre indica.

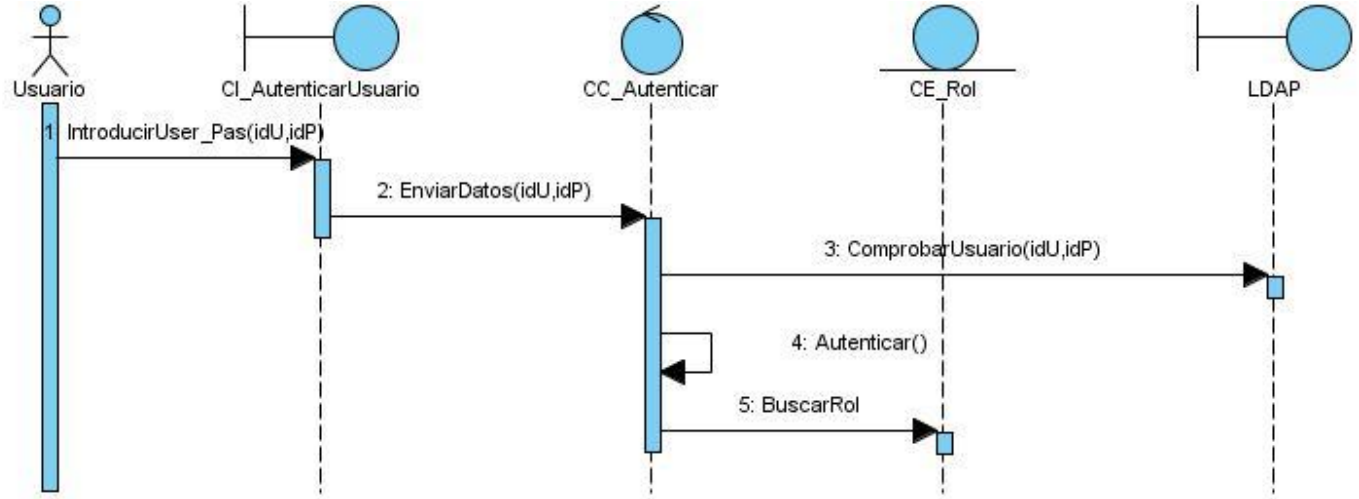


Figura 23. Diagrama de secuencia del Caso de Uso “Autenticar”

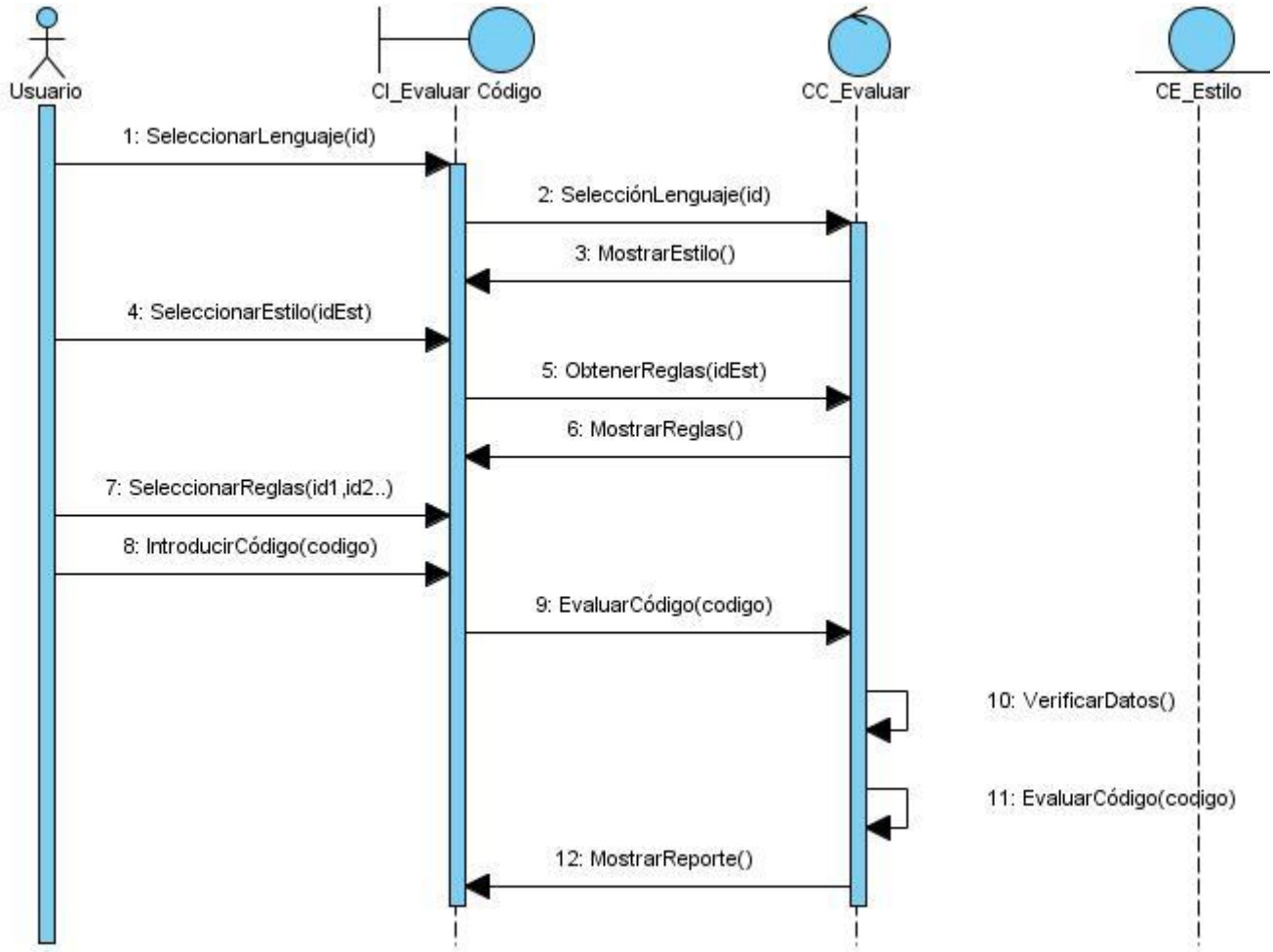


Figura 24. Diagrama de secuencia del Caso de Uso “Evaluar Código”

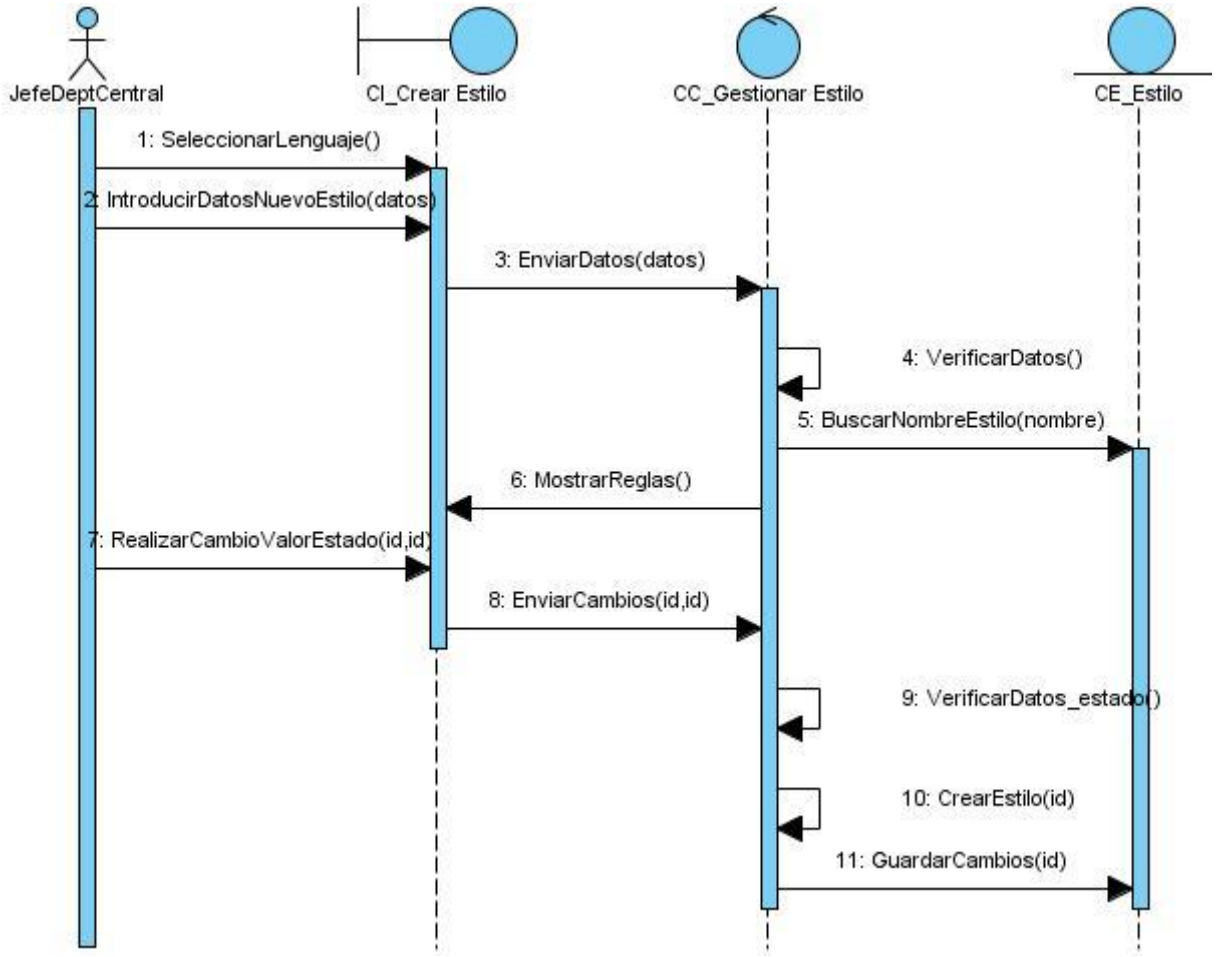


Figura 25. Diagrama de secuencia del escenario “Crear Estilo” del Caso de Uso “Gestionar Estilo”

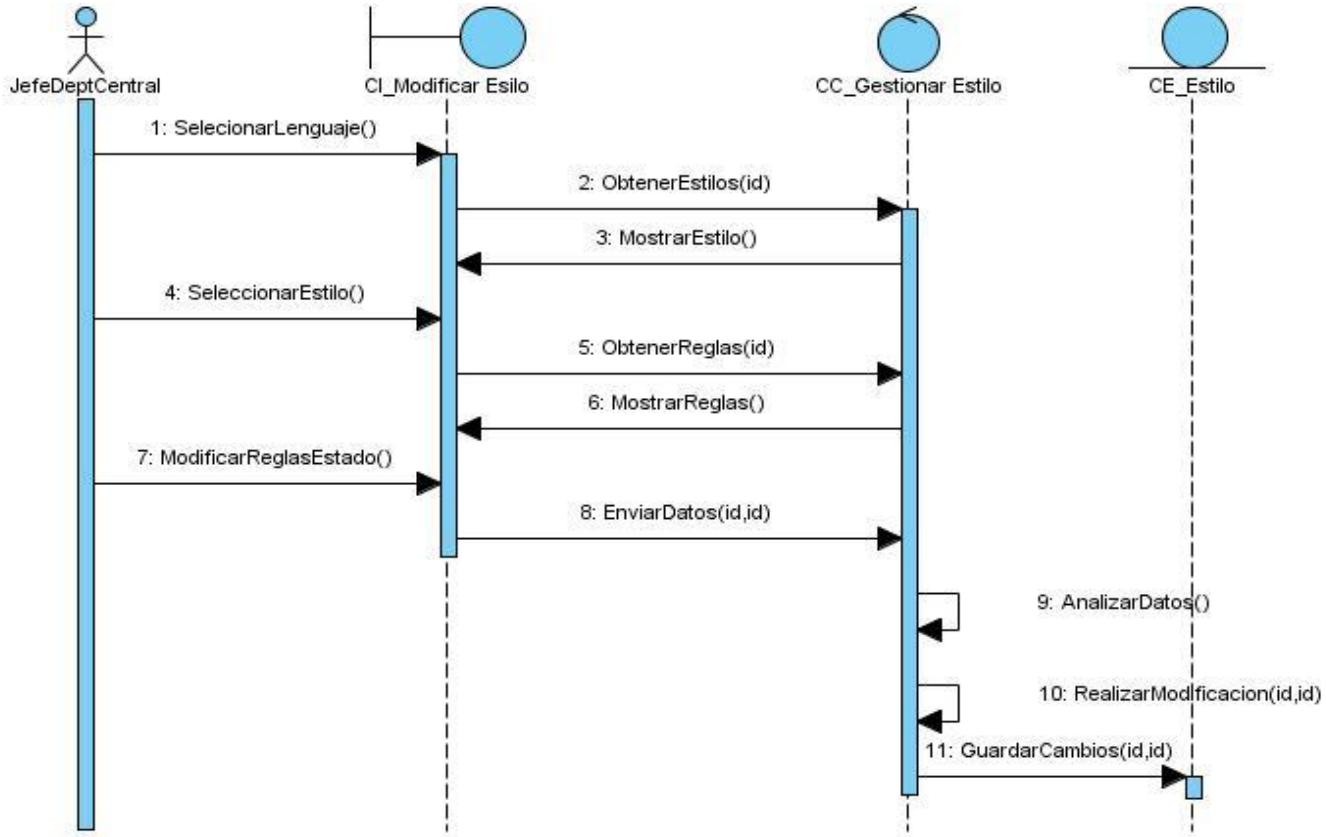


Figura 26. Diagrama de secuencia del escenario "Modificar Estilo" del Caso de Uso "Gestionar Estilo"

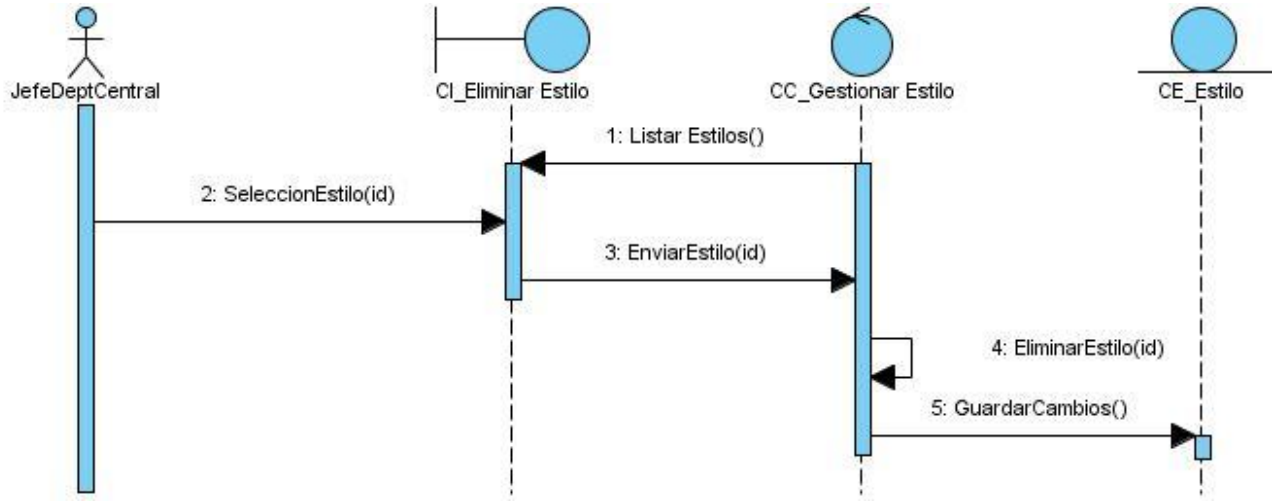


Figura 27. Diagrama de secuencia del escenario "Eliminar Estilo" del Caso de Uso "Gestionar Estilo"

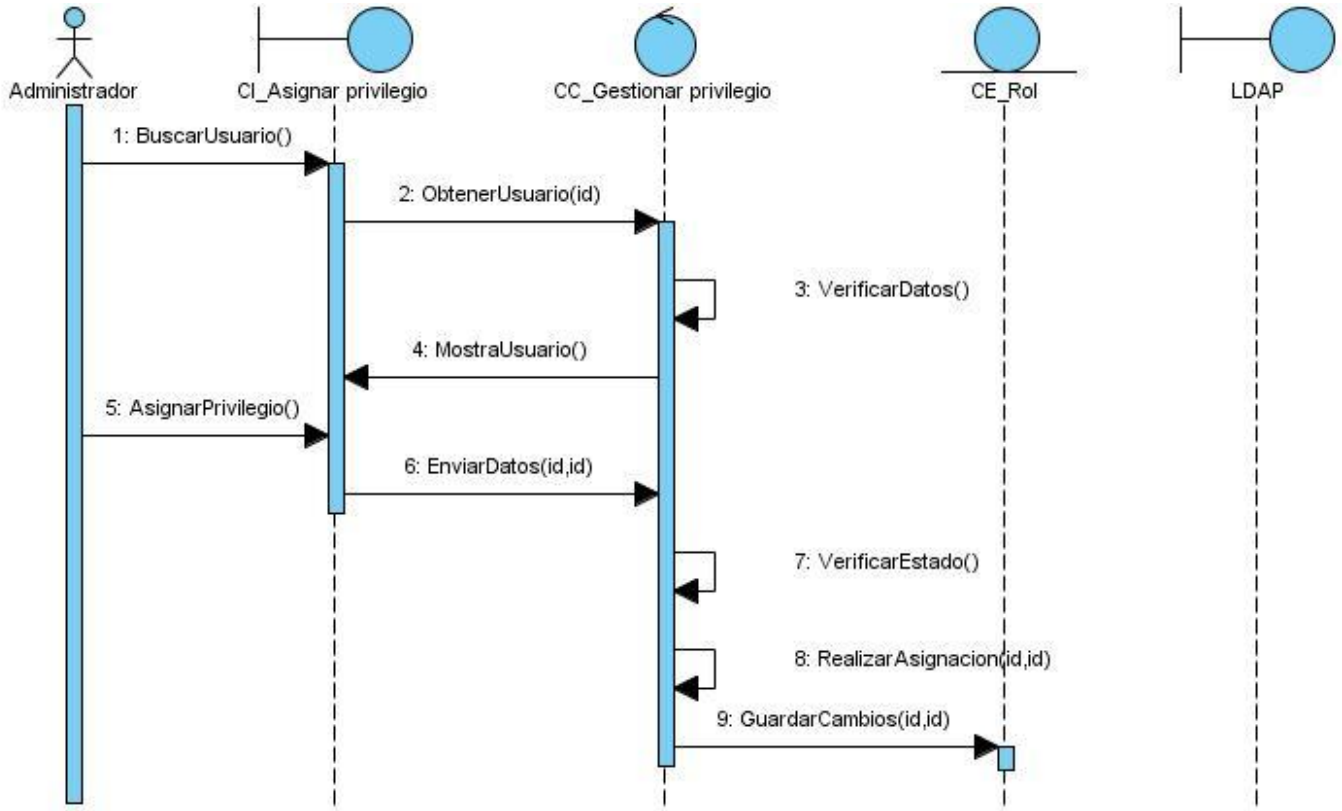


Figura 28. Diagrama de secuencia del escenario “Asignar Privilegio” del Caso de Uso “Gestionar Privilegios”

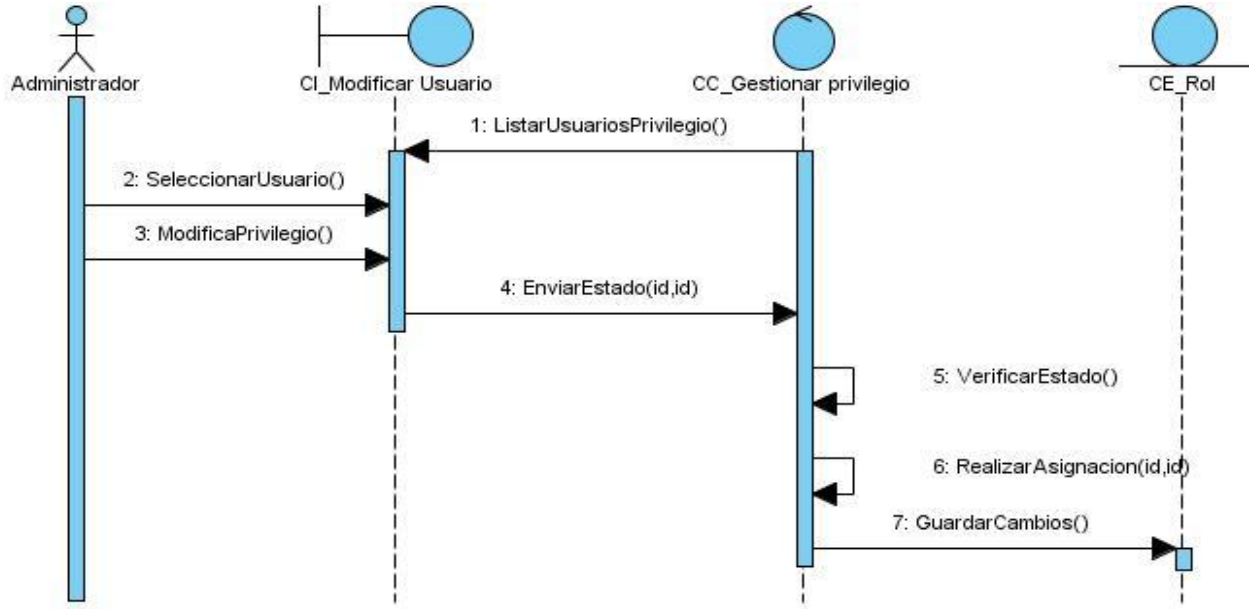


Figura 29. Diagrama de secuencia del escenario “Modificar Privilegio” del Caso de Uso “Gestionar Privilegios”

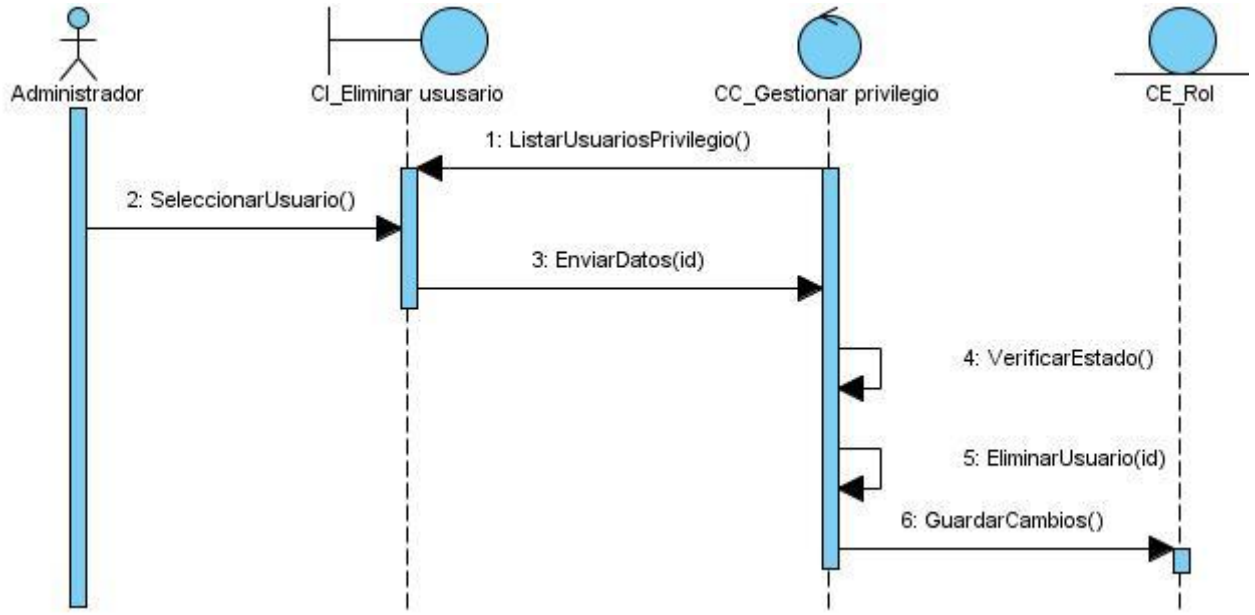


Figura 30. Diagrama de secuencia del escenario "Eliminar Privilegio" del Caso de Uso "Gestionar Privilegios"

3.2 Modelo de Diseño

Es un modelo de objeto que describe las actividades que se realizan en cada Caso de Uso centrándose en los requisitos funcionales y no funcionales. Es una abstracción de la implementación y una evidencia de cómo los requisitos tienen impacto en el sistema, demostrando además muy marcadamente una de las características básicas de RUP: el proceso guiado por casos de uso.

3.2.1 Diagrama de clases de diseño

Un diagrama de clases del Diseño es una representación más concreta que el diagrama de clase del Análisis, modela la parte estática del sistema, las clases y sus relaciones. Para clarificar el contenido de un diagrama de este tipo se utilizan determinados estereotipos, el grupo que se utilizó para los diagramas de este trabajo son los estereotipos Web, al tratarse de una aplicación Web. Entre estos podemos encontrar: Pagina Servidora, Pagina Cliente, Formulario, y otros de los cuales no se ha hecho uso.

Clases	Estereotipos	Función
Server Page (Página Servidora) [SP]		Representa una clase para el acceso a datos, su principal función es construir CP.

Client Page (Pagina Cliente) [CP]		Representa una clase con que interactúan con el usuario, su principal función es visualizar, interactuar, mostrar los que el usuario necesite.
Form (Formulario) [Fr]		Representa una clase que interactúa directamente con el usuario y su principal función es enviar datos entrados por el usuario a la SP.

Tabla 7. Estereotipos del Modelo de Clases de Diseño

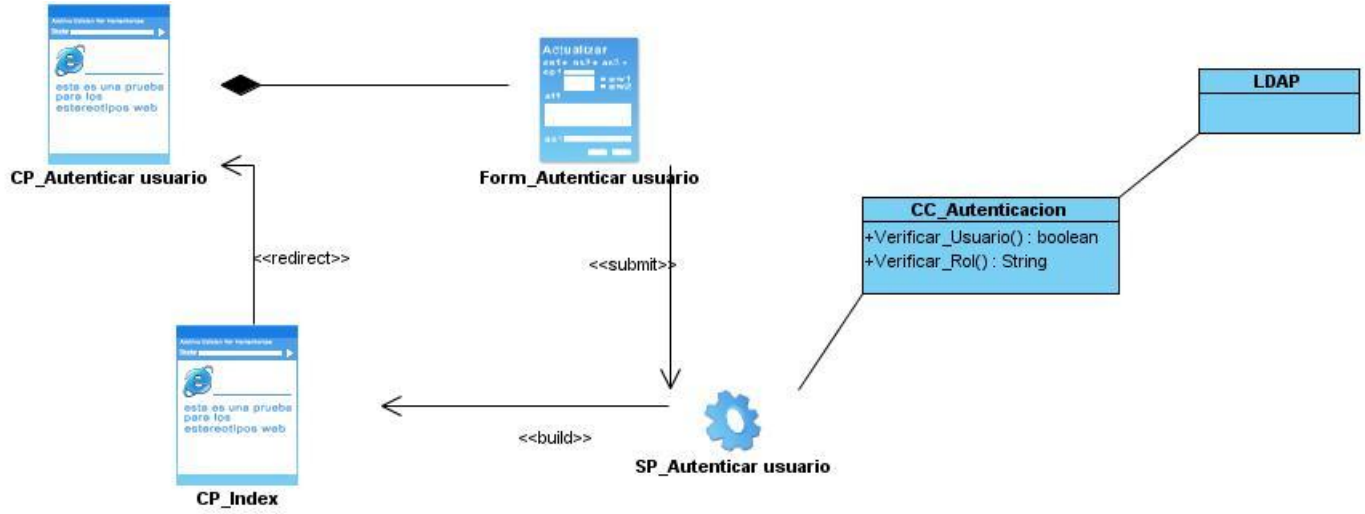


Figura 31. Diagrama de clases del Diseño del Caso de Uso “Autenticar”

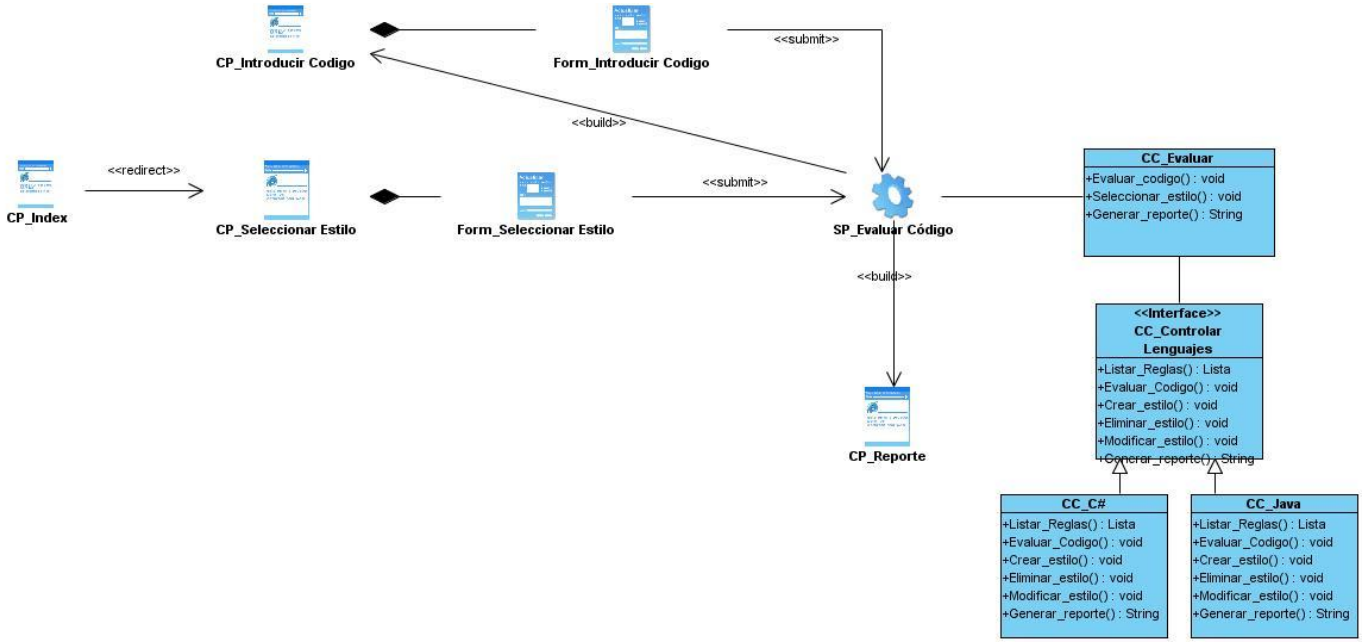


Figura 32. Diagrama de clase de Diseño del Caso de Uso “Evaluar Código”

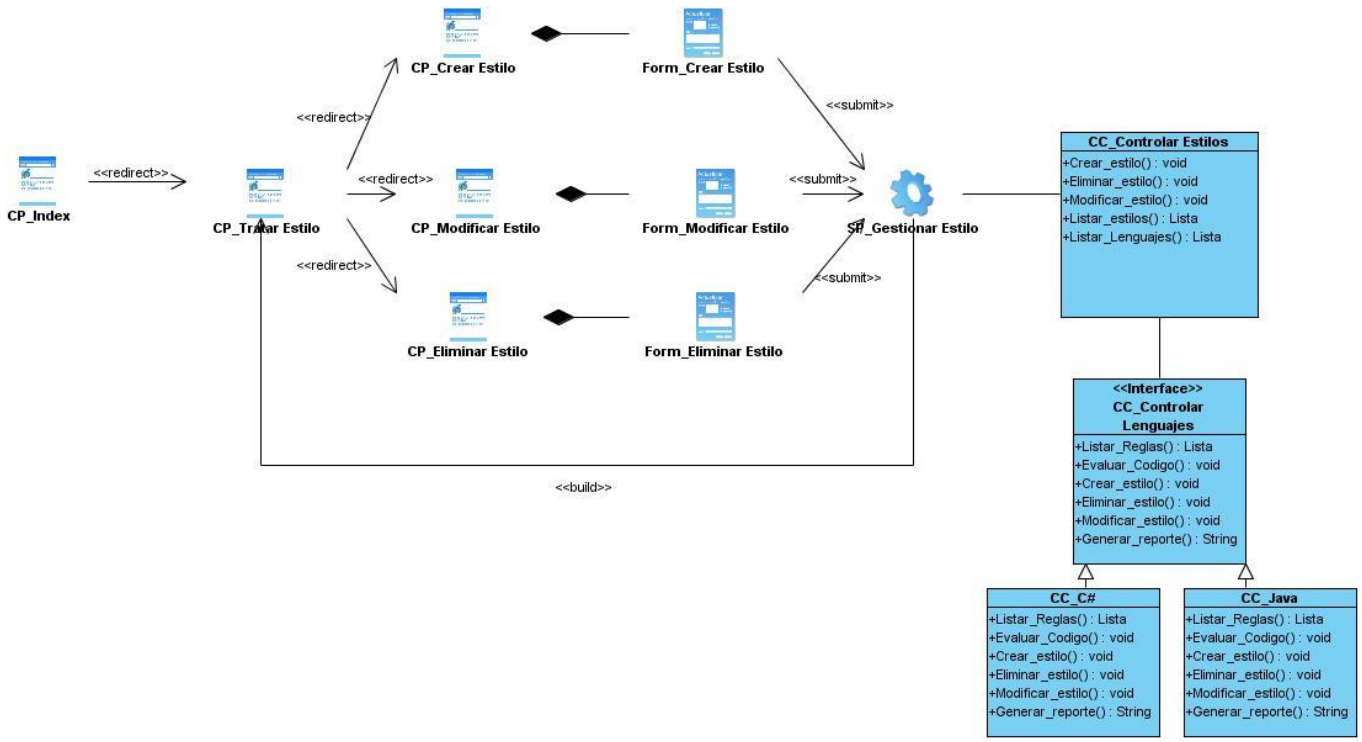


Figura 33. Diagrama de clase de Diseño del Caso de Uso “Gestionar Estilo”

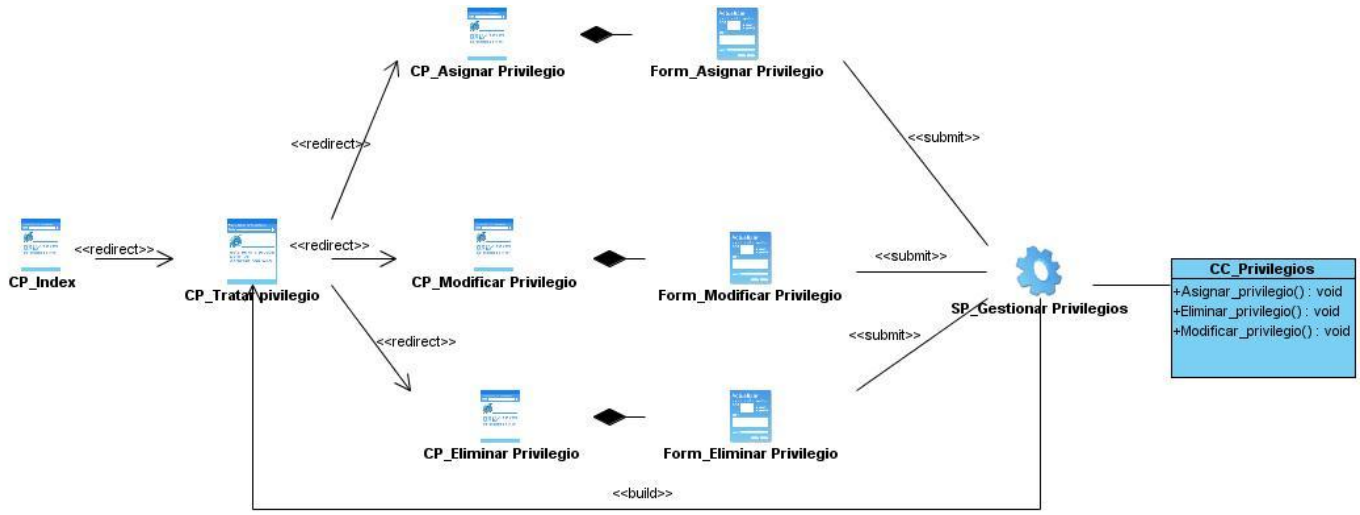


Figura 34. Diagrama de clase de Diseño del Caso de Uso “Gestionar Privilegio”

3.3 Patrones utilizados

Un patrón es la descripción de un problema y su solución, y provee al diseñados o programador de soluciones genéricas para circunstancias diversas. Muchos patrones ofrecen orientación sobre cómo asignar las responsabilidades a los objetos ante determinada categoría de problema (17).

3.3.1 Patrón Arquitectónico

Los patrones arquitectónicos son soluciones a problemas ya conocidos que ayudan a un mejor rendimiento, desarrollo y mantenimiento del software donde se apliquen. Como estilo propio impone reglas de transformación a un diseño de arquitectura. Imponen reglas de transformación a un diseño de arquitectura, describiendo la forma en que la aplicación debe ser construida y describiendo algunas funcionalidades a nivel de infraestructura (11).

Modelo Vista Controlador

Se propone utilizar el patrón Modelo Vista Controlador (MVC), el cual se caracteriza por separar el componente que manipula los datos de la aplicación, la interfaz gráfica de usuario y la lógica de control de la aplicación en tres partes diferentes (18).

Modelo: Administra el comportamiento y los datos del dominio de aplicación, responde a requerimientos de información sobre su estado (usualmente formulados desde la vista) y responde a instrucciones de cambiar el estado (habitualmente desde el controlador).

Vista: Maneja la visualización de la información.

Controlador: Se encarga de controlar el flujo entre la vista y el modelo (los datos).

Tanto la vista como el controlador dependen del modelo, el cual no depende de las otras clases. Esta separación permite construir el modelo independientemente de la representación visual y agregar nuevas opciones de representación de forma tal que no afecten el modelo. Este patrón permite el soporte de múltiples vistas sin afectar los datos.

3.3.2 Patrón de Diseño

Los patrones de diseño caracterizan por estar expresado en una relación de tres reglas, entre un determinado contexto, un problema y una solución. Para el diseño de software, el contexto permite al lector entender el entorno en el que reside el problema y la solución que podría ser apropiada dentro del mismo bajo un conjunto de requisitos incluyendo las limitaciones y restricciones (11).

Patrón Factory Method

Los patrones *GoF (Gang of Four)* se clasifican según su propósito en creacionales, estructurales y de composición, mientras que respecto a su ámbito se clasifican en clases y objetos. Para el desarrollo de la aplicación se eligió el uso del patrón *Factory Method*, pertenece a la clasificación creación. Tiene como propósito de abstraer el proceso de creación de instancias, resolviendo problemas relacionados con la creación de objeto.

El patrón *Factory Method* (método de fabricación) centraliza la creación de objetos en una clase controladora. Define una interfaz para crear objetos, dejando que sean las subclases quienes decidan de donde instanciar. Permite encapsular el conocimiento referente a la creación de objetos. Hace que el diseño sea más adaptable a cambios (19).

Es válido mencionar que entre los patrones *Abstract Factory* y *Factory Method* existe relación, según el libro del *GoF* indica que ellos no surgen aislados el uno del otro, sino en términos de lenguaje o sistema de patrones. Se puede determinar cuándo, cómo y por qué utilizar uno de ellos. A continuación, la figura 35 muestra la relación entre los patrones (19).

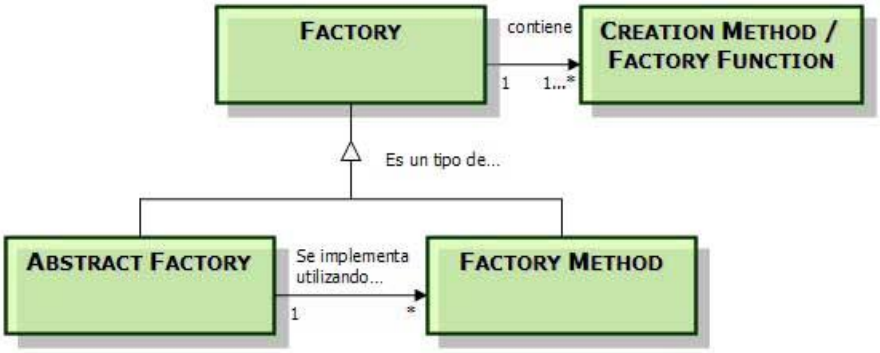


Figura 35. Relación entre los patrones *Factory*

Conclusiones

En el presente capítulo, los diagramas del Análisis, tanto de estructura como de interacción, sirven como un acercamiento a los diagramas de clases del Diseño, desarrollados para brindar a los programadores un modelo del sistema detallado y comprensible. Se especificaron los patrones de arquitectura y de diseño a utilizar, describiendo la importancia que presentan y los beneficios para la futura aplicación.

Conclusiones generales.

El estudio de los principales elementos asociados a los estilos de código, su importancia, y las principales herramientas que evalúan los estilos de código a nivel mundial, constituyen una documentación que permite identificar los posibles aportes de dichas herramientas al proceso de formación de un ingeniero en ciencias informáticas en el tema de la evaluación de estilos de código.

El propio estudio antes mencionado en combinación con la revisión de las principales tendencias y tecnologías en el desarrollo de sistemas informáticos, permitieron proponer una herramienta Web que cumple por sus propiedades con las características deseadas: multilenguaje, fácilmente accesible e independiente del sistema operativo.

El estudio realizado permitió establecer las funcionalidades y características del futuro sistema, que a su vez son la guía para la construcción de los modelos de análisis y diseño de RUP, que garantizan la implementación del sistema.

Recomendaciones.

- ✓ Desarrollar el sistema propuesto, para validar la solución obtenida y que beneficie el proceso de enseñanza-aprendizaje de PP1.
- ✓ Extender la herramienta propuesta para que permita la evaluación de estilos de código escritos en otros lenguajes de programación, según las necesidades de la asignatura.
- ✓ Vincular una ayuda al sistema para facilitar la interacción con el usuario final.

Referencias

(1) Modelo del Profesional y Objetivos Generales. *Modelo del Profesional y Objetivos de la carrera de Ingeniería en Ciencias Informáticas.* [En línea] 12 de 10 de 2010. [Citado el: 27 de febrero de 2011.] http://intranet2.uci.cu/node/46069/formacion/modelo.

(2) Soto, Prof Lauro. MiTecnológico. *Definicion De Lenguaje De Programación.* [En línea] 2010. [Citado el: 15 de enero de 2011.] http://www.mitecnologico.com/Main/DefinicionDeLenguajeDeProgramacion.

(3) **Calleja, M. A.** *Estándares de codificación.* 2000.

(4) *Estándares de Codificación*. [En línea] [Citado el: 15 de febrero de 2011.] Departamento Central de Práctica Profesional.

(5) StyleCop CodePlex. *StyleCop.* [En línea] [Citado el: 13 de enero de 2011.] http://stylecop.codeplex.com.

(6) *sourceforge.net.* [En línea] 2010. [Citado el: 15 de enero de 2011.] http://checkstyle.sourceforge.net/.

(7) **Copeland, Tom.** *PMD Applied.*

(8) FxCop. *msdn.* [En línea] [Citado el: 20 de mayo de 2011.]

http://msdn.microsoft.com/en-us/library/bb429476%28VS.80%29.aspx.

(9) Metodología de desarrollo de software. *Metodología de desarrollo de software.* [En línea] [Citado el: 13 de enero de 2011.]

(10**) José H. Canós, Patricio Letelier y Mª Carmen Penadés**. Métodologías Ágiles en el Desarrollo de Software. Valencia : s.n.

(11) **Pressman, Roger S.** *Software engineering: A practitioner's approach(SEVENTH EDITION).*

NewY ork San Francisco : s.n. ISBN 978-0-07-337 597 -7

(12) Visual paradimg UML*. Visual paradimg.* [En línea] [Citado el: 20 de mayo de 2011.]

http://www.visual-paradigm.com/product/vpuml/.

(13) Rational Rose Enterprise Edition . *EcuRed.* [En línea] [Citado el: 20 de mayo de 2011.] http://www.ecured.cu/index.php/Rational_Rose_Enterprise_Edition.

(14) Programación Java. *Lenguaje de Programación.* [En línea] [Citado el: 20 de mayo de 2011.] www.lenguaje-de-programacion.com/programacion-java.shtml.

(15) Inicio de Visual C#. *msdn.* [En línea] [Citado el: veinte de 05 de 2011.]

http://msdn.microsoft.com/es-es/vcsharp/aa336809.aspx.

(16) **Torossi, Gustavo.** *El Proceso Unificado de Desarrollo de Software.*

(17) **Lambar, Graig.** *UML y Patrones: Introducción al análisis y diseño orientado a objetos.* PRENTICE HALL, México, : production supervision: Dawn Speth White, 1999. ISBN: 970-17-0261-1.

(18) Welicki, León. *msdn*. [En línea] http://msdn.microsoft.com/es-es/library/bb972251.aspx.

(19) Patrones de Fabricación: Fábricas de Objetos . *msdn.* [En línea] [Citado el: 21 de mayo de 2011.] http://msdn.microsoft.com/es-es/library/bb972258.aspx.

Bibliografía

Calleja, M. A. *Estándares de codificación.* 2000.

José H. Canós, Patricio Letelier y Mª Carmen Penadés. Métodologías Ágiles en el Desarrollo de Software. Valencia : s.n.

Lambar, Graig. *UML y Patrones: Introducción al análisis y diseño orientado a objetos.* PRENTICE HALL, México, : production supervision: Dawn Speth White, 1999. ISBN: 970-17-0261-1.

Pressman, Roger S. *Software engineering: A practitioner's approach (SEVENTH EDITION).* NewY ork San Francisco : s.n. ISBN 978-0-07-337 597 -7

Torossi, Gustavo. *El Proceso Unificado de Desarrollo de Software.*