UNIVERSIDAD CÉSAR VALLEJO

FACULTAD DE INGENIERÍA

ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS

"APLICACIÓN DE TÉCNICAS DE REFACTORIZACIÓN PARA GARANTIZAR MAYOR MANTENIBILIDAD EN EL SISTEMA PROTOTIPO SIAEC EN SU PROCESO DE EVOLUCIÓN"

TESIS PARA OBTENER EL TÍTULO PROFESIONAL DE INGENIERO DE SISTEMAS

AUTOR:

Br. VALENCIA CERNA, NELIDA JANETH

ASESOR:

Mg. CÁRDENAS ESCALANTE, LAÍN JARDIEL

LÍNEA DE INVESTIGACIÓN:

SISTEMA DE INFORMACIÓN TRANSACCIONAL

Trujillo – Perú

2016

PÁGINA DEL JURADO

El presidente y los miembros del Jurado Evaluador designado por la Escuela Profesional de Ingeniería de Sistemas.

APRUEBAN

La tesis denominada:

"APLICACIÓN DE TÉCNICAS DE REFACTORIZACIÓN PARA GARANTIZAR MAYOR MANTENIBILIDAD EN EL SISTEMA PROTOTIPO SIAEC EN SU PROCESO DE EVOLUCIÓN"

Presentada por:

_____

Br. Valencia Cerna, Nelida Janeth

Aprobado por:

_____

Dr. Pacheco Torres, Juan Francisco

_____ _____

Mg. Cárdenas Escalante, Laín Jardiel

Mg. Urquizo Gómez, Yosip Vladimir

DEDICATORIA

A Nuestro Padre Celestial que gobierna los cielos y la tierra, y me brinda lo necesario para seguir de pie.

A mi maravillosa y hermosa Familia, especialmente a mis padres quienes se esfuerzan día a día para brindarme lo mejor.

Este logro es de ustedes queridos padres.

A mis mejores amigos, quienes fueron cómplices y de mucho apoyo durante mi formación Universitaria.

Valencia Cerna Nelida Janeth

AGRADECIMIENTO

A Dios,

Nuestro Padre Celestial quien tiene paciencia y misericordia de todos sus hijos, gracias por toda la bendición que me brindas, por darme valentía y fuerzas para sobresalir en estos años de formación Universitaria, eres roca que sostiene mi vida, todo te lo ofrezco a ti mi Señor.

A mis amados Padres,

Por su gran esfuerzo de día a día, gracias por dejarnos la mejor herencia que ustedes pueden dar "Educación", gracias por su inmenso amor, motivación y esperanza, por dar todo de sí, no importa si trabajan de día o de noche todo siempre lo hacen por amor.

A la Universidad Cesar Vallejo,

Por los beneficios brindados a cada uno de sus alumnos durante los 5 años de formación universitaria, beneficios que me brindaron para poder cumplir las metas que eh planteado; por la buena selección de docentes quienes dan su esfuerzo para prepararnos para los retos que están por llegar en nuestro entorno profesional.

A mi asesor el Ing. Cárdenas Escalante Laín Jardiel,

Por guiarme durante el Desarrollo del Proyecto de investigación, por su paciencia y por brindarme sus conocimientos, que son muy útiles para una correcta formación.

Valencia Cerna Nelida Janeth

DECLARATORIA DE AUNTENTICIDAD

Yo **Valencia Cerna, Nelida Janeth** con DNI N.º **70200629**, a efecto de cumplir con las disposiciones vigentes consideradas en el Reglamento de Grados y Títulos de la Universidad César Vallejo, **Facultad de Ingeniería**, **Escuela de Ingeniería de Sistemas**, declaro bajo juramento que toda la documentación que acompaño es veraz y auténtica.

Así mismo, declaro también bajo juramento que todos los datos e información que se presenta en la presente tesis son auténticos y veraces.

En tal sentido asumo la responsabilidad que corresponda ante cualquier falsedad, ocultamiento u omisión tanto de los documentos como de información aportada por lo cual me someto a lo dispuesto en las normas académicas de la Universidad César Vallejo.

Trujillo, diciembre del 2016

_____

Valencia Cerna, Nelida Janeth

PRESENTACION

Señores Miembros del Jurado:

En el cumplimiento del Reglamento de Grados y Títulos de la Universidad César Vallejo presento ante ustedes la Tesis titulada: **"APLICACIÓN DE TÉCNICAS DE REFACTORIZACIÓN PARA GARANTIZAR MAYOR MANTENIBILIDAD EN EL SISTEMA PROTOTIPO SIAEC EN SU PROCESO DE EVOLUCIÓN".** La misma que someto a vuestra consideración y espero que cumpla con los requisitos de aprobación para obtener el Título Profesional de Ingeniero de Sistemas.

Valencia Cerna Nélida Janeth

ÍNDICE

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE ANEXOS

RESUMEN

La presente investigación denominada “APLICACIÓN DE TÉCNICAS DE REFACTORIZACIÓN PARA GARANTIZAR MAYOR MANTENIBILIDAD EN EL SISTEMA PROTOTIPO SIAEC EN SU PROCESO DE EVOLUCIÓN” tiene como propósito aplicar Técnicas de refactorización para organizar el código de una forma más eficiente y fácil de entender, de tal forma que, si estamos trabajando con más programadores y necesitan ver tu código, no les resulte complicado saber qué hace tu código y pueda ser reutilizado sin complicaciones. Una de las razones para Refactorizar es ayudar al código a mantenerse en “buena forma”, ya que con el tiempo los cambios en el software hacen que este pierda su estructura, y esto hace difícil ver y preservar el diseño, por lo tanto, podría decirse también que Refactorizar ayuda a evitar los problemas típicos que aparecen con el tiempo, como, por ejemplo, un mayor número de líneas para hacer las mismas cosas o código duplicado por ello se concluye que:

La aplicación de Técnicas de Refactorización si logró que el sistema prototipo SIAEC puede ser mantenible para cuando se quiera añadir más requerimientos, o agregar más módulos al Sistema ya que no se invertirá demasiado tiempo para poder entender el código fuente.

La implementación de técnicas de refactorización disminuyó la densidad de defectos en un 50.1% del sistema prototipo SIAEC.

La implementación de técnicas de refactorización redujo determinadamente en un 49.82% la Complejidad del sistema prototipo SIAEC.

La implementación de técnicas de refactorización incrementó la Comprensibilidad en un 68.97% del sistema prototipo SIAEC.

Palabras clave: Técnicas de refactorización, Mantenibilidad, Eficiencia.

ABSTRACT

The present investigation called "APPLICATION OF REFACTION TECHNIQUES TO GUARANTEE GREATER MAINTENANCE IN THE SIAEC PROTOTYPE SYSTEM IN ITS PROCESS OF EVOLUTION" has the purpose of applying Refactorization Techniques to organize the code in a more efficient and easy to understand way, in such a way that, if we are working with more programmers and need to see your code, it is not difficult for them to know what your code does and can be reused without complications. One of the reasons for Refactoring is to help the code to stay in "good shape", since over time the changes in the software cause it to lose its structure, and this makes it difficult to see and preserve the design, therefore, it could It should also be said that Refactoring helps to avoid the typical problems that appear over time, such as, for example, a greater number of lines to do the same things or duplicate code, so it is concluded that:

The application of Refactoring Techniques did achieve that the SIAEC prototype system can be maintained for when you want to add more requirements, or add more modules to the System since you will not spend too much time to understand the source code.

The implementation of refactoring techniques decreased the defect density in 50.1% of the SIAEC prototype system.

The implementation of refactoring techniques reduced the complexity of the SIAEC prototype system by 49.82%.

The implementation of refactoring techniques increased the Comprehensibility by 68.97% of the SIAEC prototype system.

Keywords: Refactoring techniques, Maintainability, Efficiency.

CAPÍTULO I

INTRODUCCIÓN

I. INTRODUCCIÓN

1.1 Realidad Problemática

El Mantenimiento y la Calidad de desarrollo de Software son considerados en todas las Organizaciones como principal objetivo estratégico de ejecución, razones por la cual los procesos más sobresalientes de las Empresas son dependientes a los sistemas de información para su buen funcionamiento, además se debe tener en cuenta de que si un software no está bien programado o no cumple con los principios afecta los costos y esfuerzos de mantenimiento a la empresa.

Los Desarrolladores de Software o también llamados Programadores dedican numerosas horas al desarrollo de software, aunque de cierta manera sabemos que en nuestro mundo el tiempo es limitado más aun para los que se dedican al trabajo; pero no se trata solo del tiempo que transcurre durante nuestra labor sino que a pesar de las horas dedicadas a la programación también se trata de conseguir que el código sea de la mayor calidad posible, pero la característica más importante o prioritaria para un Desarrollador es sin lugar a duda la entrega rápida al cliente e incluyendo también la satisfacción de uso del software, en pocas palabras satisfacción al cliente o usuario; sin embargo, el deseo de querer cumplir ciertos factores conllevan a formas no adecuadas de programación, y es por el simple hecho de que el desarrollador solo prioriza una y es la que tiene que ver con el Usuario y esto indica que el software debe funcionar como sea, se aplique lo que se aplique; y porque no decir también que se tiene como segunda prioridad el prestigio del Desarrollador o de la Empresa donde labore.

Como desarrolladores sabemos perfectamente que el usuario interactúa directamente con el software mediante la presentación y no revisa la estructura de la construcción del código ya que el usuario sólo le interesa que funcione y tenga los requerimientos que el indicó.

Por lo tanto, los desarrolladores olvidan la forma correcta de organizar el código llegando a incumplir inconscientemente algunos principios tales como los que están basados en el Diseño Orientado a Objetos, por ejemplo tenemos el DRY - Don't repeat yourself, este principio consiste en la reducción de duplicación de código, usualmente se da el caso de que el desarrollador de cierta forma no se dé cuenta de que está haciendo duplicación de código, ya que simplemente por terminar lo más rápido posible no tienen en cuenta la estructuración de su código; o suele ser el caso de que por pereza le conlleve a la duplicación del mismo, esto se refiere que en el momento de implementar un método cualquiera y que se desee agregar otro similar al ya creado, solo implicaría cambiar algunas modificaciones para diferenciarlo un poco del original, por ende esta práctica de copiar y pegar llegaría a ser muy fácil y sin darse cuenta de que se han creado varias copias del método y que están distribuidas en diferentes partes del código.

Sin embargo de alguna u otra forma llega ese momento cuando el cliente añade otro requerimiento para el software y quizá ese requerimiento indique modificar el método entonces el programador modifica sólo un método pero no recuerda que se tiene que hacer la misma modificación o corrección en los otros métodos que se crearon; por lo tanto surgen varios errores de compilación e implica más horas para tratar de modificar todo los errores generados; por otra parte también puede darse el caso de que la duplicación se origine de la poca coordinación en los equipos de desarrollo, el trabajo en equipo es muy importante para cualquier proyecto en ejecución de cualquier rubro.

Otro de los principios básicos de Diseño Orientado a Objetos es KISS - Keep It Simple Stupid, este principio trata de minimizar la complejidad, usualmente existen formas más sencillas de programar pero el desarrollador simplemente lo hace difícil y se complica mucho en el desarrollo pudiendo hacer que el código sea más entendible de lo que quizá es; otro principio es YAGNI – You Aren’t Gonna Need It, en este caso el principio nos indica de que no debemos crear código que nunca se tiene como plan usarlo, el programador debe aprender a desarrollar

para las necesidades actuales mas no futuras, esto debido a diferentes razones: quizá no seas tú el que modifique el código, otra podría ser que existe un código mejorado que se pueda aplicar y reemplazar al ya creado.

Los principios básicos de diseño orientado a objetos, hacen mención de los problemas que se ocasionan durante el desarrollo de software, se mencionó lo que es duplicación de código, complejidad, código no usado y entre otras más características relacionadas a lo ya mencionado, son reconocidas también como Malos olores de código (Code Smell), los malos olores son síntomas que nos indican problemas más profundos de la calidad ya sea de código o diseño e incluso ambos.

Si bien sabemos que al no tener el código organizado implica muchas cosas, por ejemplo; si hablamos de la Mantenibilidad, es una de las características de calidad de Software y debe establecerse como objetivo principal a principios de desarrollo del software, por otra parte, sin dejar de mencionar las otras características de calidad de software, pues estas pueden derivarse de la Mantenibilidad. El IEEE define mantenibilidad como: *"La facilidad con la que un sistema o componente software puede ser modificado para corregir fallos, mejorar su funcionamiento u otros atributos o adaptarse a cambios en el entorno"*. En la ISO 9126-1 se define que "*La Mantenibilidad está indicada por los siguientes subatributos: Facilidad de Análisis, Facilidad de Cambios, Facilidad de Pruebas y Estabilidad*".

Entonces, se sabe que la realidad del día a día lleva inevitablemente a los siguientes problemas:

- ✓ Pérdida de Calidad de Software que se manifiesta con la existencia de Malos olores en el código:
  - La duplicación de código.
  - Métodos o clases muy largas
  - Creación de código no usado.
- ✓ Aumento de Complejidad de Software.
- ✓ Baja Comprensibilidad de código fuente del software.
- ✓ Baja Conformidad de Software.

Por lo tanto, implicaría un mayor esfuerzo de programación para cada cambio que se quiera realizar e inclusive afecta los costos y esfuerzo de mantenimiento a la empresa.

Una forma de solucionar esos problemas comunes de la programación es aplicar técnicas de refactorización, consiste en mejorar el código una vez escrito, cambiando su estructura interna sin modificar su comportamiento externo para dar lugar a otra sintaxis que se adapte mejor a las especificaciones actuales.

1.2 Trabajos Previos

1.2.1 Nivel Internacional

- (Marticonera Sánchez, 2013)

  En su Tesis *"Refactorización sobre Programación genérica en Lenguajes Orientado a Objetos".*

  Planteo como Objetivo Principal lo siguiente:

  - *"Definición y Ejecución de refactorizaciones con el mayor grado de reutilización para la familia de lenguajes orientados a objetos, estáticas o fuertemente tipados, con capacidades para programación genérica"*

  Como Objetivos Específicos definió lo siguiente:

  - Almacenamiento y recuperación de la información del código
  - Definición de los elementos de las refactorizaciones
  - Características de las refactorizaciones en base a los elementos que la conforman
  - Catálogo de refactorización en programación genérica
  - Validación de la solución planteada sobre el catálogo de refactorización
  - Establecer un proceso de refactorización
  - Construcción de un prototipo

  La Tesis fomenta diferentes definiciones que están enlazadas a la Refactorización más que todo que influyen dentro de ella, se habla sobre la Calidad de software donde se define la importancia de la mantenibilidad, la eficiencia, entre otros parámetros importantes que se deben tener en cuenta para la fase de Desarrollo de software; también se menciona la Evolución del Software donde se toma definiciones de (Sommerville, 2011); además de cómo aplicar Funciones básicas del Catálogo de Refactorizaciones sobre genericidad.

  Luego de la investigación e implementaciones, se concluye lo siguiente según el objetivo principal:

- ✓ Se logró Formalizar en un mayor grado la definición de refactorizaciones estableciendo una estructura más precisa para el debido entendimiento del mismo, como fin se tenía la eliminación de un cierto grado de subjetividad y la falta de precisión.

- (Cuenca Ortega, 2013)

  En su Tesis *"Técnicas de Refactorizacion para Maude".*

  Planteo como Objetivo Principal:

  - *"Implementación de sistema de refactorización para el lenguaje Maude para mejorar la legibilidad del código".*

  Maude es un lenguaje y un entorno de programación que da soporte a una lógica denominada "*Lógica Ecuacional con Pertenencia" y también a la "Lógica de reescritura".*

  El sistema de refactorización es aplicado a Maude mediante un catálogo de técnicas para la respectiva mejora de código, el catalogo consta de cinco técnicas:

  - ✓ ***Añadir un Atributo constructor***
  - ✓ ***Eliminar una definición no usada***
  - ✓ ***De ecuaciones condicionales a incondicionales***
  - ✓ ***Añadir atributos de memorización***
  - ✓ ***Transformación a recursión de cola***

  Durante el desarrollo de la Tesis y con la obligación de tratar de lograr el Objetivo principal se concluyó lo siguiente:

  - ✓ El número de refactorizaciones que se lograron implementar en el sistema Maude son cuatro de las cinco refactorizaciones que se han presentado, que hemos elegido con criterios de representatividad, por lo que uno de los primeros trabajos futuros seria implementar las refactorizaciones restantes.
  - ✓ El sistema Maude actualmente es capaz de Refactorizar módulos funcionales, es decir únicamente operadores

ecuacionales. Por lo tanto, se aplicaría como trabajos futuros la extensión del lenguaje Maude para que pueda trabajar también con los módulos del sistema que contengan reglas.

1.3 Teorías Relacionadas al tema

1.3.1 PROGRAMACIÓN ORIENTADA A OBJETOS.

Es importante primero aclarar la diferencia que existe entre Programación Orientada a Objetos y Lenguaje Orientada a Objetos:

- **Programación Orientada a Objetos**

  Es una "filosofía" o un modelo de programación, que consiste en su teoría y metodología, es conveniente conocer y estudiar, antes de nada.

- **Lenguaje Orientada a Objetos**

  Es un lenguaje de programación que permite diseñar aplicaciones orientadas a objetos.

Según (Grady, 2001) en su libro "Análisis de diseño Orientado a Objetos con Aplicaciones", define la Programación Orientada a Objetos como:

"Método de implementación, donde los programas se organizan como colecciones cooperativas de objetos, cada uno representa una instancia de alguna clase, además las clases pertenecen a una jerarquía de clases se unen mediante relaciones de herencia."

Se debe tener en cuenta que los conceptos fundamentales de programación son:

- **Objetos**
- **Clases**
- **Herencia**
- **Mensajes**
- **Polimorfismo**

¿Qué es un Objeto?

Se denomina objeto a una sola unidad donde se combina datos y funciones, por lo tanto, podemos decir que dentro de los objetos habitan datos de los lenguajes de programación tradicionales, tales como números, arrays, cadenas y registros además de funciones que

operan sobre ellos; los objetos de programación se usan mayormente para el modelado de objetos o entidades reales. Se podría decir que un objeto es la representación en un programa de un concepto que contiene toda la información necesaria para abstraer los datos que describen sus atributos o las operaciones que se puedan realizar sobre los mismo.

El medio para acceder a los datos privados de un objeto son las funciones, por ejemplo, cuando se quiere acceder a la lectura de un elemento dato se hace el llamado a la función del objeto luego este devolverá el valor que contiene la función invocada; se realiza ese proceso debido a que no se puede acceder directamente a los datos ya que estos permanecen ocultos, esto ayuda a evitar alguna acción accidental dirigida al objeto emitida por funciones externas.

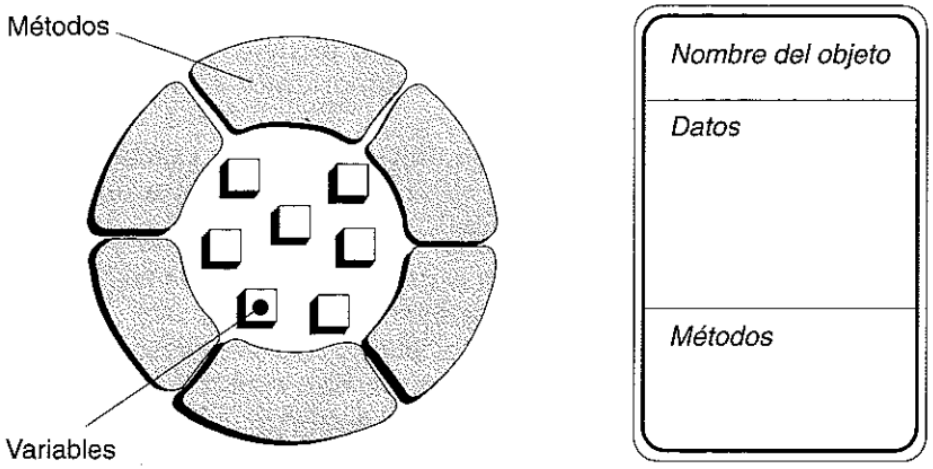


***Figura 1**. Notaciones graficas de objetos*

¿Qué es una Clase?

Según (Grady, 2001)*"Los objetos son las instancias de una clase, se usa el símil "variable – tipo" de la programación estructurada, se tiene entendido que un objeto es una variable que tiene el comportamiento y estados del tipo (objeto)"*

Una Clase podría definirse como una entidad que declara un conjunto similar de objetos, en la programación orientada a objetos se considera clase a una estructura que comprende datos y procedimientos o funciones, además tienen como propósitos: definir abstracciones y favorecer la modularidad.

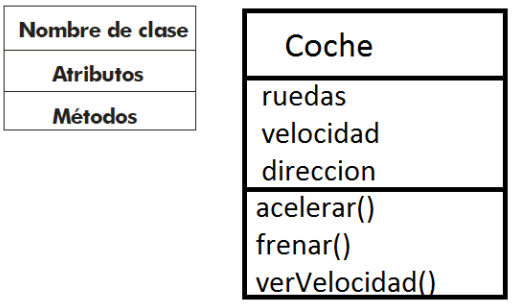


Figura 2. *Representación gráfica de una Clase.*

```
1  public class Persona {
2        public String nombre;
3        public int edad;
4
5        public void correr(){
6                /* por implementar */
7        }
8
9  }
```

Figura 3.*Representación lógica de una Clase.*

¿Qué es una Herencia?

Las propiedades comunes de los objetos son los atributos y operaciones que son clasificadas en una clase, donde ésta también cuenta con propiedades y funciones comunes que se agrupan en una Superclase. La herencia es un mecanismo que permite la definición de nuevas clases partiendo de otra que ya está definida, cuando la clase realiza el proceso de herencia, automáticamente adquieren todo el comportamiento de esa clase definida, sin embargo, no puede quedarse solo con las características heredadas, sino que también se podría introducir nuevas características particulares e independientes que las diferencian de la superclase.

Se cuenta con dos diferentes tipos de Herencia: simple y múltiple.

- **Herencia Simple**

  Una clase tiene como máximo una sola superclase; la herencia simple admite que la herencia de las propiedades que contiene la superclase se realice mediante una cadena jerárquica.

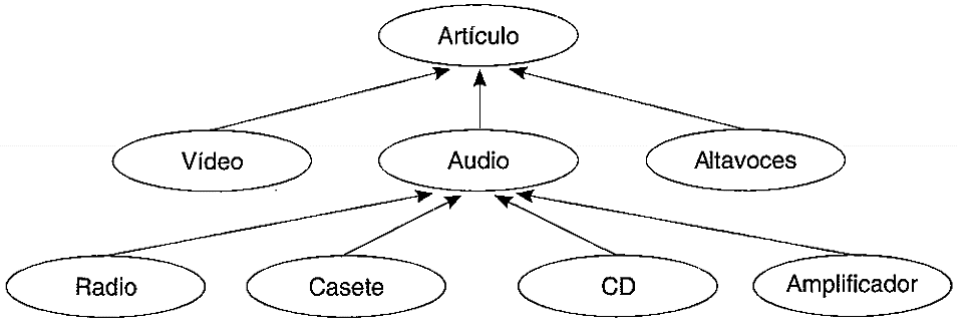


Figura 4. *Representación gráfica de una Herencia Simple.*

- **Herencia Múltiple**

  La herencia múltiple permite que cada clase pueda heredar métodos y variables de cualquier superclase.

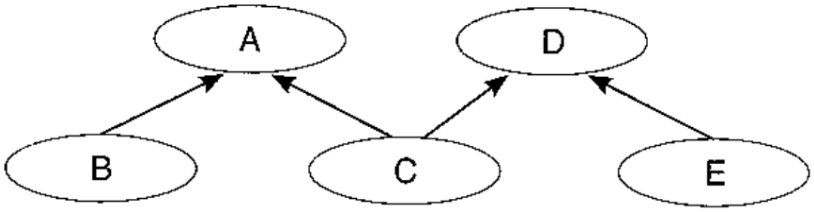


Figura 5*. Representación gráfica de una Herencia Múltiple.*

```
1  public class SuperHeroe extends Persona{
2
3      //Atributos:
4      public String nombreDeFiccion;
5
6      //Metodos:
7      public void correrMuyRapido(){
8          //TODO: implementar
9      }
10
11     public void volar(){
12         //TODO: implementar
13     }
14
15 }
```

Figura 6*. Representación lógica de una Herencia.*

1.3.2 CICLO DE VIDA DEL SOFTWARE

La complejidad del proceso de producción de software se intenta abordar mediante la descomposición en diversas etapas y que posteriormente mediante esa descomposición ha sido nombrada como Ciclo de Vida del Software. Se ha propuesto diferentes modelos de ciclo de vida donde se plantean variantes a partir de las siguientes fases principales:

- Análisis y Definición de Requisitos.
- Especificación.
- Diseño.
- Programación (escritura del código).
- Prueba e instalación.
- Operación y mantenimiento

1.3.3 GESTIÓN DE CALIDAD

Según (S. Pressman, 2002) *"Es la concordancia del software producido con los requerimientos explícitamente establecidos y con los estándares de desarrollo prefijados y con los requerimientos implícitos no establecidos formalmente, que desea el usuario"*

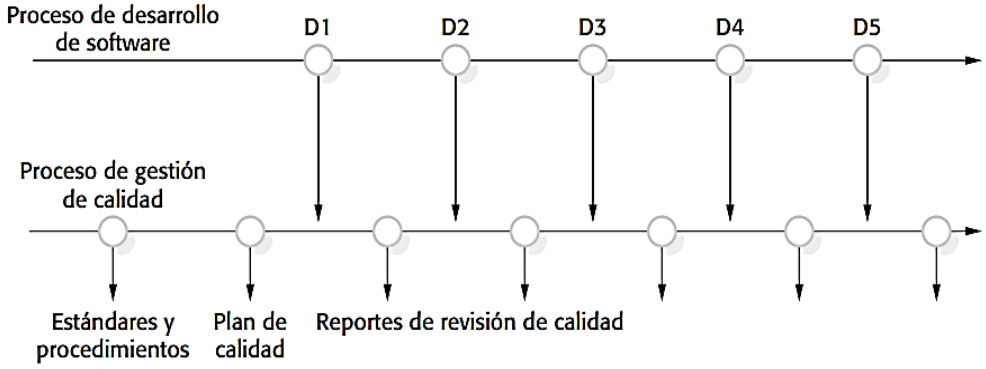


Figura 7. *Gestión de calidad y desarrollo de software.*

El desarrollo de los primeros grandes sistemas de software continuaron invadiendo la ingeniería de software en la década de los 60' pero también se iniciaron los problemas de calidad de software ya que en esa década el software era lento, poco fiable, difícil de mantener y reutilizar; esta preocupación condujo a la adopción de técnicas formales de gestión de calidad del software. La gestión de

Calidad de software implica aplicar procesos específicos de calidad (**Imagen1**) incluyendo la verificación de la continuidad de los procesos planeados; también garantiza la conformidad del proyecto mediante un plan de calidad donde se establecen metas de calidad para la definición de procesos y estándares que se usaran para el desarrollo del proyecto; cuando hablamos de la definición de procesos y estándares nos referimos al "*Aseguramiento de Calidad*"(QA), ya que tiene como objetivo asegurar la obtención de la Calidad de software.

Ya que se mencionó Aseguramiento de Calidad, también hablaremos sobre "Control de Calidad"; el control de calidad consta en la aplicación de procesos de calidad para eliminar aquellos productos que no cuentan con el nivel requerido de calidad; esta expresión no se usa mucho en la Industria de software.

- **Calidad de Software**

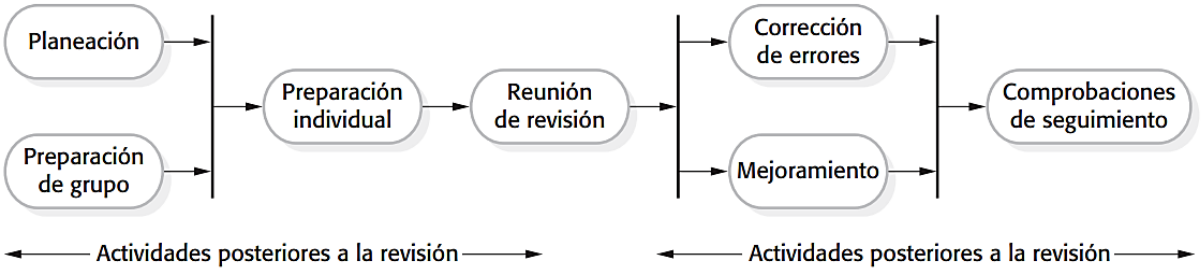
Protección	Comprensibilidad	Portabilidad
Seguridad	Comprobabilidad	Usabilidad
Fiabilidad	Adaptabilidad	Reusabilidad
Flexibilidad	Modularidad	Eficiencia
Robustez	Complejidad	Facilidad para que el usuario aprenda a utilizarlo

Figura 8. *Atributos de calidad de software*

La Calidad de software se basa en las características no funcionales, digamos que, en cuanto a la funcionalidad del software si no se obtuvo lo que se esperaba, entonces se genera la insatisfacción del usuario al recibir el producto, por ende, esto conlleva a los usuarios buscar otras formas de hacer lo que quieren o que alguien más haga exactamente lo que se desee, entonces digamos que si el software que se desarrolla no es fiable y resulta muy lento en el instante que el usuario final realiza sus peticiones, se asegura que no se logren las metas que el usuario planteo.

Sin embargo, no se trata que tan sólo la funcionalidad del software tenga la implementación correcta, sino también se depende de los atributos no funcionales declarados en el sistema, existen 15 atributos más destacados para la calidad de software, los cuales se listan en la (**Imagen2**). Según (Sommerville, 2011) "*Los atributos de calidad de software se relacionan con la confiabilidad, usabilidad, eficiencia y mantenibilidad del software*"

- **Revisión e Inspección de Software**



1.3.4 LA MANTENIBILIDAD

Según (Sommerville, 2011) Autor del libro Ingeniería de Software lo define como:

"El mantenimiento del software es el proceso general de cambiar un sistema después de que éste se entregó. Los cambios se implementan modificando los componentes del sistema existentes y agregándole nuevos componentes donde sea necesario".

Según (Rodriguez Candela, 2000) en su libro *"Fiabilidad, mantenibilidad, efectividad: un enfoque sistémico"* define la mantenibilidad como:

"Medida de la facilidad con la que una aplicación de software es modificad, o corregida cuando se detectan fallos."

La Mantenibilidad es considerada por la combinación de La Reparabilidad y la Flexibilidad; un sistema de información es reparable siempre y cuando este permita la corrección de errores detectados; también se le consideraría flexible a un software cuando este permita cambios para la breve satisfacción del requerimiento que se requiera añadir.

El cambio constante del software es muy importante ya que las organizaciones dependen totalmente de sus Sistemas de Información, si hablamos de inversión pues en su mayoría las empresas invierten millones de dólares en la implementación de un sistema de información o mejor dicho software; los cambios generan inevitablemente versiones diferentes de un sistema e incluyendo los componentes que forman del mismo, se recomienda guardar cada versión para posteriores soluciones requeridos.

La mantenibilidad cuenta con algunas leyes de Mantenimiento de software, son las siguientes:

- **Continuidad del Cambio:** Después de la supuesta culminación de desarrollo del software es entregado al usuario para su posterior uso, la frase "*supuesta culminación*" se refiere a que

justamente al entregar el producto final al usuario, este, en el preciso momento de uso del software descubre nuevas funcionalidades para su sistema de software.

✓ **<u>Tipos de Mantenimiento en ISO 14764</u>**

- **Adaptativo:** Modificación del producto software después de su distribución, para conseguir que sea utilizable en un nuevo entorno que ha cambiado o que puede cambiar en el futuro ya sea en el sistema operativo, así como en la arquitectura donde se ejecuta el producto software.
- **Correctivo:** Modificación reactiva de un producto software, es la forma más básica de mantenimiento, y consiste en la detección de defectos.

  Ejemplo: *Errores actuales detectados.*
- **Perfectivo:** Modificación de un producto software para la mejora de rendimiento o su mantenibilidad.
- **Preventivo:** Modificación de un producto software para evitar fallos efectivos mediante la detección y corrección de defectos latentes.

  Ejemplo: *Errores potenciales.*

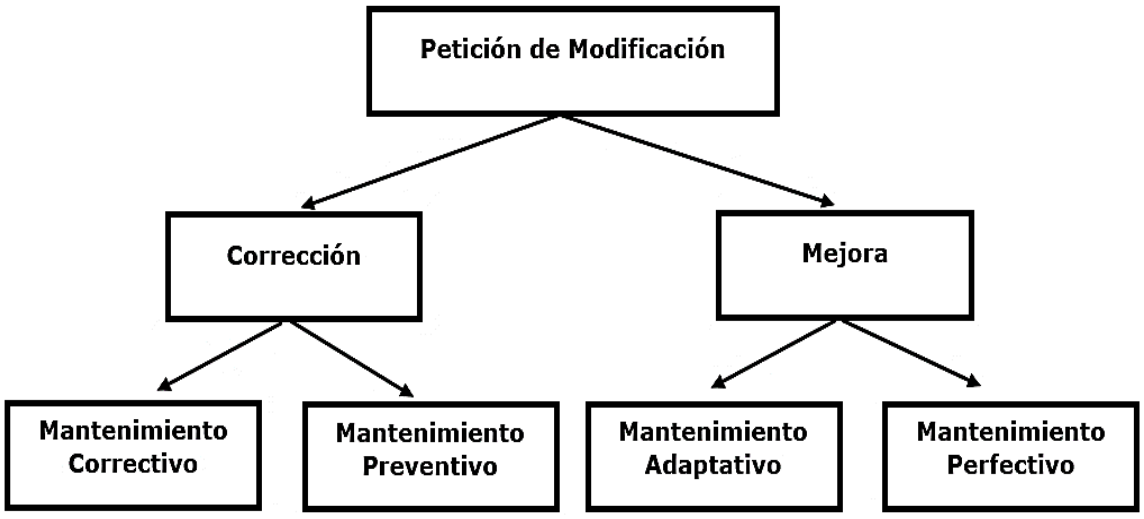


Figura 9*. Clasificación de las peticiones de modificación y tipos de mantenimiento*

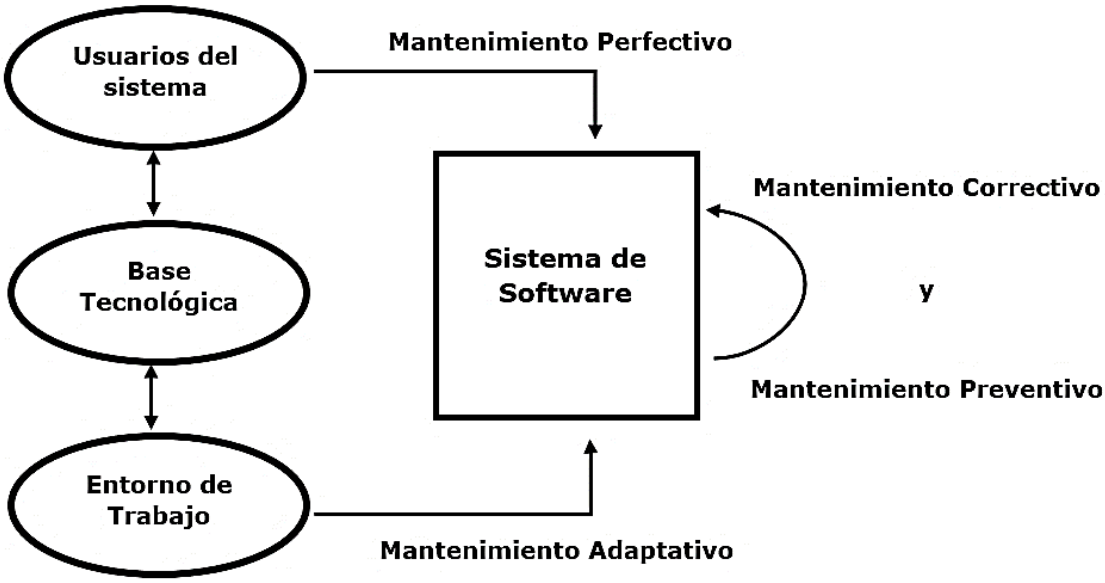


Figura 10. *Función de los tipos de mantenimiento*

- En esta investigación se aplicará Mantenimiento preventivo explicamos a que se orienta más este tipo de mantenimiento.

MANTENIMIENTO PREVENTIVO

Se ha planteado que el Mantenimiento consiste en modificar el software para que fácilmente se reutilice, este tipo de mantenimiento es preventivo debido a que mejorar la propiedad de reusabilidad del software, mejor dicho se encarga de velar por la mejor de las propiedades del software como por ejemplo aumentar la calidad de mantenimiento del sistema, se puede hacer una reestructuración para así poder mejorar la legibilidad del software, y aplicar buenas prácticas para mejorar y facilitar la comprensión del programa.

✓ **El Proceso de Mantenimiento en ISO 14764**

Solo se incluyen actividades y tareas necesarias para la modificación un producto software, las actividades que forman el proceso de mantenimiento consumen las entradas (inputs) para producir salidas (outputs); las salidas son datos u objetos que son generados por las actividades de mantenimiento.

El Proceso de Mantenimiento de software es activado siempre y cuando exista un requerimiento necesario para mantener un software. Además, se debe tener en cuenta que en cuanto el mantenimiento de software es activado, deben iniciar el desarrollo de planes incluyendo los procedimientos de mantenimiento que contienen los recursos necesarios y con disponibilidad activa. El proceso se tiene por concluido cuando un producto software es retirado completamente.

.

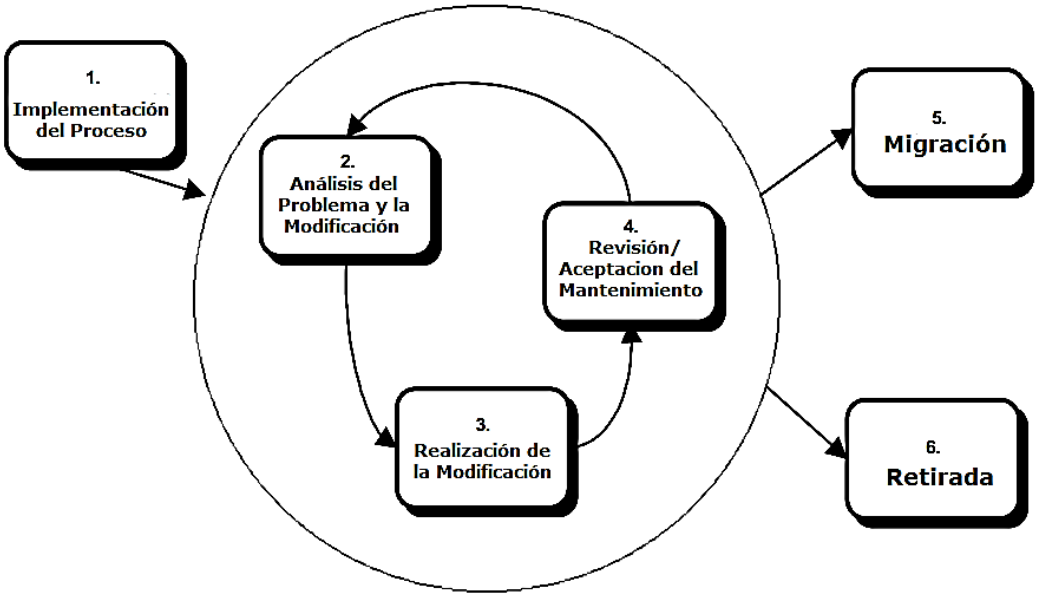


***Figura 11**. Actividades del proceso de Mantenimiento de Software*

1.3.5 PRUEBAS DE SOFTWARE

El fin específico de las pruebas es demostrar lo que un programa hace lo que se intenta que haga, así como detectar defectos antes de su uso; cuando probamos el software, la ejecución se realiza con datos artificiales para posteriormente verificar los resultados de las pruebas que se realiza para identificar errores o información de atributos no funcionales; el proceso de pruebas contiene dos metas diferentes:

1. Exhibir al desarrollador e incluso al cliente que el software cumple con los requerimientos, esto indica que en la documentación de requerimientos debe contener por lo menos, una prueba por cada requerimiento recolectado.
2. Localizar comportamientos incorrectos o indeseables del software; puede que en diferentes situaciones no haya concordancia con las especificaciones ya que tales situaciones son consecuencias de la deficiencia del software.

Las pruebas son consideradas como parte del proceso de verificación y validación del software, usualmente los conceptos de verificación y validación son confundidos frecuentemente, sin embargo (W. Boehm, 1979) pionero de la Ingeniería de Software expreso la diferencia de la siguiente manera:

- *"Validación; ¿construimos el producto correcto?"*
- *"Verificación; ¿construimos bien el producto?"*

Estos procesos buscan comprobar que el software con planificaciones de desarrollo cumpla son las especificaciones para así poder brindar la funcionalidad deseada por las personas que solicitan el software.

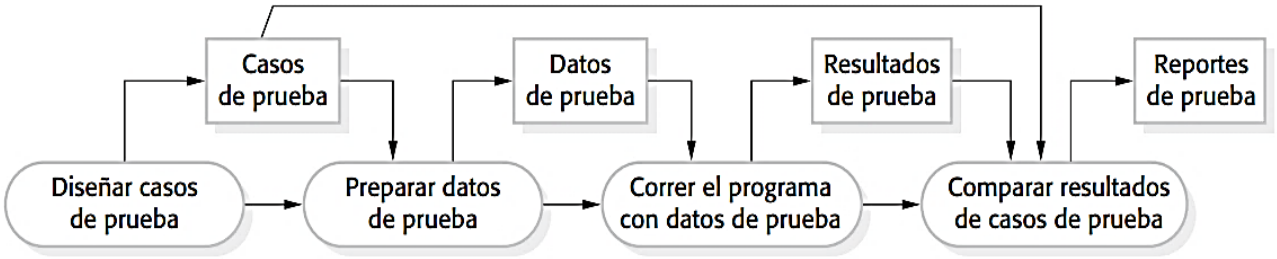


Figura 12. *Modelo de proceso de pruebas del software*

1.3.5.1 Pruebas de Desarrollo

En las pruebas de desarrollo se incluyen todas las actividades de pruebas que aplican los equipos que forman parte del desarrollo del sistema; estas pruebas de desarrollo consisten en identificar bugs en el software, por ende, constantemente están entrelazadas con la depuración ya que este es un proceso donde se localizan problemas en el código para posteriormente corregirlos.

Durante el desarrollo las pruebas se realizan en tres niveles:

1. **Pruebas de Unidad:** se enfocan en la comprobación de la funcionalidad de objetos y métodos.
2. **Pruebas del componente:** se enfoca en comprobar interfaces del componente.
3. **Pruebas del sistema:** se enfocan en probar las interacciones de los componentes.

En este caso hablaremos sobre las pruebas del Sistema ya que es lo que nos importa más.

PRUEBAS DEL SISTEMA

Las pruebas del sistema se enfocan en probar las interacciones de los componentes y objetos que conforman el sistema, además de la interacción debe descubrir los bugs, encontrar interpretaciones erróneas que son cometidas por los desarrolladores. Debido al enfoque en las interacciones, las pruebas que son basadas en casos son bastante efectivo para las pruebas del sistema, el probar los casos de uso obliga a que se realicen diferentes interacciones.

Para poder diseñar casos de pruebas que sean específicos y necesarios se hace uso de los diagramas de secuencia, ya que es donde se muestra las entradas requeridas y las salidas creadas.

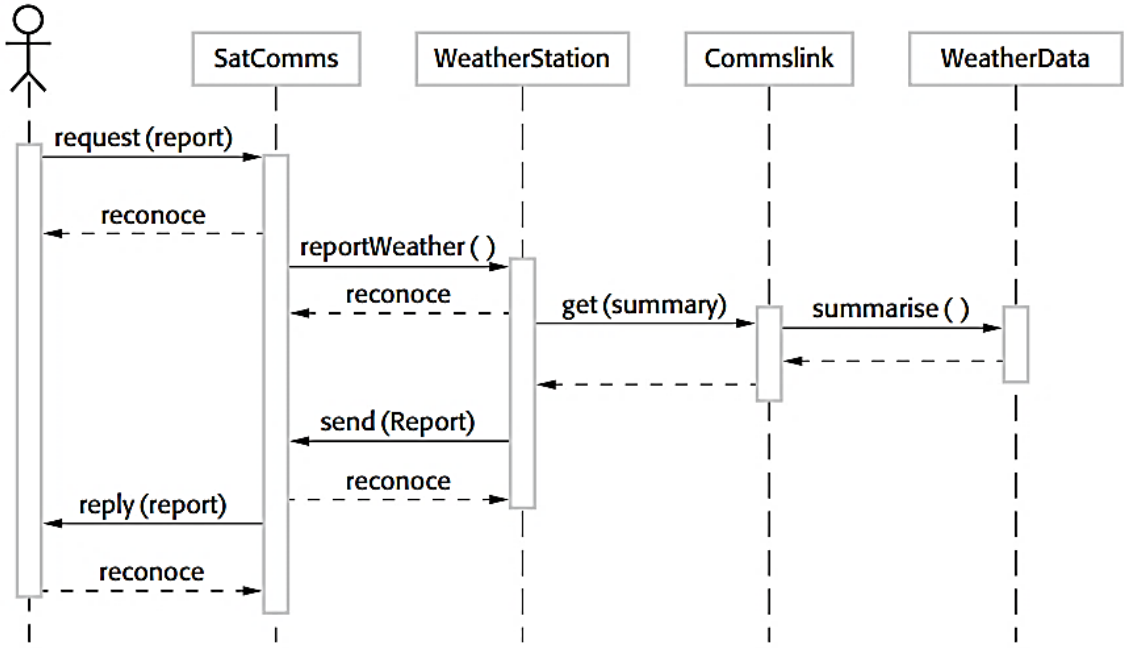


Figura 13. *Diagrama de secuencia*

Según (Sommerville, 2011) describe el diagrama de la siguiente manera:

- ✓ *"Una entrada de una petición para un reporte tiene que contar con reconocimiento asociado. En última instancia, a partir de la petición debe regresarse un reporte. Durante las pruebas, se debe crear un resumen de datos que sirva para comprobar que el reporte se organiza correctamente."*
- ✓ *"Una petición de entrada para un reporte a WeatherStation da como resultado la generación de un reporte resumido. Usted puede probar esto en aislamiento, creando datos brutos correspondientes al resumen que preparó para la prueba de SatComms, y demostrar que el objeto WeatherStation produce este resumen. Tales datos brutos se usan también para probar el objeto WeatherData*."

1.3.6 REFACTORIZACIÓN

Según (Fowler, 1999) Autores del Libro “Refactoring, *Mejorar el diseño de código existente.*” define la refactorización de la siguiente manera:

“Refactorizar es una técnica disciplinada para reestructurar una porción de código, alterando su estructura interna sin modificar su comportamiento externo”.

Refactorización es el nombre que se da a una colección de pequeñas técnicas que funcionan independientemente y tienen como objetivo mejorar el código o inclusive se le puede considerar como limpieza del código “*Limpiar Código*”, esta singular palabra Refactorizar vela por la buena salud del código fuente. No obstante, en la Ingeniería de software se usa a menudo para la descripción del proceso de modificación en el código fuente, también es considerada como parte del proceso de Desarrollo y Mantenimiento del software, además permite tomar diseños defectuosos de código mal escrito (complejidad innecesaria, duplicidad, etc.) para luego adaptarlo a uno bueno y más organizado.

Figura 14. *Representación gráfica de Refactoring*

Los programadores alternan la inserción de funcionalidades nuevas además de casos de pruebas con refactorización para mejorar la consistencia interna y la claridad del código.

Cualquier momento es el adecuado para agregar nuevos conocimientos al código fuente, pero se debe tener en cuenta unas pequeñas recomendaciones antes de empezar a Refactorizar el código:

- ***Si parte del código funciona, no debe ser tocado;*** algunas veces el desarrollador quiere hacer unos retoques al código que funciona perfectamente pero no se da cuenta del tiempo perdido que ocasiona hacer esa tarea, eso sí es una pérdida de tiempo muy lamentable.
- ***Uso riguroso de las pruebas***; No se puede comenzar un proceso de refactorización si no se ha realizado una previa Planificación de pruebas. Las pruebas son necesarias debido a que permiten comprobar que una vez refactorizado el software mantiene su funcionalidad inicial.
- ***Usar los bad smells ("malos olores")***; Los bad smells ayudan a detectar fácilmente fragmentos de código dónde se puede aplicar refactorizaciones.

Las ventajas de la Refactorización son múltiples ya sea a nivel de aplicación, así como también a nivel personal; ya que la refactorización conlleva a replanteamiento de un problema, el enfrentamiento ante algo que ya funciona pero que puede ser mejorado según la experiencia del desarrollador.

1.3.7 TÉCNICAS DE REFACTORIZACIÓN

a. Extraer método (Extract Method)

Según (Fowler, 1999) es una de las técnicas de refactorización más común y fácil que conlleva a varios beneficios; como la legibilidad del código y reducir las suposiciones cuando deseen leer el código fuente.

```
 1: public class Receipt
 2: {
 3:     private IList<decimal> Discounts { get; set; }
 4:     private IList<decimal> ItemTotals { get; set; }
 5:
 6:     public decimal CalculateGrandTotal()
 7:     {
 8:         decimal subTotal = 0m;
 9:         foreach (decimal itemTotal in ItemTotals)
10:             subTotal += itemTotal;
11:
12:         if (Discounts.Count > 0)
13:         {
14:             foreach (decimal discount in Discounts)
15:                 subTotal -= discount;
16:         }
17:
18:         decimal tax = subTotal * 0.065m;
19:
20:         subTotal += tax;
21:
22:         return subTotal;
23:     }
24: }
```

Se puede apreciar que el método ***CalculateGrandTotal*** se realizan tres cosas diferentes: calcular el subtotal, descuento y cálculo del impuesto. Entonces si nosotros queremos ahorrar tiempo y aumentar legibilidad se podría separar en distintos métodos cada tarea, de la siguiente manera:

```
 1: public class Receipt
 2: {
 3:     private IList<decimal> Discounts { get; set; }
 4:     private IList<decimal> ItemTotals { get; set; }
 5:
 6:     public decimal CalculateGrandTotal()
 7:     {
 8:         decimal subTotal = CalculateSubTotal();
 9:
10:         subTotal = CalculateDiscounts(subTotal);
11:
12:         subTotal = CalculateTax(subTotal);
13:
14:         return subTotal;
15:     }
16:
17:     private decimal CalculateTax(decimal subTotal)
18:     {
19:         decimal tax = subTotal * 0.065m;
20:
21:         subTotal += tax;
22:         return subTotal;
```

b. <u>Extract Interface</u>

Cuando se observa más de una clase usando un subconjunto similar de los métodos de otra clase.

```
 1: public class ClassRegistration
 2: {
 3:     public void Create()
 4:     {
 5:         // create registration code
 6:     }
 7:
 8:     public void Transfer()
 9:     {
10:         // class transfer code
11:     }
12:
13:     public decimal Total { get; private set; }
14: }
15:
16: public class RegistrationProcessor
17: {
18:     public decimal ProcessRegistration(ClassRegistration registration)
19:     {
20:         registration.Create();
21:         return registration.Total;
22:     }
23: }
```

se extraerán los métodos que utilizan los consumidores y los colocó en una interfaz. Ahora los consumidores no les importa saber acerca de la clase que implementa estos métodos. Hemos desacoplado nuestro consumidor de la implementación real y depende sólo de contrato que hemos creado.

```
 1: public interface IClassRegistration
 2: {
 3:     void Create();
 4:     decimal Total { get; }
 5: }
 6:
 7: public class ClassRegistration : IClassRegistration
 8: {
 9:     public void Create()
10:     {
11:         // create registration code
12:     }
13:
14:     public void Transfer()
15:     {
16:         // class transfer code
17:     }
18:
19:     public decimal Total { get; private set; }
20: }
21:
22: public class RegistrationProcessor
23: {
24:     public decimal ProcessRegistration(IClassRegistration registration)
25:     {
26:         registration.Create();
```

1.3.8 MANTEMA: METODOLOGÍA PARA MANTENIMIENTO DE SOFTWARE

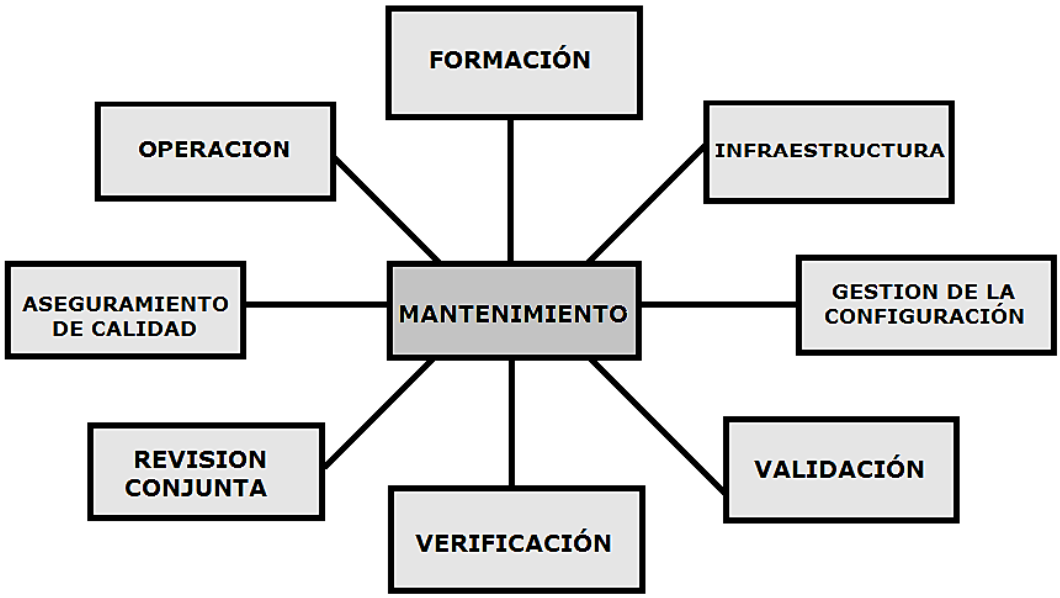
Los desarrolladores tienen en cuenta que el mantenimiento es el conjunto de tareas donde se realizan cambios a un determinado software ya sea eliminando, añadiendo o modificando sentencias del código fuente para una posterior mejora del software, es probable que los directivos de las empresas en las que trabajan perciben este proceso como una de las bases más importantes del Negocio, entonces se opta por un absoluto control y su continua mejora.

La Metodología MANTEMA muestra la visión del proceso de mantenibilidad desde un nivel de abstracción mayor, sin embargo, los niveles de probabilidad sobre el interés del contenido de instrucciones, campos de los ficheros son muy bajos, ya que se dirige más a las mejores técnicas para el entendimiento y mejora del Software. El proceso se puede ver como un conjunto de todas las operaciones que son necesarias para aplicar sobre el software y así poder implementar las modificaciones solicitadas, sin embargo para

poder asignar este conjunto de operaciones de una base metodológica, se define con anterioridad el proceso de Mantenimiento, donde se encuentre detallado lo que se debe realizar, cuando, cómo y por quien, de tal forma que para cada intervención que sea ejecutada durante el mantenimiento sea conforme a la instancia definida en el proceso de Mantenimiento.

ESTRUCTURA GENERAL DE LA METODOLOGIA

MANTEMA contiene procesos satélites al mantenimiento.



ESTRUCTURA DETALLADA DE LA METODOLOGIA

ACTIVIDADES Y TAREAS INICIALES COMUNES

	Actividades y tareas iniciales comunes (1 de 2)					
	Estudio inicial			Planificación del proceso		
	I0.1 Iniciar y recoger información	I0.2 Preparar propuesta de mantenimiento	I0.3 Contrato	I1.1 Planificar calendario y responsables	I1.2 Adquirir conocimiento de la aplicación	I1.3 Desarrollar planes
Entradas	Solicitud de prestación del servicio de mantenimiento	Cuestionario inicial Entrevistas	Propuesta de mantenimiento	Contrato de mantenimiento (DOC3)	Producto software en operación o desarrollo	Producto software en operación Lista de elementos software Nueva documentación del aplicativo
Salidas	Cuestionario inicial (DOC1)	Propuesta de mantenimiento (DOC2) Documento de riesgos (DOC4)	Contrato de mantenimiento (DOC3)	Listado de responsables, calendario de reuniones, etc..	Lista de elementos software Nueva documentación del aplicativo Contrato definitivo de mantenimiento (DOC3)	Resumen técnico (DOC5) Plan de mantenimiento del periodo
Técnicas	Entrevista	Identificación y estimación de riesgos (sección 4.3.1, página 127)			Estudio de la documentación Observación y entrevistas Instalación de herramientas Ingeniería inversa	Planificación Estimación de recursos para mantenimiento no planificable (sección 0, página 129)
Responsable	Equipo de mantenimiento Organización del sistema	Responsable de mantenimiento	Organización del sistema Responsable de mantenimiento	Organización del sistema Responsable de mantenimiento	Equipo de mantenimiento Organización del sistema Usuario	Equipo de mantenimiento
Interfaces con otros procesos			Aseguramiento de la calidad			
	Pueden omitirse si no se externaliza					

	Actividades y tareas iniciales comunes (2 de 2)				
	Planificación del proceso			Análisis de la petición de modificación	
	I1.4 Definir procedimientos de petición de modificación	I1.5 Implementar proceso de G.C.S.	I1.6 Preparar entornos de pruebas	I2.1 Recibir petición de modificación	I2.2 Decidir tipo de mantenimiento
Entradas	Plan de mantenimiento	Sistema en explotación	Elementos software del sistema en operación	Petición de modificación (DOC6)	Petición de modificación recibida
Salidas	Modelos de documentos Procedimientos	Proceso de gestión de la configuración	Copias de los elementos software en operación	Petición de modificación recibida	Informe sobre el tipo de mantenimiento necesario, su criticidad, etc. Decisión sobre el tipo de mantenimiento a aplicar
Técnicas					Evaluación de impacto
Responsable	Equipo de mantenimiento Organización del sistema Solicitante	Equipo de mantenimiento	Equipo de mantenimiento	Gestor de peticiones Usuario Solicitante	Gestor de peticiones Planificador
Interfaces con otros procesos	Gestión de la Configuración	Gestión de la Configuración	Gestión de la Configuración		

ANÁLISIS DE PETICIÓN DE MODIFICACIÓN

MANTENIMIENTO PLANIFICABLE

Puesto que no todas las actividades del mantenimiento panificable son aplicables a todos los tipos de mantenimiento que hemos englobado bajo esta denominación (correctivo no urgente, perfectivo, preventivo y adaptativo), ya que conservan algunas diferencias, indicamos con las siguientes claves los tipos de mantenimiento a los que cada tarea es aplicable.

Claves:

- ✓ **CP** = mantenimientos correctivo no urgente y perfectivo.
- ✓ **A** = Adaptativo
- ✓ **P** = Preventivo

PROCESO DE MANTENIMIENTO PLANIFICABLE

	Actividades y tareas del mantenimiento planificable (1 de 3)					
	Análisis de la petición			Intervención y pruebas		
	CP/P	CP/P	CP	A	A	CP/P/A
	P1.1 Valorar petición	P1.2 Documentar posibles soluciones	P1.3 Elegir alternativa adecuada	P2.1 Planificar calendario	P2.2 Realizar copia del producto software	P2.3 Ejecutar intervención
Entradas	Producto software en explotación. Petición de modificación (DOC6).	Producto software en explotación Petición de modificación en espera	Producto software en explotación Alternativas de implementación (DOC10)	Documentación del proyecto	Producto software en explotación	CP: Producto software en explotación Diagnóstico del error (DOC9) Alternativa seleccionada P: Producto software en explotación Lista de elementos software y propiedades mejorables (DOC12) A: Copia del producto software
Salidas	Petición de modificación en espera Calendario de intervención P: Lista de elementos software y propiedades mejorables (DOC12)	C P: Diagnóstico del error y posibles soluciones (DOC9) Alternativas de implementación (DOC10) Medidas del producto (DOC16a) P: Lista de elementos software y propiedades mejorables (DOC12) Medidas del producto software en explotación (DOC16a)	Alternativa seleccionada (DOC9 completo)	Calendario de intervención Estimación de plazos Disponibilidad de recursos	Copia del producto software	CP: Producto software corregido Documento de acciones correctivas/perfectivas P: Producto software perfeccionado Documento de acciones preventivas (DOC13) A: Copia adaptada
Técnicas	Estimación de esfuerzos (sección 3.5, página 79) Gestión de proyectos P: Análisis de cartera (Sneed, 1995)	Ingeniería inversa Análisis de la documentación del proyecto	Consulta a la base de datos histórica	Gestión de proyectos Métricas CC para mantto. adaptativo (sección 4.4.2, página 161)		Reingeniería Reestructuración Codificación Redocumentación
Responsable	Equipo de mantenimiento	Equipo de mantenimiento	Equipo de mantenimiento	Equipo de mantenimiento	Equipo de mantenimiento	Equipo de mantenimiento
Interf. con otros proc		Aseguramiento de la calidad	Aseguramiento de la calidad		Gestión de la configuración	Aseguramiento de la calidad

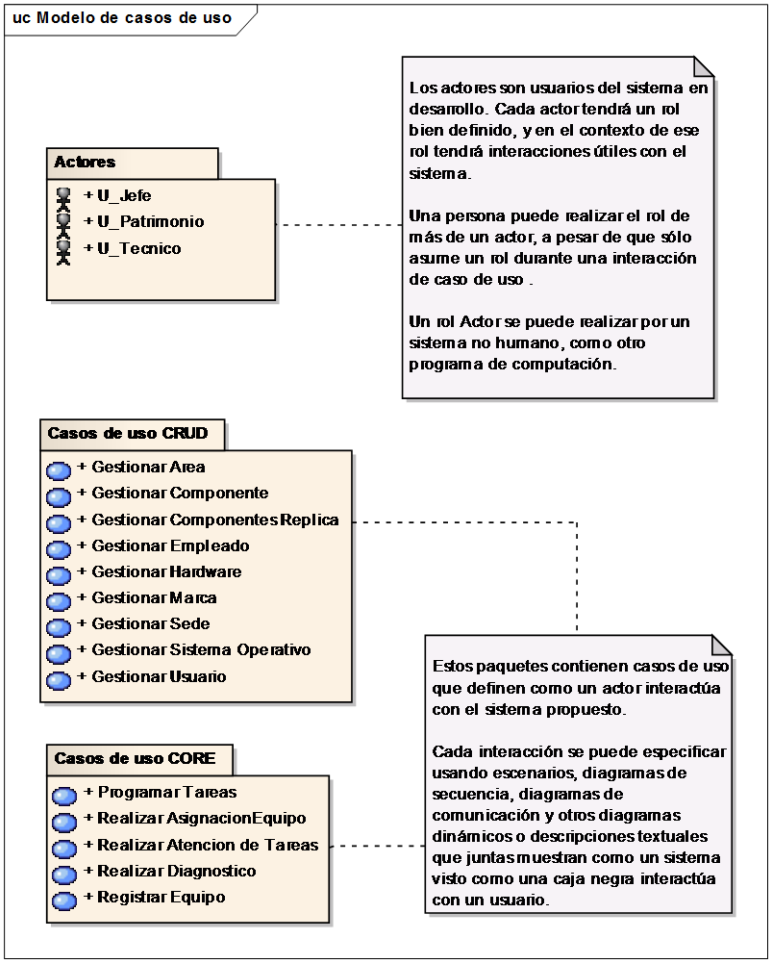
	Actividades y tareas del mantenimiento planificable (2 de 3)					
	Intervención y pruebas					
	CP/P/A	CP/P/A	CP/P/A	CP/P/A	CP/P/A	CP/P/A
	P2.4 Ejecutar pruebas unitarias	P2.5 Ejecutar pruebas de integración	P2.6 Ejecutar paralelamente en software antiguo y nuevo	P2.7 Verificar y validar corrección con el cliente	P2.8 Redocumentar manual de usuario	P2.9 Pasar a producción
Entradas	Producto intervenido (corregido, perfeccionado o copia adaptada) Documento de intervención (de acciones correctivas, perfectivas, preventivas o adaptativas: DOC7/11/13) Petición de modificación (DOC6)	Producto intervenido comprobado unitariamente Petición de modificación (DOC6)	Producto intervenido comprobado	Documento de intervención (de acciones correctivas, perfectivas, preventivas o adaptativas: DOC7/11/13) Elementos software corregidos y probados Documentación con las pruebas realizadas (DOC8)	Producto software totalmente comprobado Manual del usuario	Producto software totalmente comprobado
Salidas	Producto intervenido comprobado unitariamente Documento con las pruebas unitarias realizadas (DOC8)	Producto intervenido comprobado Documento con las pruebas de integración realizadas	Producto intervenido comprobado y en correcto funcionamiento Medidas del producto (DOC16b) A Documentación actualizada	Documento de aceptación de la corrección validado por el cliente Producto software totalmente comprobado	Manual de usuario redocumentado	Producto software totalmente comprobado en explotación con la Petición de Modificación servida
Técnicas	Técnicas de prueba	Técnicas de prueba	Realización de las operaciones reales en las versiones antigua y nueva del producto software Técnicas de no regresión			
Responsable	Equipo de mantenimiento	Equipo de mantenimiento	Equipo de mantenimiento Usuario	Equipo de mantenimiento Solicitante Usuario	Equipo de mantenimiento	Equipo de mantenimiento
Interf. con otros proc	Aseguramiento de la calidad Validación Verificación	Aseguramiento de la calidad Validación Verificación	Operación	Revisión conjunta		Gestión de la configuración Infraestructura

	Actividades y tareas del mantenimiento planificable (3 de 3)	
	Cierre de intervención	
	CP/A	A (sólo Adaptativo)
	P2.10 Realizar revisión	P3.1 Archivar datos del producto inicial
Entradas	Producto software en explotación con la Petición de Modificación servida	Producto software inicial Datos relacionados con el producto software inicial
Salidas	Producto software en explotación con la Petición de Modificación servida revisado	Producto software inicial archivado Datos del producto software antiguo archivados
Técnicas		
Responsable	Equipo de mantenimiento	Equipo de mantenimiento
Interfaces con otros procesos		Gestión de la configuración

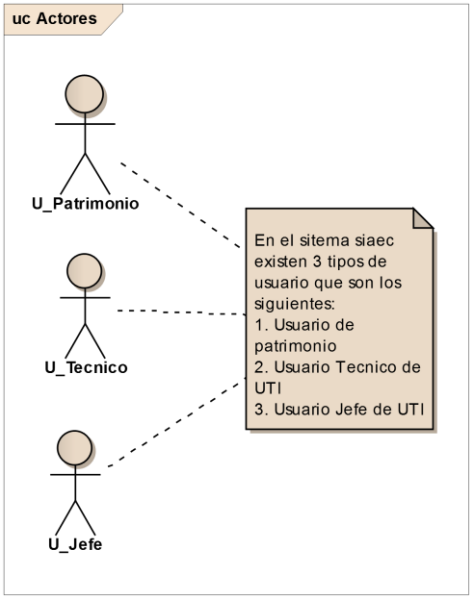
1.3.9 SISTEMA PROTOTIPO SIAEC

SIAEC: Sistema de Administración de Equipos de Computo

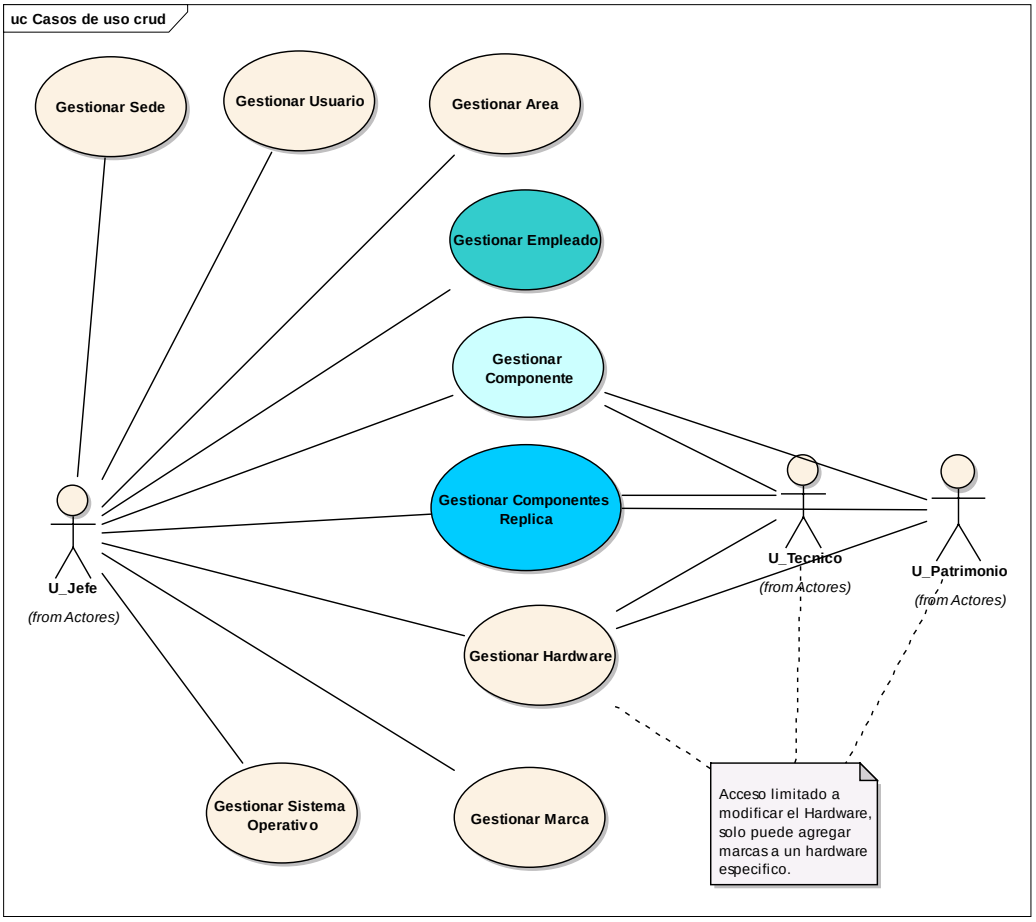
MODELO DE CASOS DE USO



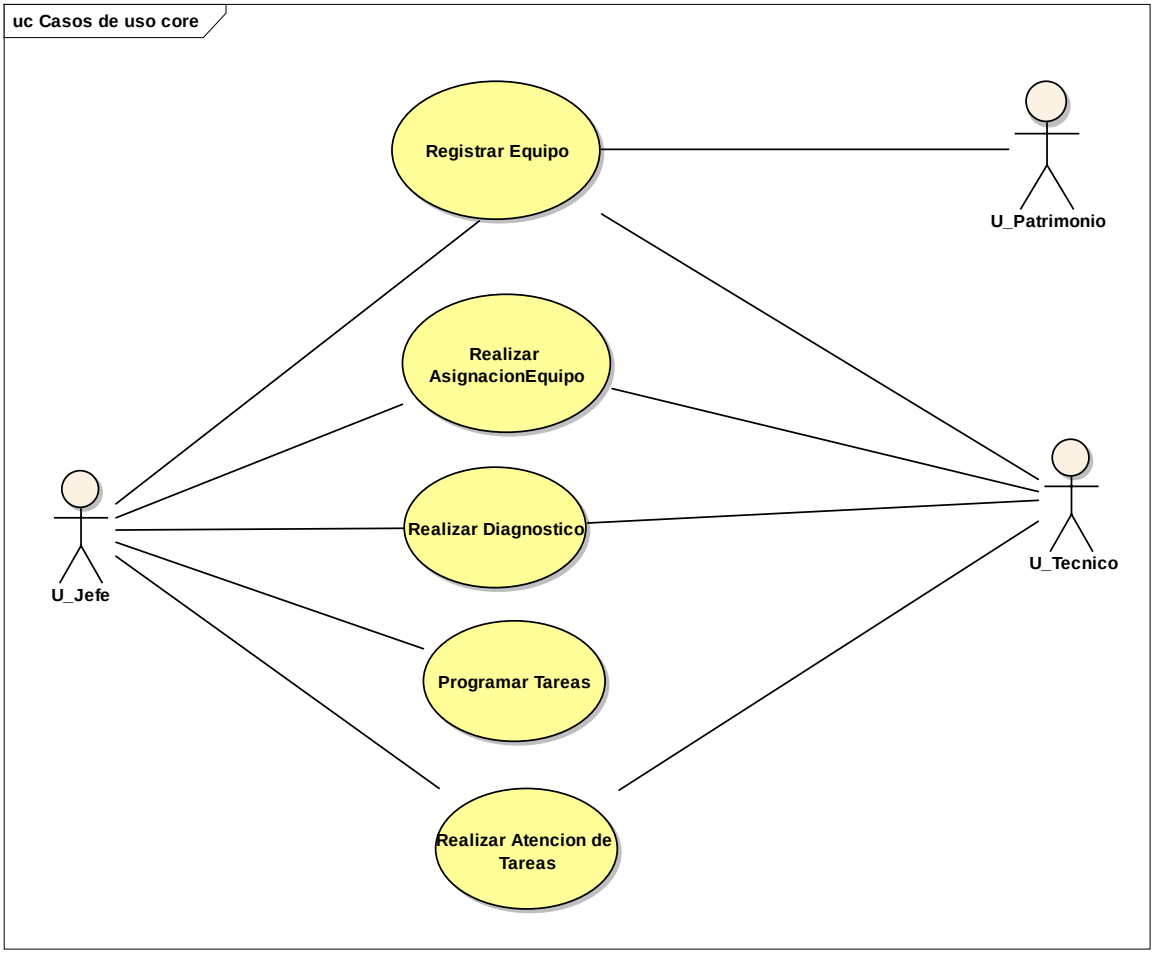
ACTORES



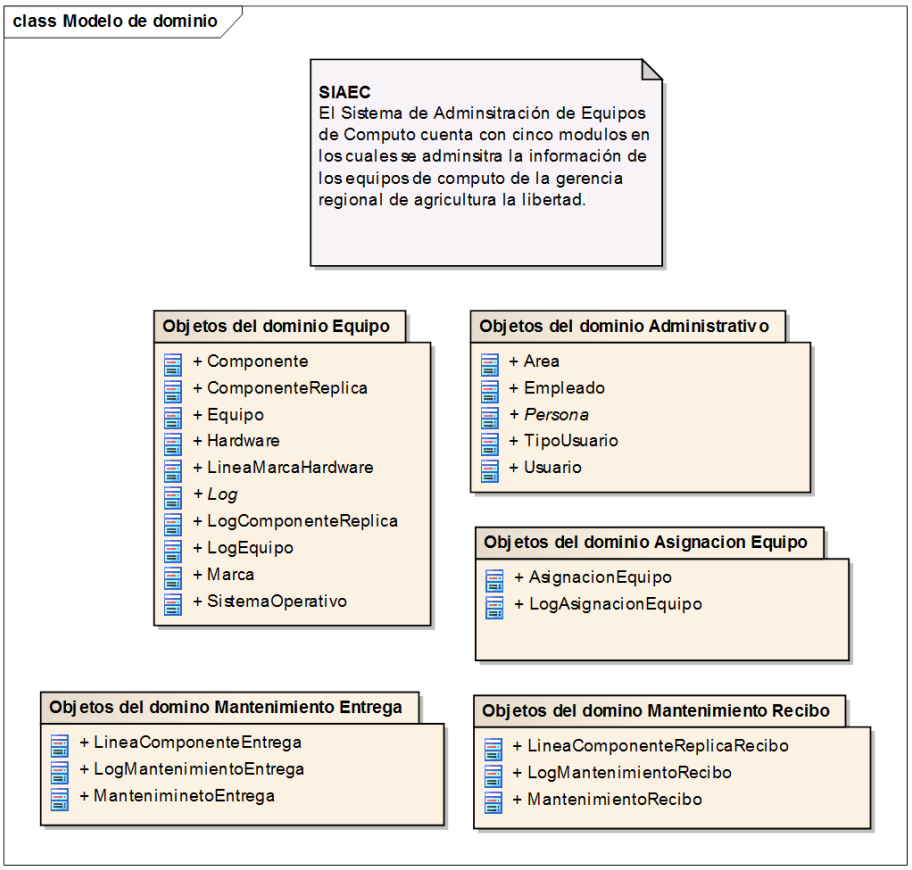
MODELO CASOS DE USO – CRUD



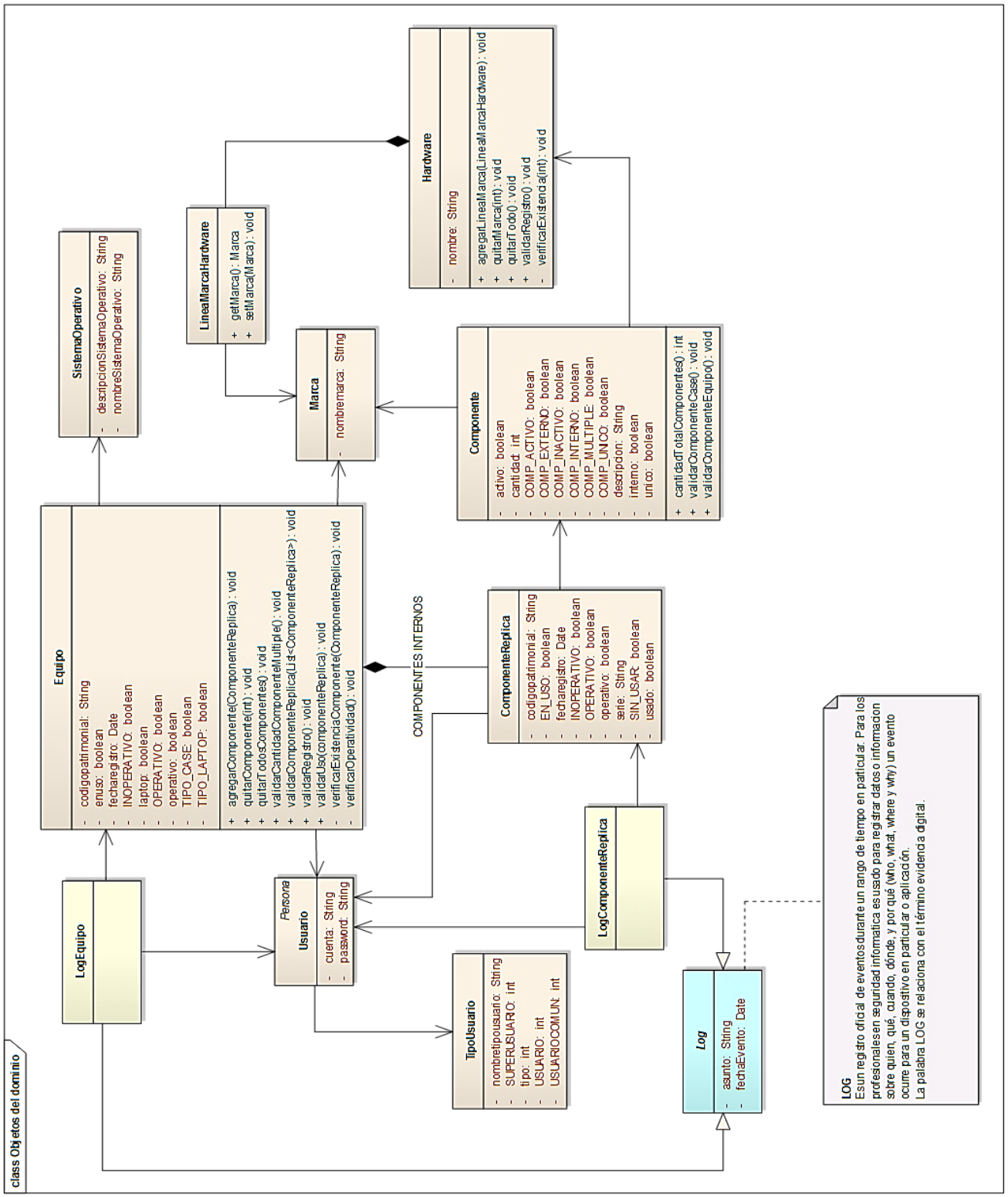
MODELO DE CASOS DE USO – CORE



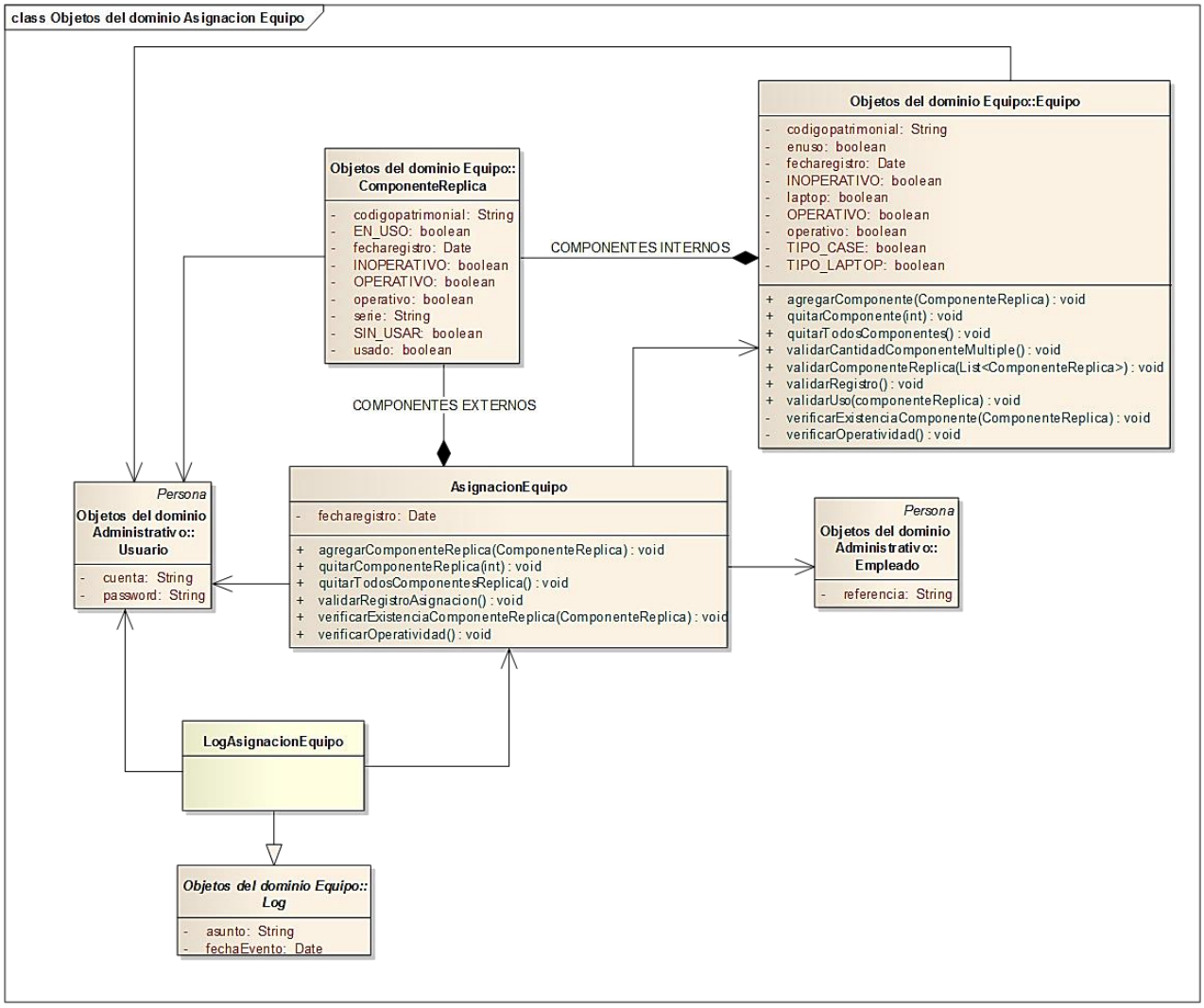
MODELOS DE DOMINIO EN GENERAL



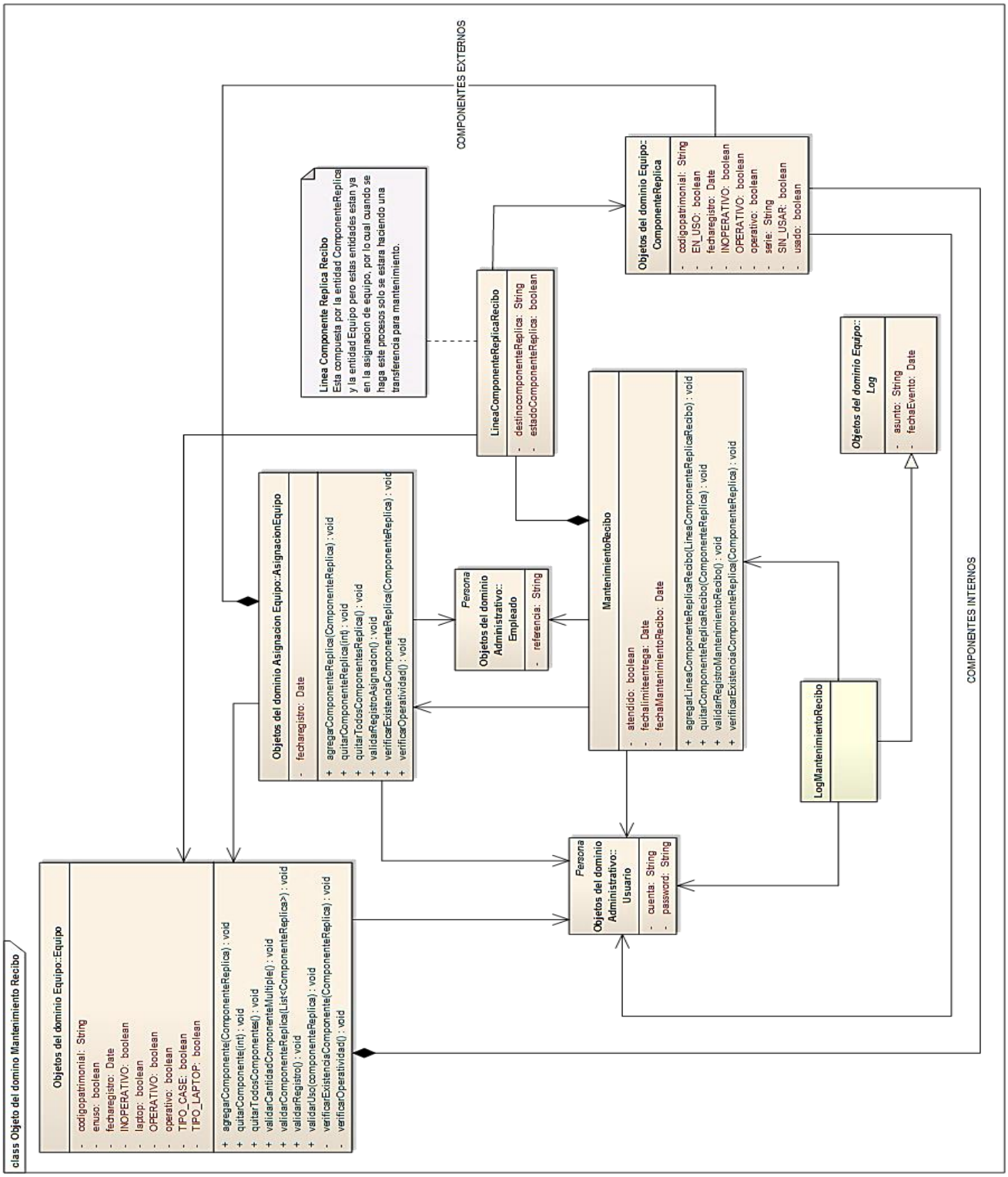
MODELO DE DOMINIO GESTIONAR EQUIPOS



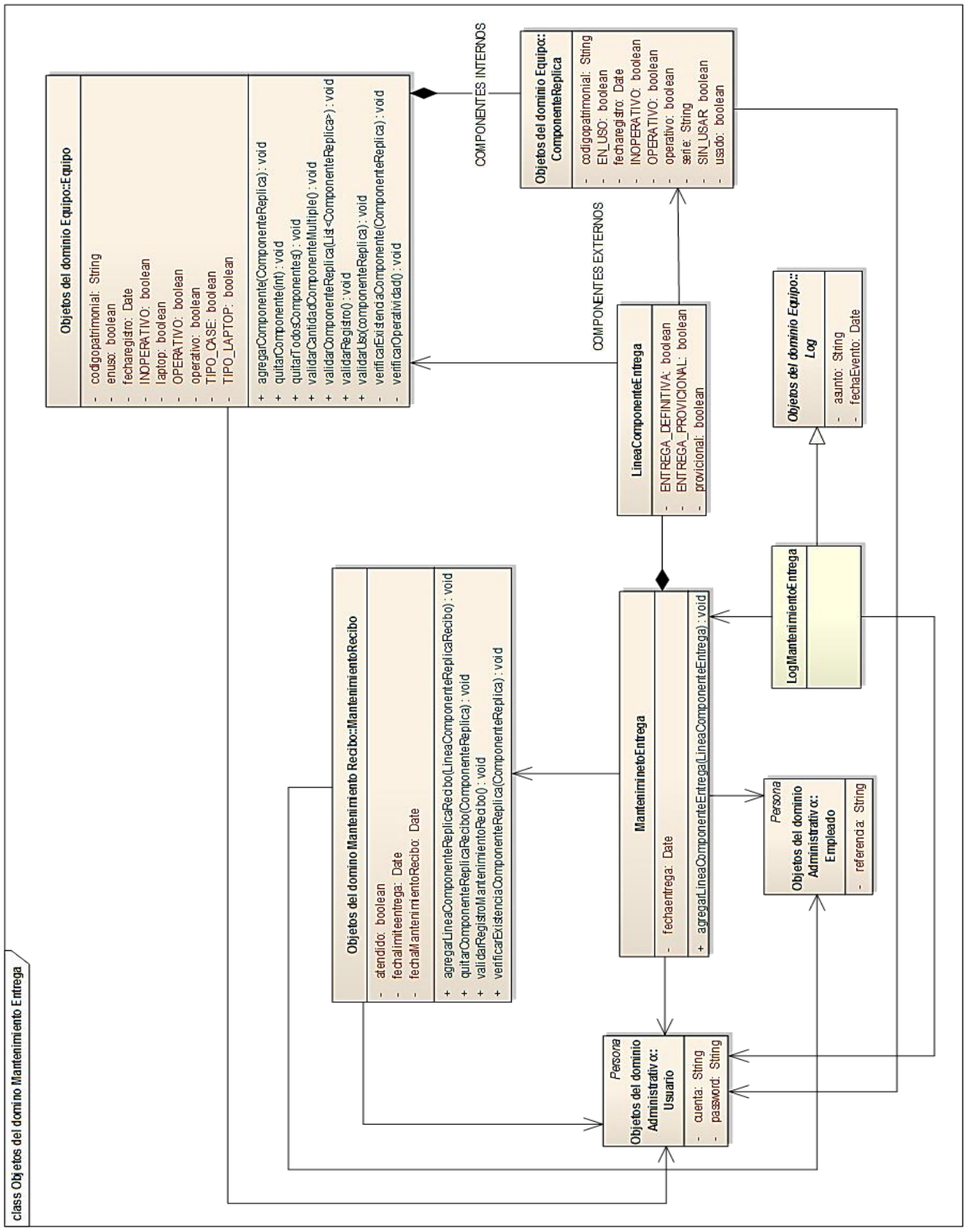
MODELO DE DOMINIO ASIGNAR EQUIPO



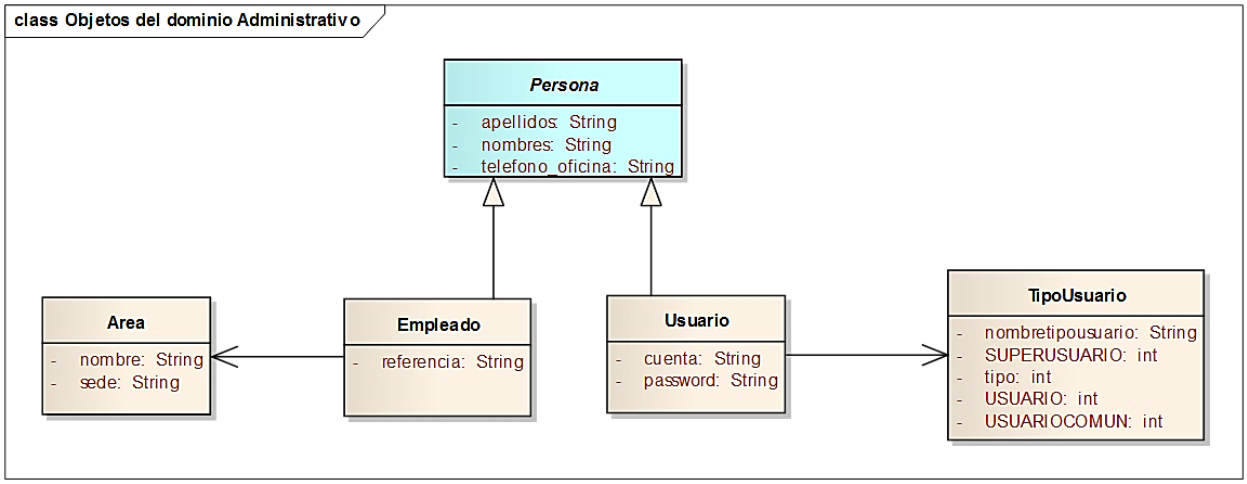
MODELO DE DOMINIO DE MANTENIMIENTO DE RECIBO



MODELO DE DOMINIO DE MANTENIMIENTO ENTREGA



MODELO DEL DOMINIO ADMINISTRATIVO



1.4 Formulación del Problema.

¿Cómo la aplicación de técnicas de refactorización garantiza mayor mantenibilidad del sistema prototipo SIAEC en su proceso de evolución?

1.5 Justificación del Estudio

La presente investigación se justifica de manera operativa por lo siguiente, al momento de obtener el prototipo SIAEC como segunda versión, donde ya se han aplicado las técnicas necesarias de Refactorización, su finalidad será de que cada vez que el desarrollador u otra persona como programador trate de agregar los nuevos requerimientos que el cliente necesite nuevamente, entender la fuente de código del software ya no será problema, debido a que el código estará en perfecta reestructuración u organización, por ende la Mantenibilidad será viable y satisfactorio.

Se puede decir que basado en tecnologías, el prototipo SIAEC propuesto como segunda versión usara la primera versión del Sistema SIAEC donde se implementa el módulo administrativo.

Económicamente hablando, para esta investigación no incluye algún tipo de inversión, por lo tanto, el proyecto de investigación es factible.

En lo Social es donde el proyecto abarca más, ya que uno de los objetivos de la refactorización es que el Programador adopte buenas prácticas en el proceso de Desarrollo, pero aplicar técnicas de refactorización ayudara más que todo, al entendimiento rápido del Código Fuente así evitar las pérdidas de tiempo, además de garantizar que el software es altamente Mantenible y de Calidad.

1.6 Hipótesis

La aplicación de técnicas de refactorización permite garantizar significativamente una mayor mantenibilidad del sistema prototipo SIAEC en su proceso de evolución.

1.7 Objetivos.

1.7.1 Objetivo General

Garantizar mayor mantenibilidad del sistema prototipo SIAEC en su proceso de evolución con la aplicación de técnicas de refactorización.

1.7.2 Objetivos Específicos

- ✓ Disminuir la densidad de defectos del sistema prototipo SIAEC.
- ✓ Reducir la Complejidad del sistema prototipo SIAEC.
- ✓ Incrementar la Comprensibilidad del sistema prototipo SIAEC.

CAPITULO II

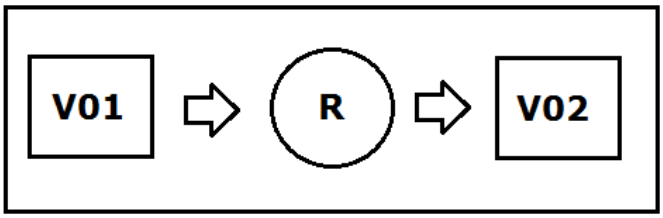
MÉTODO

II. MÉTODO

2.1 Diseño de Investigación

- ✓ Diseño Experimental Pre-Experimental

  Se aplica este tipo de Diseño de acuerdo a la técnica de contrastación, ya que se aplica el método Pre-test y Post-Test.



- **V01:** Indica el Sistema prototipo SIAEC en una primera versión, donde se aplicó el proceso de pruebas para la verificación de su funcionamiento.
- **R:** Indica las Técnicas de Refactorización a aplicar para la mejora significativa de la mantenibilidad del sistema prototipo SIAEC durante su proceso de evolución.
- **V02:** Indica el Sistema Prototipo SIAEC en una segunda versión, donde se muestra las mejoras que se realizaron mediante la refactorización.

2.2 Variables de Operacionalización

- ✓ **Variable Independiente:** Técnicas de Refactorización
- ✓ **Variable Dependiente:** La Mantenibilidad
- ✓ **Unidad de Estudio:** Sistema Prototipo SIAEC

VARIABLE	DEFINICIÓN CONCEPTUAL	DEFINICIÓN OPERACIONAL	DIMENSION	INDICADORES	ESCALA DE MEDICIÓN
Técnicas de Refactorización	**"***Técnica disciplinada para la reestructuración de un organismo de código existente, alterando su estructura interna sin cambiar su comportamiento externo".* (Fowler, 1999)	*"Técnicas detalladas de transformaciones de código, cada refactorización aplicada permite que el software tenga facilidad de cambio en el código fuente, también facilitara la lectura del mismo, más que todo entender lo que quiere decir el código, por lo tanto, esto nos indica mejora de diseño y la obtención de un código más simple sin la alteración de su funcionalidad externa."*	**Modelo de Casos de Uso**	*"Especificación de Caso de Uso"*	Cuantitativa de Razón
			Modelo de Implementación	*"Modelo de la arquitectura. Diagrama de Componentes y el Programa orientado a objetos."*	
			Modelo de Pruebas	*"Casos de Pruebas. Resultado de Pruebas"*	
			Modelo de diseño.	*"Diagrama de implementación. Diagrama de Secuencia. Diagrama de Clases."*	

La Mantenibilidad	*"Medida de la facilidad con la que una aplicación de software es modificada, o corregida cuando se detectan fallos."* (ISO/IEC, 2003)	*"La Mantenibilidad indica la medición de que tan rápido y fácil puede ser restaurado un sistema cuando se presenta alguna deficiencia en el software.* *Sin embargo, para llevar a cabo la mantenibilidad se debe tener en cuenta: diseño del equipo e instalación, disponibilidad del personal con niveles de habilidad necesarios, procedimientos de mantenimiento y prueba de equipo adecuado y finalmente ambiente físico."*	**Comprensibilidad**	***"DIT** (Profundidad en árbol de herencia)."). Es una medida de la complejidad de una clase y su potencial de reuso"*	Cuantitativa de Razón
			Complejidad	***"CBO** (Acoplamiento entre Objetos). Es un indicador del esfuerzo en el mantenimiento y testeo"*	
				***"DIT** (Profundidad en árbol de herencia). Es una medida de la complejidad de una clase y su potencial de reuso"*	
				***"WMC** (Métodos ponderados por clase). Es una medida de complejidad de una clase"*	
			Conformidad	*"Cantidad de estándares satisfechas relativas con la mantenibilidad"*	

2.3 Población y Muestra

Debido a que mi unidad de estudio es un Software solo aplicare mi investigación a un solo sistema por lo tanto con cuento con una población ni una muestra.

2.4 Técnicas e instrumentos de recolección de datos, validez y confiabilidad

Este tipo de investigación no cuenta con instrumento de recolección de datos como la encuesta, entrevista entre otros, ya que para poder realizar los cálculos respectivos al producto software existen las métricas de software para indicar la calidad que cuenta el software.

Instrumentos:

- Herramientas de medición automática de métricas de software.

2.5 Métodos de análisis de datos

2.5.1 Métrica de Comprensibilidad.

NOMBRE:	DIT – Profundidad en árbol de herencia
PROPÓSITO:	Como medida de la complejidad de una clase, complejidad de diseño y el potencial de reúso.
MÉTODO DE APLICACIÓN:	Es la cuenta directa de los niveles de jerarquía de la herencia desde la clase raíz.

2.5.2 Métrica de Complejidad.

NOMBRE:	CBO – Acoplamiento entre objetos
PROPÓSITO:	Se propone como un indicador del esfuerzo necesario para el mantenimiento y testeo.
MÉTODO DE APLICACIÓN:	Cantidad de clases las cuales otra clase está ligada.
INTERPRETACIÓN:	Mientras más acoplamiento se da en una clase. Más difícil será reutilizar la clase. Las clases con demasiado acoplamiento dificultan la comprensibilidad y hacen más tedioso el mantenimiento.
FUENTE DE MEDICIÓN:	Métodos de las clases del sistema

NOMBRE:	WMC – Métodos ponderados por clase
PROPÓSITO:	Como una medida de complejidad de una clase.
MÉTODO DE APLICACIÓN:	Si todos los métodos son considerados igualmente complejos WMS es simplemente el número de métodos definidos en una clase.
MEDICIÓN, FÓRMULA:	Ilustración 1. Medición y formula de WMC $WMC = \sum_{i=1}^{n} c_i$, donde una clase C_i tiene los métodos $M_1, \ldots M_n$, con su complejidad respectiva, $c_1, \ldots, c_n$.
INTERPRETACIÓN:	Se debería considerar simplemente como una medida del tamaño de una clase.
FUENTE DE MEDICIÓN:	Métodos de las clases del sistema

2.5.3 Métrica de Conformidad.

NOMBRE:	Conformidad de la Mantenibilidad
PROPÓSITO:	El grado de cumplimiento es la capacidad de mantenimiento del producto a los reglamentos, normas y convenciones.
MÉTODO DE APLICACIÓN:	Contar el número de elementos que requieren el cumplimiento que se han cumplido y comparar con el número de artículos que requieren el cumplimiento de la especificación.
MEDICIÓN, FÓRMULA:	X = A / B A = Número de objetos de cumplimiento capacidad de mantenimiento especificados que no han sido implementadas durante la prueba. B = Número total de elementos de cumplimiento de mantenimiento especificados.
TIPO DE ESCALA:	Absoluta
INTERPRETACIÓN:	0 <= X <= 1 Cuanto más cerca de 1, es lo mejor.
ISOO / IEC 12207 SLCP:	5.3 Las pruebas de calificación 6.5 Validación
FUENTE DE MEDICIÓN:	Especificación de conformidad y las normas relacionadas, convenciones y reglamentos. Diseño del código fuente y el informe de revisión.

Cronograma de Ejecución

ESCALA DE TIE

lun 21/03/16 — sáb 2/07/16 — Hoy

abril | mayo | junio | julio | agosto | septiembre | octubre | noviembre | diciembre

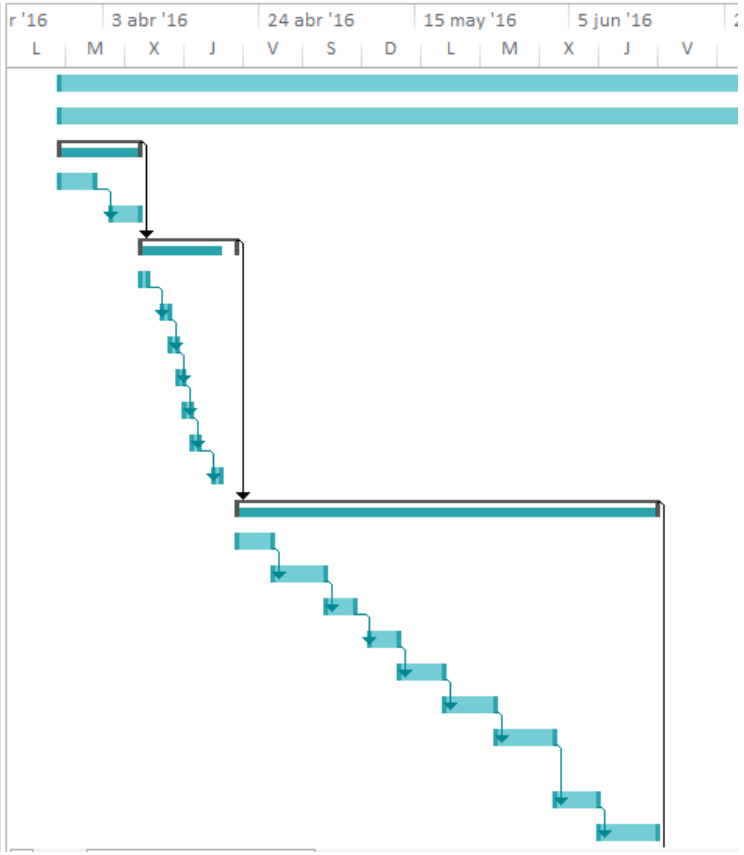
Comienzo lun 28/03/16

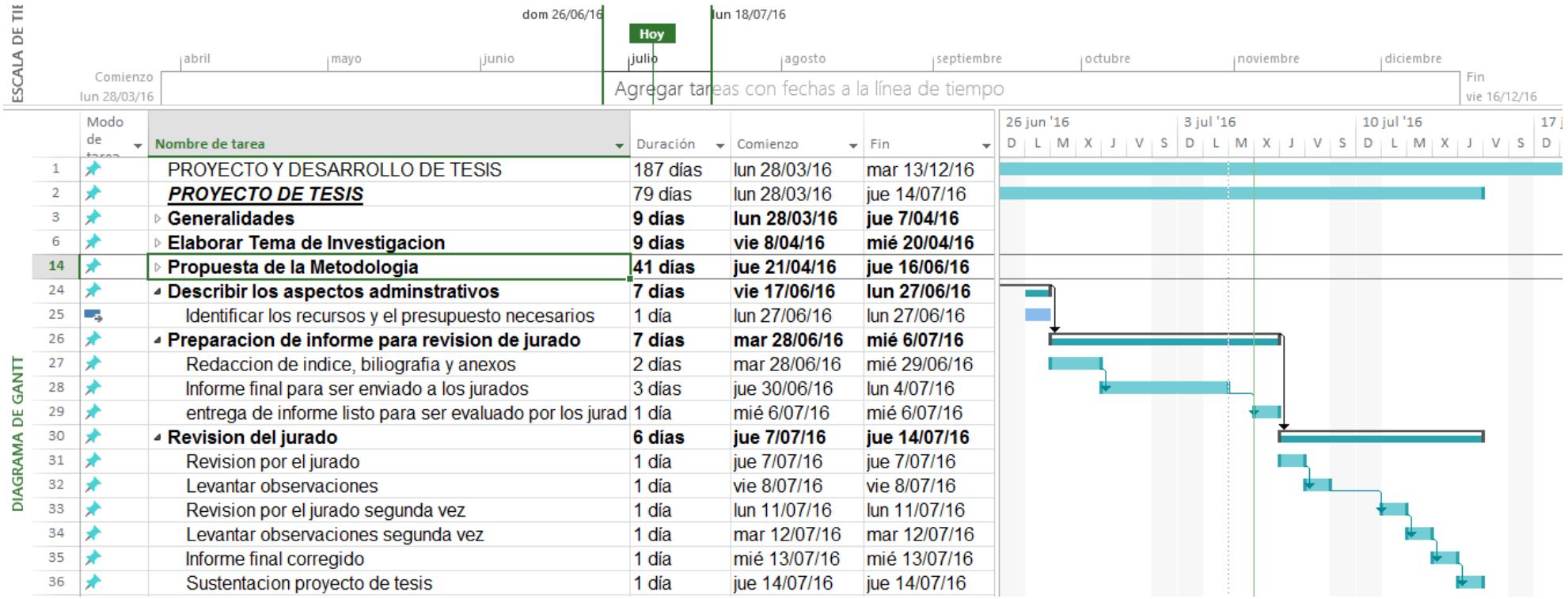
Agregar tareas con fechas a la línea de tiempo

Fin vie 16/12/16

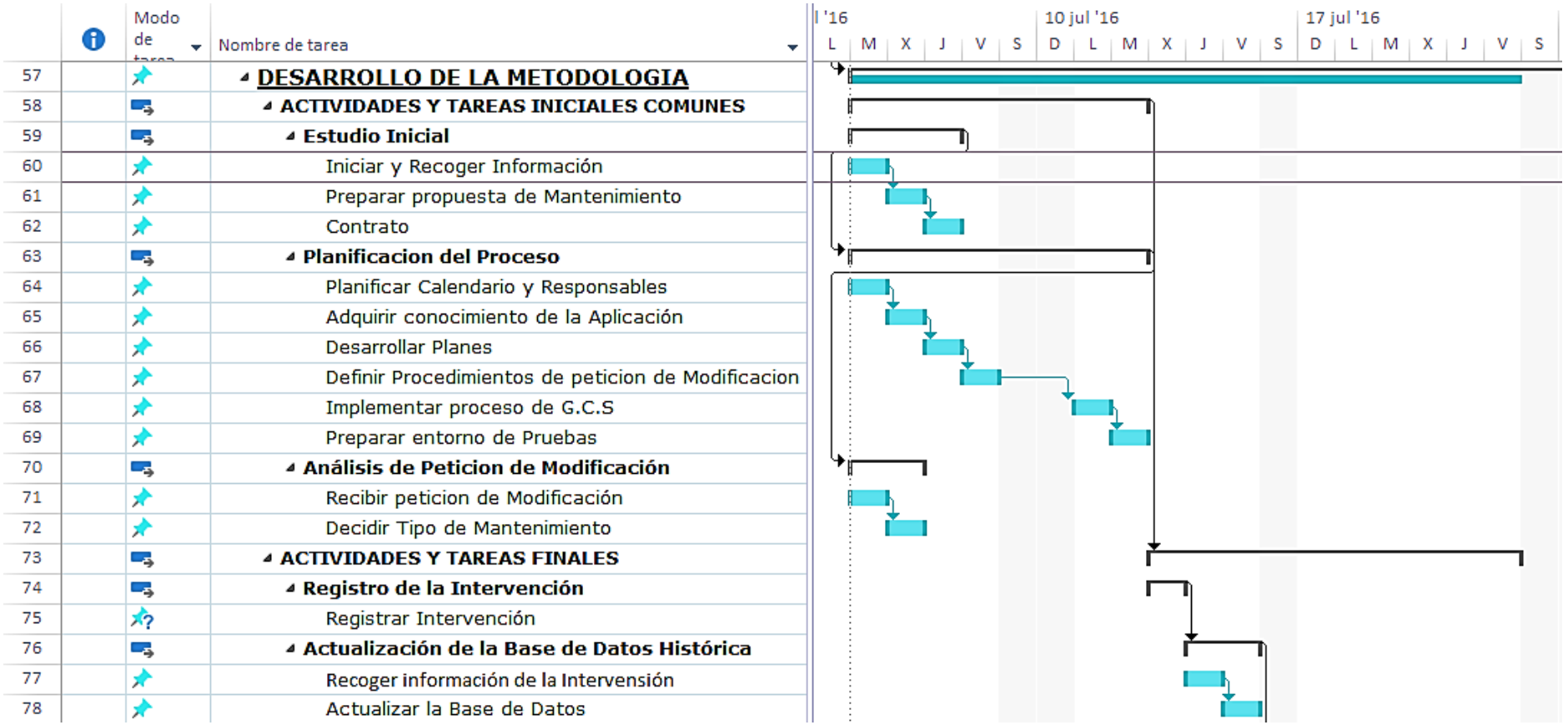
DIAGRAMA DE GANTT

	Modo de tarea	Nombre de tarea	Duración	Comienzo	Fin
1		PROYECTO Y DESARROLLO DE TESIS	187 días	lun 28/03/16	mar 13/12/16
2		***PROYECTO DE TESIS***	79 días	lun 28/03/16	jue 14/07/16
3		**Generalidades**	**9 días**	**lun 28/03/16**	**jue 7/04/16**
4		Elaborar el tema de la investigacion	5 días	lun 28/03/16	vie 1/04/16
5		Identificar Tipo de Investigacion	4 días	lun 4/04/16	jue 7/04/16
6		**Elaborar Tema de Investigacion**	**9 días**	**vie 8/04/16**	**mié 20/04/16**
7		Plantear el programa de investigacion	1 día?	vie 8/04/16	vie 8/04/16
8		Formular el problema de la Investigacion	1 día?	lun 11/04/16	lun 11/04/16
9		Establecer los objetivos	1 día?	mar 12/04/16	mar 12/04/16
10		Elaborar los Antecedentes	1 día?	mié 13/04/16	mié 13/04/16
11		Elaborar la Justificacion	1 día?	jue 14/04/16	jue 14/04/16
12		Elaborar el Marco Teorico	1 día?	vie 15/04/16	vie 15/04/16
13		Elaborar Marco Conceptual	1 día?	lun 18/04/16	lun 18/04/16
14		**Propuesta de la Metodologia**	**41 días**	**jue 21/04/16**	**jue 16/06/16**
15		Describir el tipo de estudio	3 días	jue 21/04/16	lun 25/04/16
16		Describir el diseño de estudio	5 días	mar 26/04/16	lun 2/05/16
17		Plantear la hipotesis	4 días	mar 3/05/16	vie 6/05/16
18		Identificar las variables de la investigacion	4 días	lun 9/05/16	jue 12/05/16
19		Determinar la poblacion, muestra y muestreo	4 días	vie 13/05/16	mié 18/05/16
20		Describir los criterios de selección	5 días	jue 19/05/16	mié 25/05/16
21		Determinar las tecnicas e instrumentos de recoleccion de datos	6 días	jue 26/05/16	jue 2/06/16
22		Realizar la validacion y confiabilidad del intrumento	4 días	vie 3/06/16	mié 8/06/16
23		Proponer los metodos de analisis de datos	6 días	jue 9/06/16	jue 16/06/16

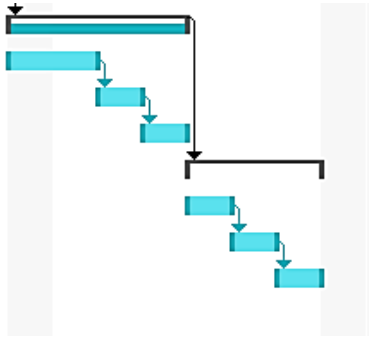




	Modo de tarea	Nombre de tarea	Duración	Comienzo	Fin
1		PROYECTO Y DESARROLLO DE TESIS	187 días	lun 28/03/16	mar 13/12/16
2		***PROYECTO DE TESIS***	79 días	lun 28/03/16	jue 14/07/16
3		**Generalidades**	**9 días**	**lun 28/03/16**	**jue 7/04/16**
6		**Elaborar Tema de Investigacion**	**9 días**	**vie 8/04/16**	**mié 20/04/16**
14		**Propuesta de la Metodologia**	**41 días**	**jue 21/04/16**	**jue 16/06/16**
24		**Describir los aspectos adminstrativos**	**7 días**	**vie 17/06/16**	**lun 27/06/16**
25		Identificar los recursos y el presupuesto necesarios	1 día	lun 27/06/16	lun 27/06/16
26		**Preparacion de informe para revision de jurado**	**7 días**	**mar 28/06/16**	**mié 6/07/16**
27		Redaccion de indice, biliografia y anexos	2 días	mar 28/06/16	mié 29/06/16
28		Informe final para ser enviado a los jurados	3 días	jue 30/06/16	lun 4/07/16
29		entrega de informe listo para ser evaluado por los jurad	1 día	mié 6/07/16	mié 6/07/16
30		**Revision del jurado**	**6 días**	**jue 7/07/16**	**jue 14/07/16**
31		Revision por el jurado	1 día	jue 7/07/16	jue 7/07/16
32		Levantar observaciones	1 día	vie 8/07/16	vie 8/07/16
33		Revision por el jurado segunda vez	1 día	lun 11/07/16	lun 11/07/16
34		Levantar observaciones segunda vez	1 día	mar 12/07/16	mar 12/07/16
35		Informe final corregido	1 día	mié 13/07/16	mié 13/07/16
36		Sustentacion proyecto de tesis	1 día	jue 14/07/16	jue 14/07/16


Modo de tarea
Nombre de tarea
l '16
10 jul '16
17 jul '16
L M X J V S D L M X J V S D L M X J V S
57 DESARROLLO DE LA METODOLOGIA
58 ACTIVIDADES Y TAREAS INICIALES COMUNES
59 Estudio Inicial
60 Iniciar y Recoger Información
61 Preparar propuesta de Mantenimiento
62 Contrato
63 Planificacion del Proceso
64 Planificar Calendario y Responsables
65 Adquirir conocimiento de la Aplicación
66 Desarrollar Planes
67 Definir Procedimientos de peticion de Modificacion
68 Implementar proceso de G.C.S
69 Preparar entorno de Pruebas
70 Análisis de Peticion de Modificación
71 Recibir peticion de Modificación
72 Decidir Tipo de Mantenimiento
73 ACTIVIDADES Y TAREAS FINALES
74 Registro de la Intervención
75 Registrar Intervención
76 Actualización de la Base de Datos Histórica
77 Recoger información de la Intervensión
78 Actualizar la Base de Datos

79			**Retirada**
80			Desarrollar Plan de retirada
81			Notificar Futura Retirada
82			Almacenar Datos de Entorno antiguo.
83			**Fin de la Externalización**
84			Entrega del inventario y de la Documentación
85			Traspaso de Experiencia y Formación
86			Cesión definitiva del Servicio



CAPITULO III

RESULTADOS

III. RESULTADOS

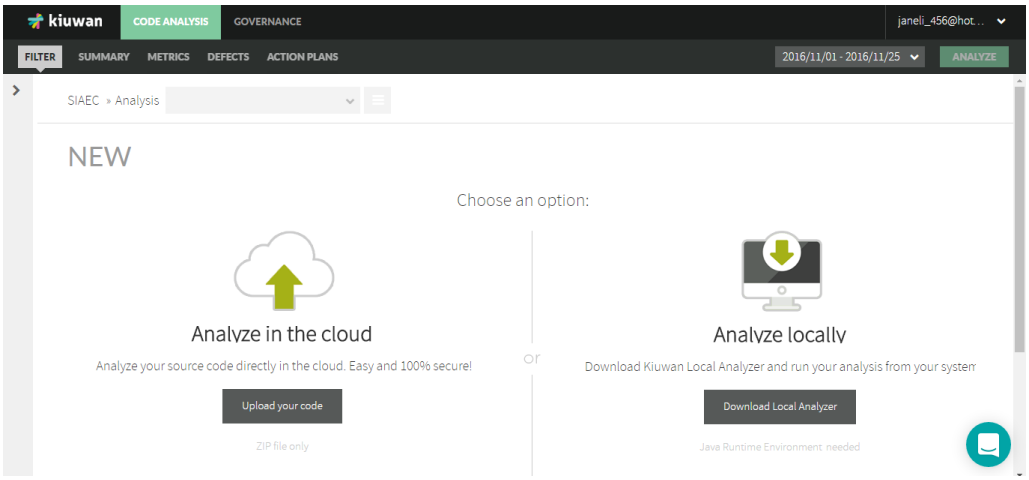
Proceso de Mejora de Pre-Test a Post-Test

FASE DE EVALUACIÓN

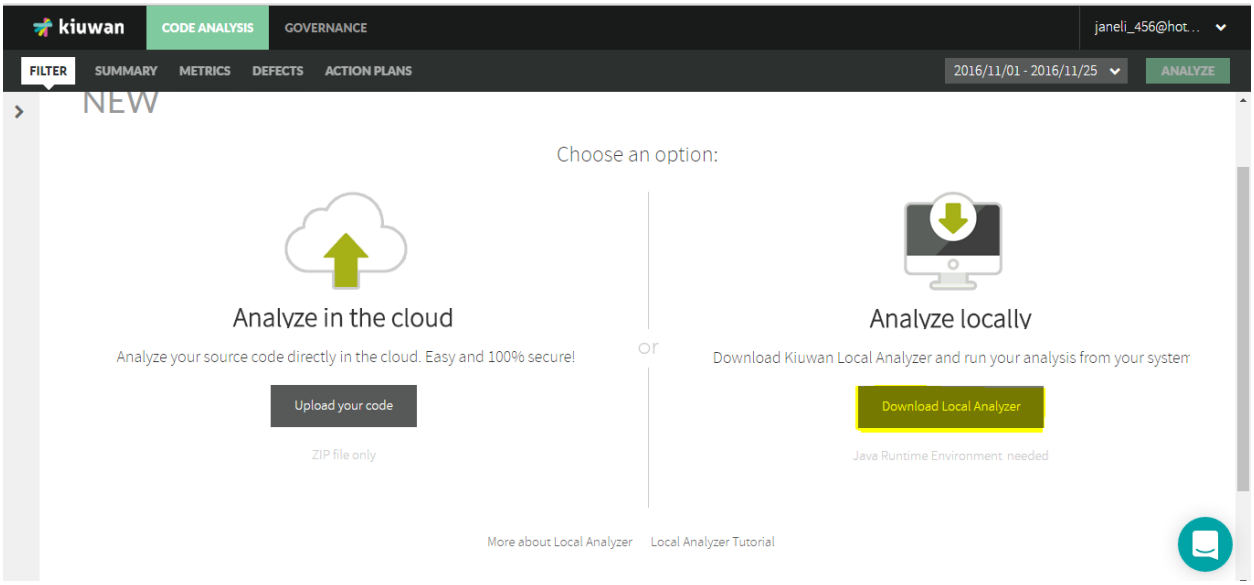
PASOS PARA INICIAR ANÁLISIS

a) Ir a www.kiuwan.com ingresar Credenciales o registrarse para poder acceder a Kiuwan.

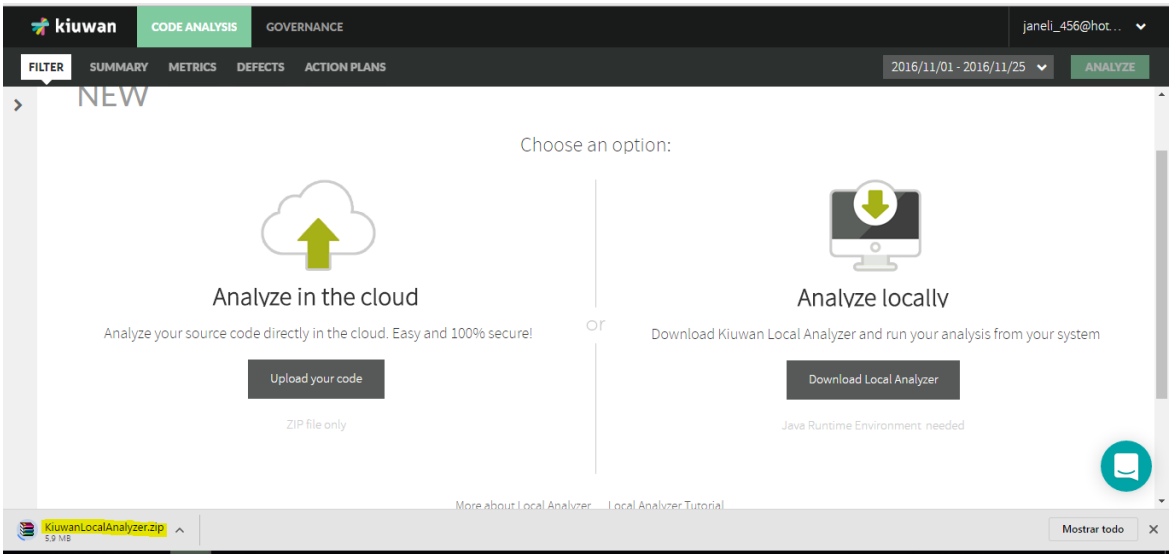
b) Una vez ingresadas las credenciales podrá colocar el nombre del proyecto y luego tendrá dos opciones para iniciar el análisis de código fuente del proyecto que se desee evaluar, ya sea en la NUBE o LOCALMENTE.

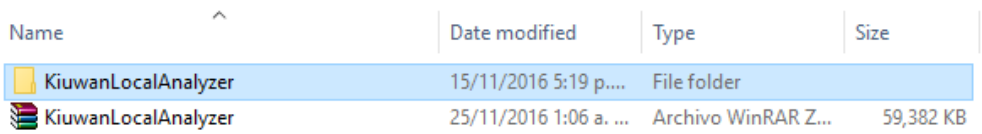


c) En este caso decidí elegir LOCALMENTE, para localmente se requiere descargar un archivo para poder subir el proyecto.

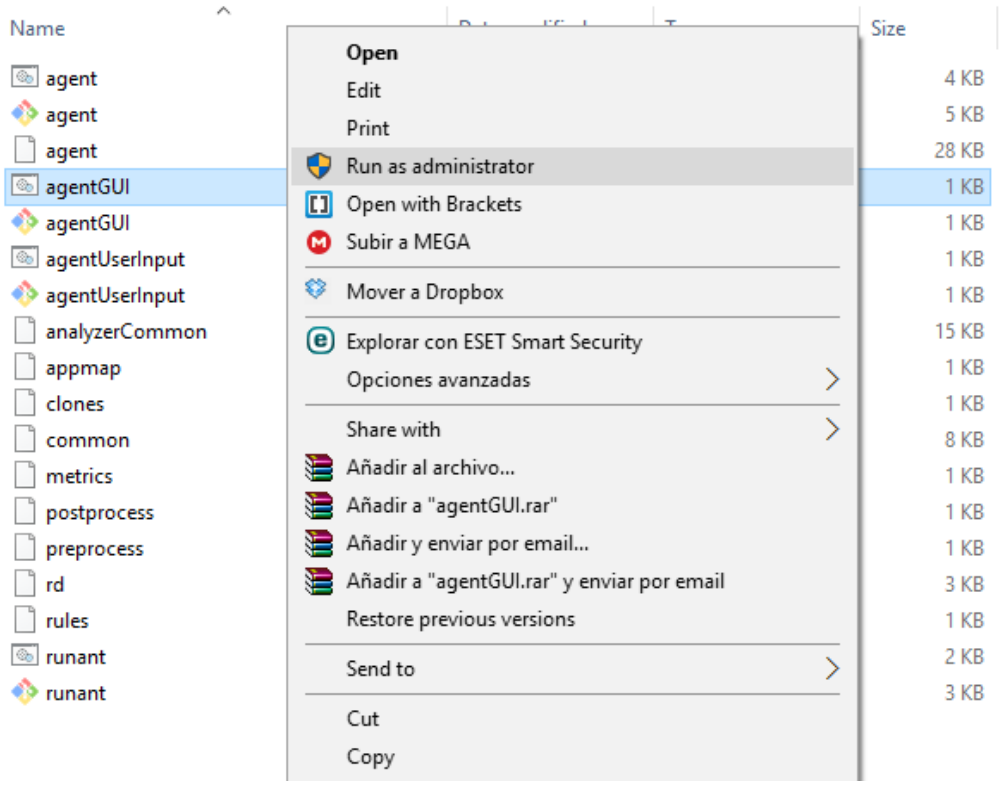


d) Se descargará un archivo. Zip, una vez descargada descomprimir el archivo.

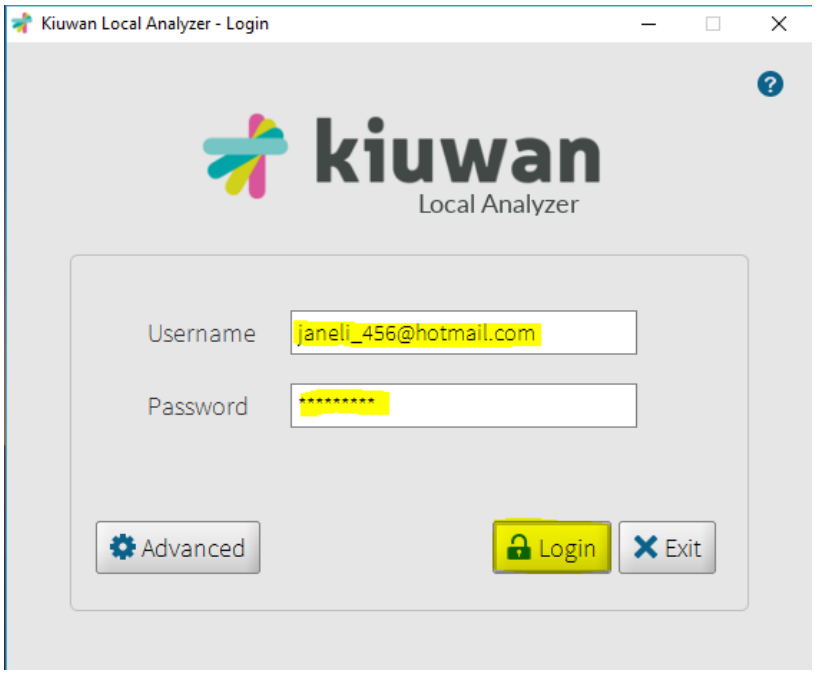




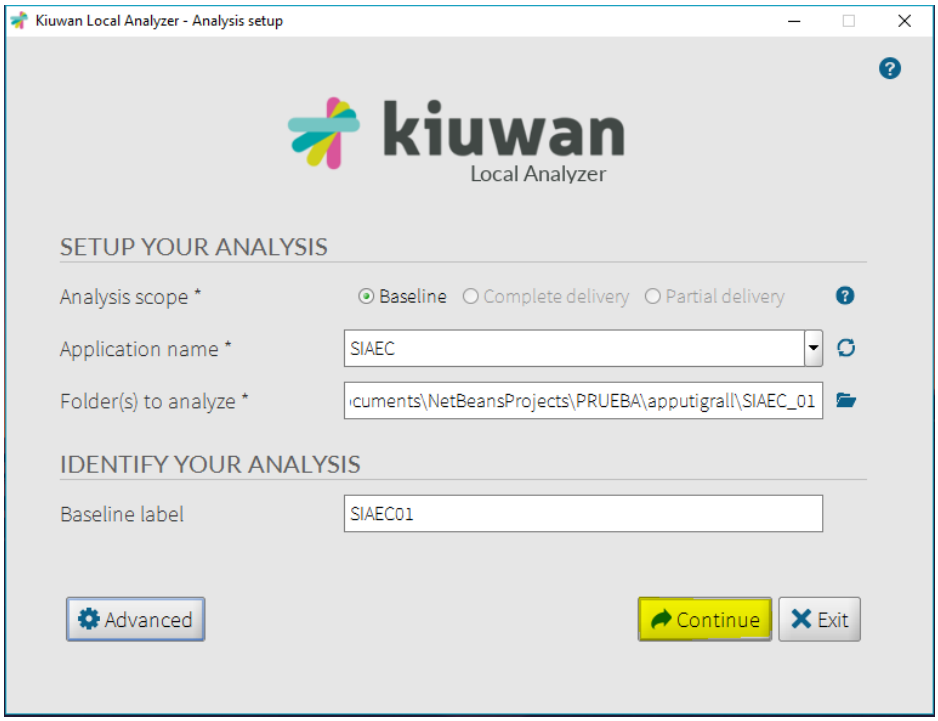
e) Ingresar a la Carpeta descomprimida e ir a la carpeta **/bin** y ejecutar como administrador **agenteGUI.cmd**



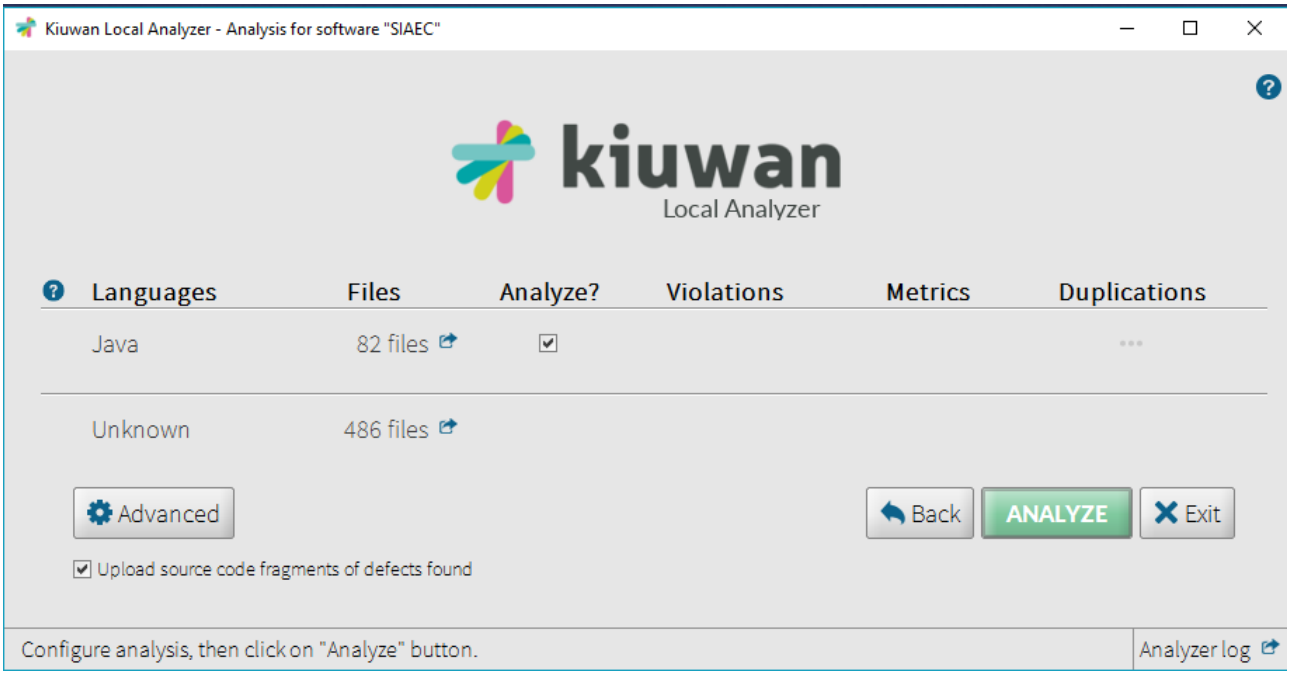
f) Ingresar las mismas credenciales con las que se registró en la Web www.kiuwan.com



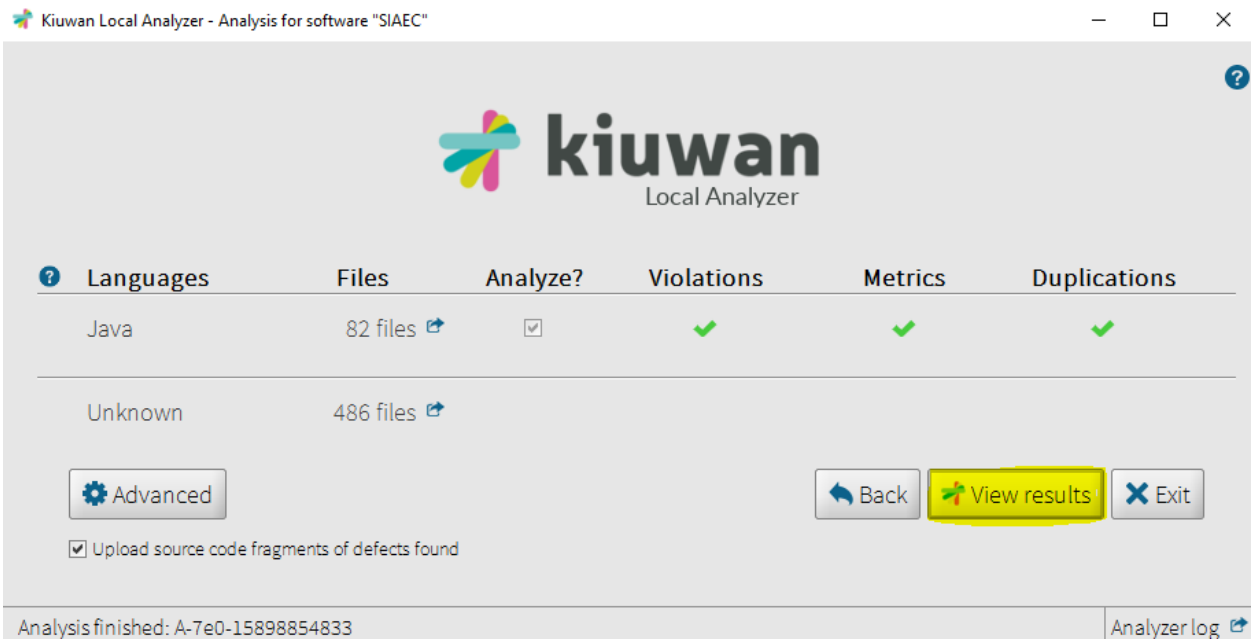
g) Se mostrará una pantalla donde ingresará: **Nombre de la Aplicación** [SIAEC]; **Ruta donde se encuentra el proyecto a analizar**; **Nombre Base** [SIAEC01] y finalmente dar **Continuar**.



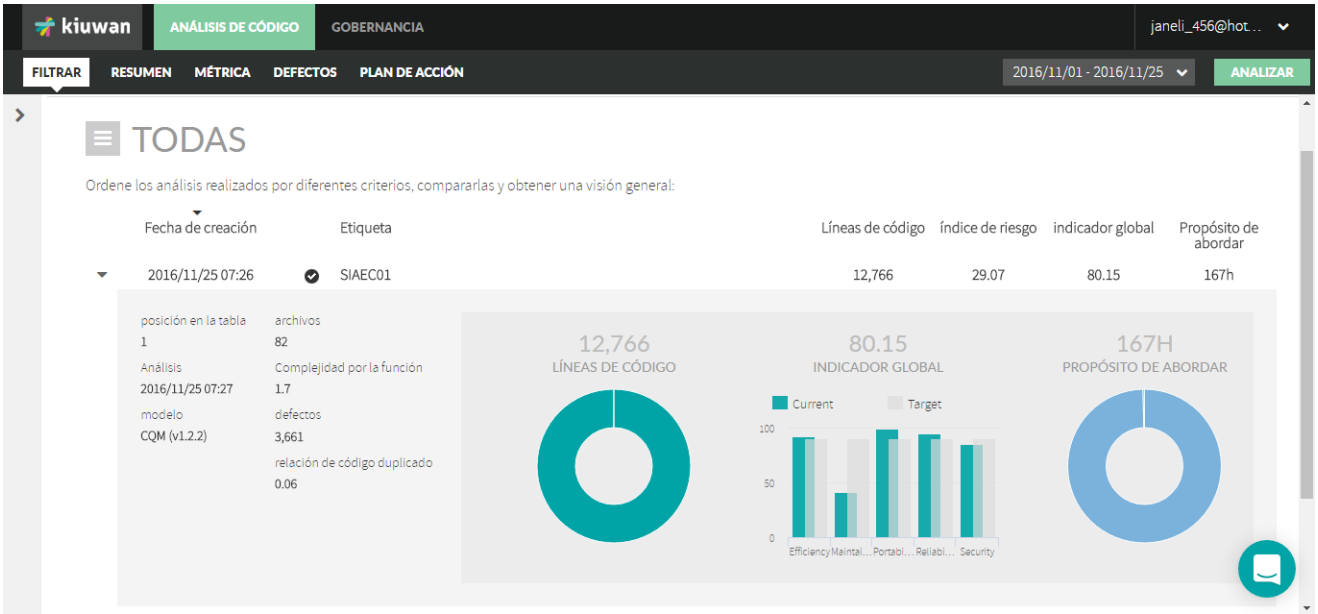
h) Ya obtenido el proyecto, se puede iniciar el **Análisis del Proyecto.**



i) Luego de analizar el proyecto de forma local, se dará una opción donde nos permita visualizar el análisis detallado del proyecto. **Ver Resultados**

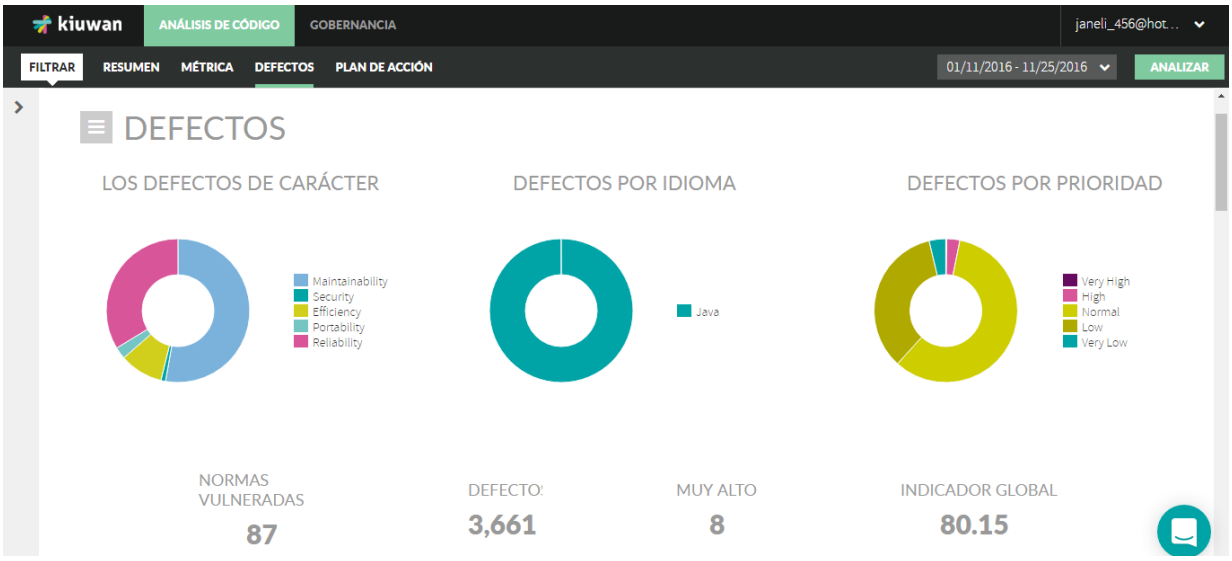


j) La opción nos llevará directamente a la Pagina web de Kiuwan, se deberá ingresar nuevamente las credenciales, y finalmente se observará el análisis del sistema. [**Sistema Prototipo SIAEC**]

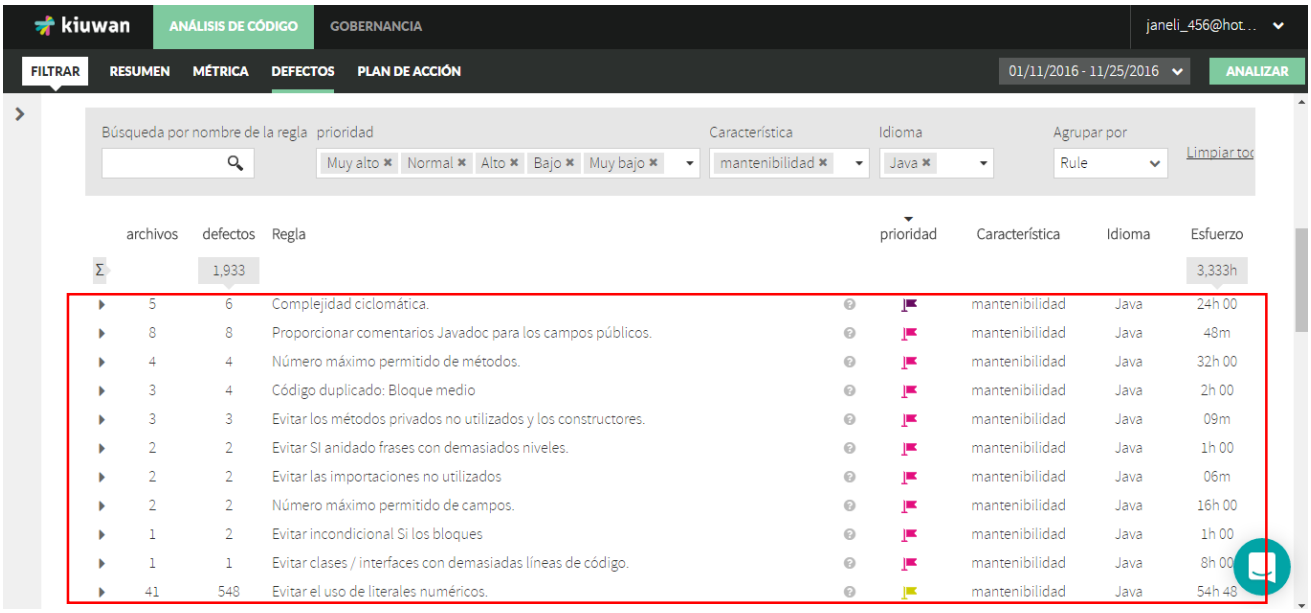


Defectos Detectados
Los defectos detallados que nos muestra Kiuwan son enfocados en:

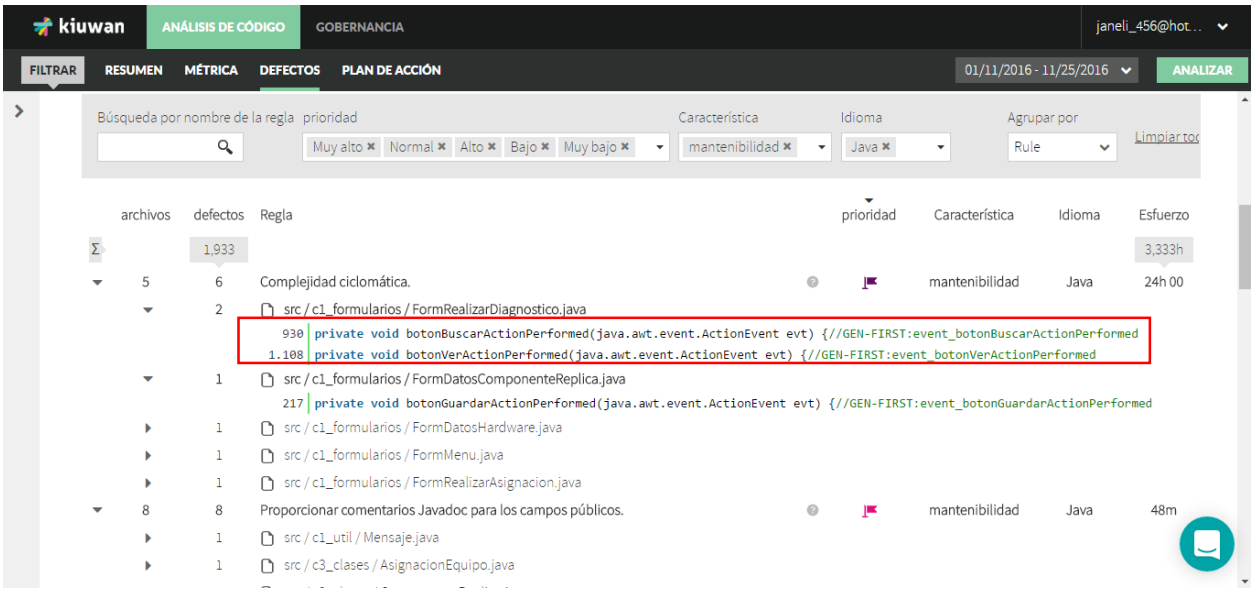
- ✓ **Mantenibilidad**
- ✓ **Seguridad**
- ✓ **Eficiencia**
- ✓ **Portabilidad**
- ✓ **Fiabilidad**



Como la mantenibilidad es el motivo de estudio, Kiuwan también nos permite realizar un filtro para solo visualizar los defectos que generan que la mantenibilidad este con mayores defectos.

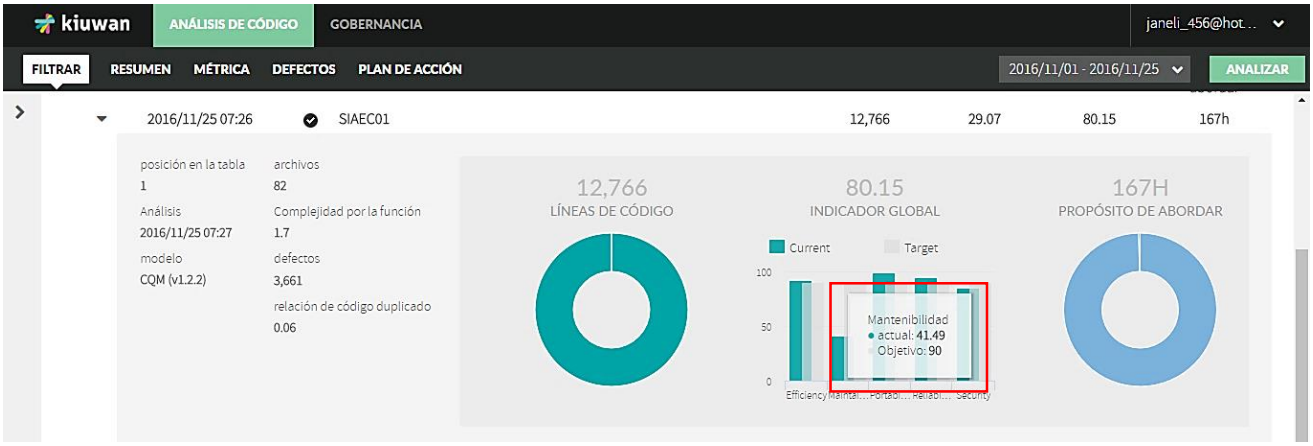


Dentro de cada uno de los defectos nos indican en que parte del código esta los defectos. (**Carpeta/Numero de Línea**)



1.1 RESULTADOS DE EVALUACION DE SISTEMA PRE-TEST

Los resultados obtenidos en forma de resumen son los siguientes.



Mantenibilidad:

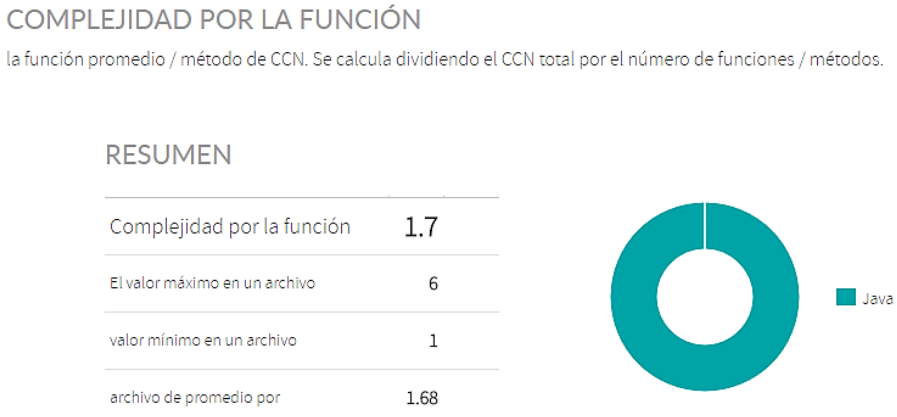
- ✓ Objetivo Esperado: **90**
- ✓ Objetivo conseguido: **41.49**

Métricas de Aplicación:

- Relación Código Duplicado: **0.06**
- Complejidad por la función: **1.7**

✓ <u>COMPLEJIDAD</u>

- Complejidad por funciones



- Complejidad

COMPLEJIDAD

Es un normalizado (entre 0 y 100) métrica basada en la complejidad ciclomática por función, la duplicación de código y capacidad de mantenimiento de índice.

RESUMEN

Complejidad	43.35

No data to display

✓ DUPLICACIÓN DE CODIGO

- Relación de código Duplicado

Ratio de código duplicado (medido en fichas). El término "token" se refiere a cada uno de los elementos atómicos identificados por un analizador sintáctico. Por ejemplo, un operador, un identificador, un número, etc.

La duplicación de código, cuando es excesiva, dificulta el mantenimiento, ya que cualquier cambio de código lógico en un clon debe realizarse en todas las instancias de clonación. Para mejorar la capacidad de mantenimiento, el código podría Refactorizar para colocar todas las instancias de clon en un solo lugar (por ejemplo, una nueva función, párrafo o método) y, a continuación, llamar al código refactorizado.

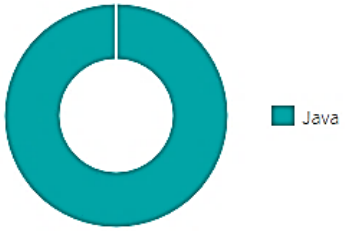
RESUMEN

relación de código duplicado	0.06
El valor máximo en un archivo	0.3
valor mínimo en un archivo	0.04
archivo de promedio por	0.18

- Duplicaciones

  Número total de bloques de código duplicado.

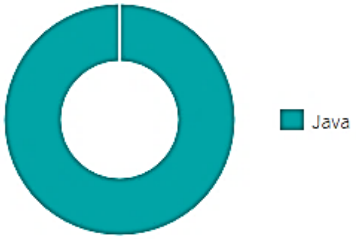
RESUMEN	
duplicaciones	13



- Radio de archivos con duplicaciones

  Radio de archivos afectados por código duplicado.

RESUMEN	
Ratio de archivos con duplicaciones	0.15



✓ <u>CODE SMELLS</u>

- Líneas de Comentarios

  Número de líneas con comentarios.
  Se incluye líneas simples de comentarios y comentarios de bloque, en su mayoría si las líneas de comentario son excesivas produce que el código tenga Mal Olor “Code Smell”

RESUMEN	
Las líneas de comentarios	1,423
El valor máximo en un archivo	87
valor mínimo en un archivo	5
archivo de promedio por	17.35

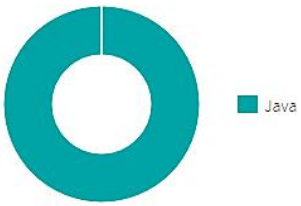
✓ RIESGO

RIESGO

indicador de riesgo, representa lo que es el riesgo de que te lleva si no hace nada para reparar sus problemas. Se calcula a partir de su indicador de GCC y el esfuerzo que tiene que pasar para alcanzar el objetivo que se configure a su aplicación. Si el indicador es mejor que su objetivo, su riesgo será 0.

RESUMEN

Riesgo	28.85



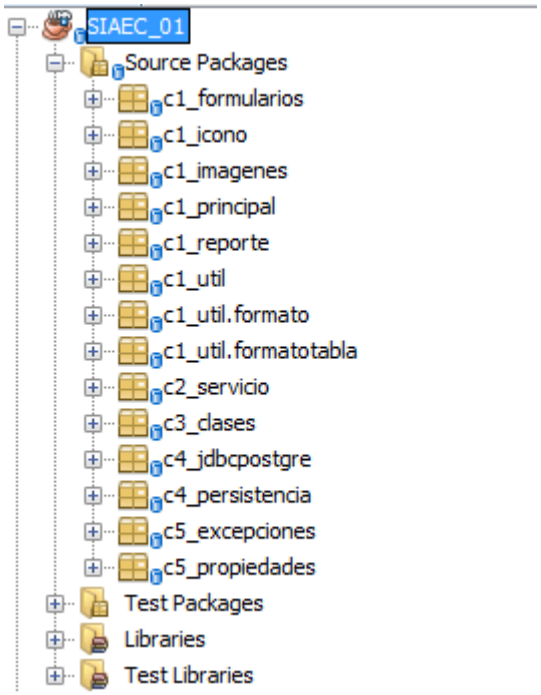
Aplicación de técnicas de refactorización y comparación Pre-Test Y Post-Test

REFACTORIZACIÓN DE ARQUITECTURA DE SOFTWARE.

El sistema prototipo SIAEC está desarrollada bajo la arquitectura N-Capas.

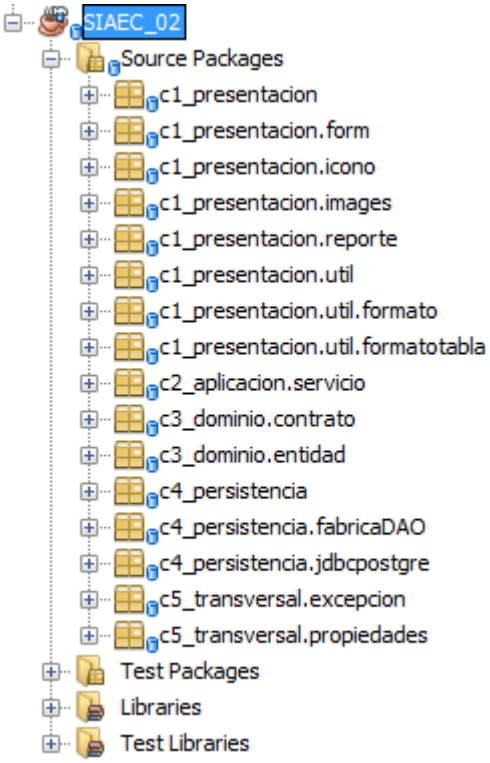
SIAEC-01

Arquitectura de software original.



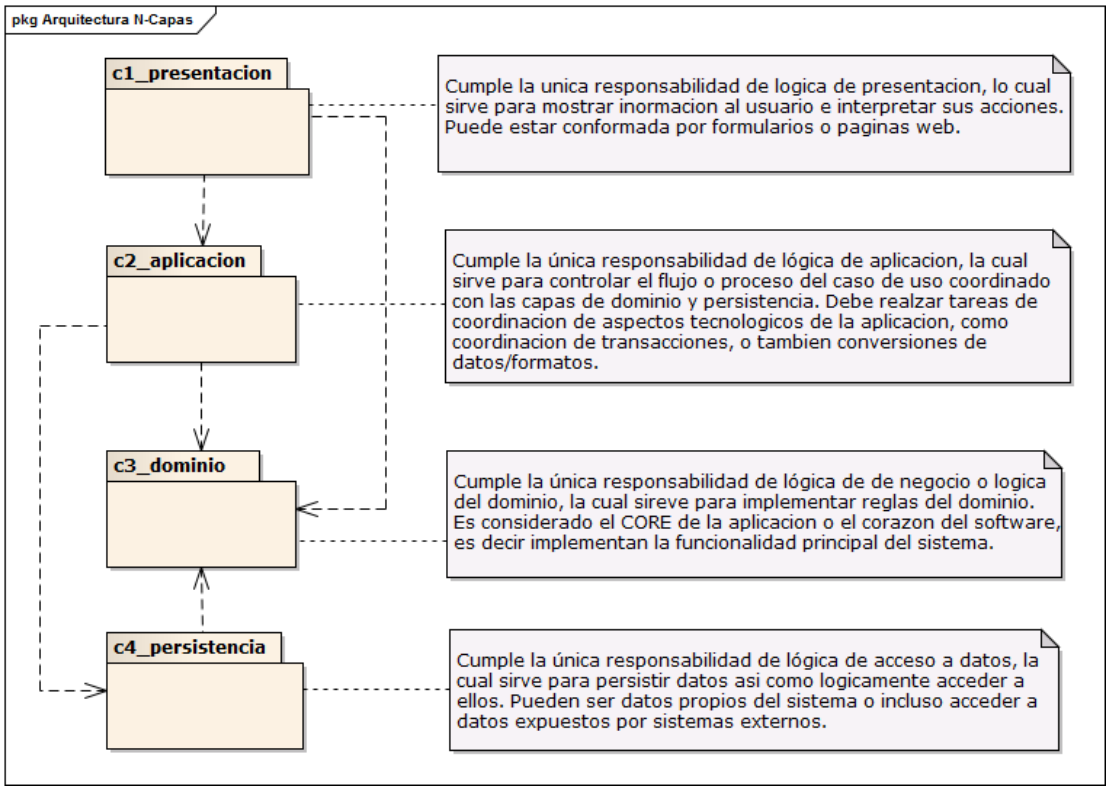
SIAEC-02

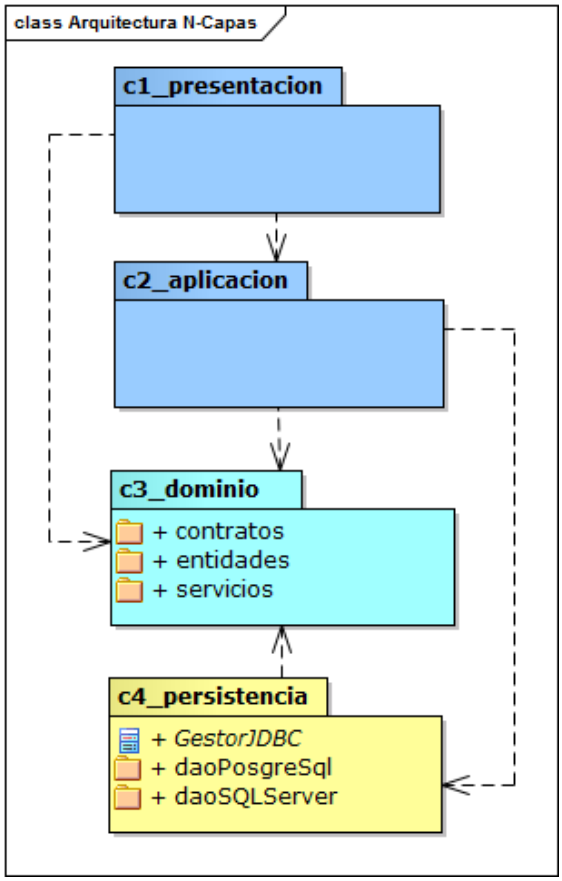
Arquitectura Refactorizada.



La arquitectura N-Capas tiene una estructura ya definida y debe ser respetada ya que esta permite localizar los cambios, permite también la separación de responsabilidades, reutilización de componentes, además los diferentes componentes pueden ser desplegados de una forma independiente y cada capa puede contener sus propias pruebas unitarias.

Por consiguiente, mostraremos como se estructura la Arquitectura N.-Capas:


pkg Arquitectura N-Capas
c1_presentacion
Cumple la unica responsabilidad de logica de presentacion, lo cual sirve para mostrar inormacion al usuario e interpretar sus acciones. Puede estar conformada por formularios o paginas web.
c2_aplicacion
Cumple la única responsabilidad de lógica de aplicacion, la cual sirve para controlar el flujo o proceso del caso de uso coordinado con las capas de dominio y persistencia. Debe realzar tareas de coordinacion de aspectos tecnologicos de la aplicacion, como coordinacion de transacciones, o tambien conversiones de datos/formatos.
c3_dominio
Cumple la única responsabilidad de lógica de de negocio o logica del dominio, la cual sireve para implementar reglas del dominio. Es considerado el CORE de la aplicacion o el corazon del software, es decir implementan la funcionalidad principal del sistema.
c4_persistencia
Cumple la única responsabilidad de lógica de acceso a datos, la cual sirve para persistir datos asi como logicamente acceder a ellos. Pueden ser datos propios del sistema o incluso acceder a datos expuestos por sistemas externos.

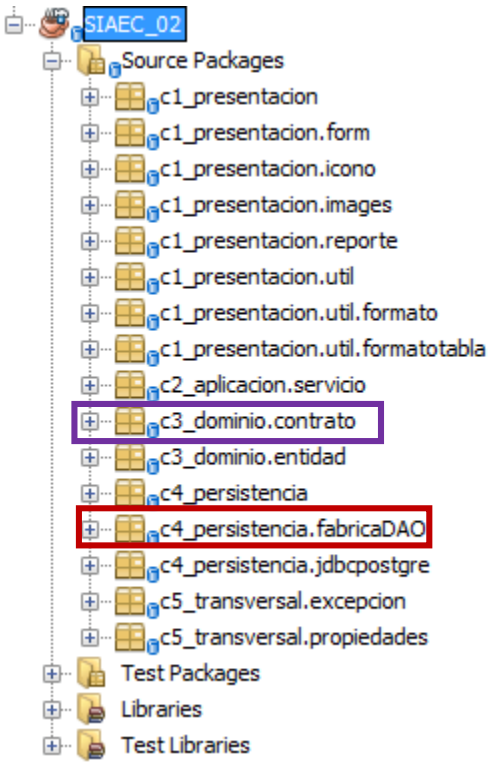

class Arquitectura N-Capas
c1_presentacion
c2_aplicacion
c3_dominio
+ contratos
+ entidades
+ servicios
c4_persistencia
+ GestorJDBC
+ daoPosgreSql
+ daoSQLServer

Para la refactorización de la Arquitectura se hizo lo siguiente:

- ✓ **Renombrar Nombre de Paquetes**
- ✓ **Añadir Paquetes:**

a. c1_dominio. contrato: Aquí se añadirán las Interfaces.

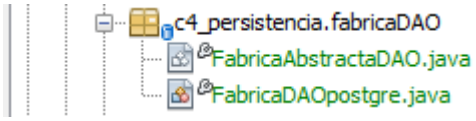
El Contrato (Interfaz) está dentro de la Capa N° 3 que es el Dominio, esta capa es responsable de representar conceptos de negocio, información sobre la situación del proceso de negocio e implementación de las reglas del dominio en otras palabras es el **corazón del negocio**.



b. En la capa de **dominio.contrato** se agregarán las Interfaces que se generarán por cada clase.

c. c4_persistencia.fabricaDAO: Aquí se añadirá el Patrón **Abstract Factory**, este patrón permite delegar en una clase la responsabilidad de crear objetos de otras clases.

Se realizaron los siguientes pasos para añadir Abstract Factory:
Se crearon dos recursos: **FabricaAbstractDAO.java** y **FabricaDAOPostgre.java**



FabricaAbstractDAO.java

FabricaAbstractaDAO.java

```
public abstract class FabricaAbstractaDAO {

    public static FabricaAbstractaDAO getInstancia(){
        String claseFabricaDAO;
        FabricaAbstractaDAO FabricaDAO;
        try {
            LectorPropiedades parametro = new LectorPropiedades
                ("c5_transversal/propiedades/Parametros.properties");
            claseFabricaDAO = parametro.getValorParametro("claseFabricaDAO");
            FabricaDAO = (FabricaAbstractaDAO)Class.forName(claseFabricaDAO).newInstance();
            return FabricaDAO;
        } catch (ClassNotFoundException |
                 InstantiationException |
                 IllegalAccessException e) {
            return null;
        }
    }

    public abstract GestorJDBC crearGestorJDBC();
    public abstract IAreaDAO crearAreaDAO (GestorJDBC gestorJDBC);
    public abstract IComponenteDAO crearComponenteDAO(GestorJDBC gestorJDBC);
    public abstract IEquipoDAO crearEquipoDAO(GestorJDBC gestorJDBC);
    public abstract IHardwareDAO crearHardwareDAO(GestorJDBC gestorJDBC);
    public abstract IMarcaDAO crearMarcaDAO(GestorJDBC gestorJDBC);
    public abstract IComponenteReplicaDAO crearComponenteReplicaDAO(GestorJDBC gestorJDBC);
    public abstract ISistemaOperativoDAO crearSistemaOperativoDAO(GestorJDBC gestorJDBC);
    public abstract IUsuarioDAO crearUsuarioDAO(GestorJDBC gestorJDBC);
    public abstract IAsignacionEquipoDAO crearAsignacionEquipoDAO(GestorJDBC gestorJDBC);
    public abstract IEmpleadoDAO crearEmpleadoDAO(GestorJDBC gestorJDBC);
    public abstract ISedeDAO crearSedeDAO(GestorJDBC gestorJDBC);
    public abstract IDiagnosticoDAO crearDiagnosticoDAO(GestorJDBC gestorJDBC);
```

FabricaDAOPostgre.java

Se heredan los datos y son obtenido con extends **FabricaAbstractDAO**

FabricaDAOpostgre.java

```
33  public class FabricaDAOpostgre extends FabricaAbstractaDAO {
34
35      @Override
36      public GestorJDBC crearGestorJDBC() {
37          return new ConexionJDBCPostgre();
38      }
39
40      @Override
41      public IAreaDAO crearAreaDAO(GestorJDBC gestorJDBC) {
42          return new AreaDAOPostgre(gestorJDBC);
43      }
44
45      @Override
46      public IComponenteDAO crearComponenteDAO(GestorJDBC gestorJDBC) {
47          return new ComponenteDAOPostgre(gestorJDBC);
48      }
49
50      @Override
51      public IEquipoDAO crearEquipoDAO(GestorJDBC gestorJDBC) {
52          return new EquipoDAOPostgre(gestorJDBC);
53      }
54
55      @Override
56      public IHardwareDAO crearHardwareDAO(GestorJDBC gestorJDBC) {
57          return new HardwareDAOPostgre(gestorJDBC);
58      }
59
60      @Override
61      public IMarcaDAO crearMarcaDAO(GestorJDBC gestorJDBC) {
62          return new MarcaDAOPostgre(gestorJDBC);
63      }
```

Las modificaciones que se tienen que realizar para que el software se adecue a este recurso nuevo se debe ir a **c2_aplicacion.servicio** y realizar los siguientes cambios, escogeremos un solo ejemplo para demostrar cómo deben ser los cambios.

GestionarAreaServicio.java

```
21  public class GestionarAreaServicio{
22      GestorJDBC gestorJDBC;
23      AreaDAOPostgre areaDAOPostgre;
24
25      public GestionarAreaServicio() {
26          gestorJDBC = new ConexionJDBCPostgre();
27          areaDAOPostgre = new AreaDAOPostgre(gestorJDBC);
28      }
29
30      public void crear(Area area) throws Exception {
31          gestorJDBC.abrirConexion();
32          try {
33              areaDAOPostgre.crear(area);
34          } catch (Exception e) {
35              gestorJDBC.cerrarConexion();
36              throw e;
37          }
38          gestorJDBC.cerrarConexion();
39      }
40
41      public void modificar(Area area) throws Exception {
42          gestorJDBC.abrirConexion();
43          try {
44              areaDAOPostgre.modificar(area);
45          } catch (Exception e) {
46              gestorJDBC.cerrarConexion();
47              throw e;
48          }
49          gestorJDBC.cerrarConexion();
50      }
```

SIAEC-02

GestionarAreaServicio.java

```java
import c3_dominio.contrato.IAreaDAO;
import c3_dominio.entidad.Area;
import c4_persistencia.GestorJDBC;
import c4_persistencia.fabricaDAO.FabricaAbstractaDAO;
import java.util.ArrayList;
import java.util.List;

/**
 *
 * @author
 * <AdvanceSoft - Mendoza Torres Valentin - advancesoft.trujillo@gmail.com>
 */
public class GestionarAreaServicio{
    GestorJDBC gestorJDBC;
    IAreaDAO areaDAO;

    public GestionarAreaServicio() {
        FabricaAbstractaDAO fabricaAbstractaDAO = FabricaAbstractaDAO.getInstancia()
        gestorJDBC = fabricaAbstractaDAO.crearGestorJDBC();
        areaDAO = fabricaAbstractaDAO.crearAreaDAO(gestorJDBC);
    }

    public void crear(Area area) throws Exception {
        gestorJDBC.abrirConexion();
        try {
            areaDAO.crear(area);
        } catch (Exception e) {
            gestorJDBC.cerrarConexion();
            throw e;
        }
        gestorJDBC.cerrarConexion();
    }
```

Los cambios deben aplicarse a todos los recursos que estén **c2_aplicacion. servicio.**

EXTRAER MÉTODO.

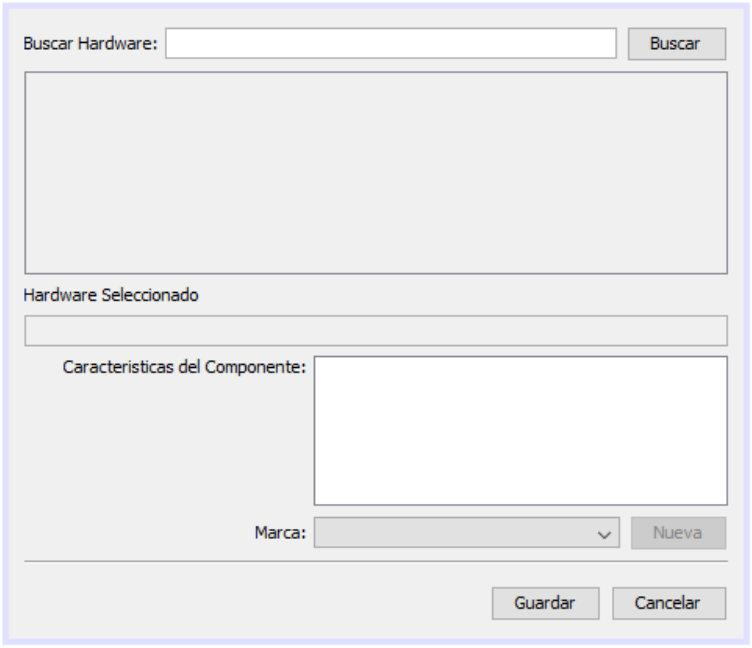
Para esta técnica de Refactorización usaremos 3 Formularios como ejemplo, para poder demostrar cómo se aplica la técnica y como mejora la legibilidad del código.

El propósito de esta técnica es para disminuir **código duplicado**.

Los Formularios son:

- ✓ **FormDatosComponentes.java**
- ✓ **FormDatosHardware.java**
- ✓ **FormGestionarComponente.java**

a) FormDatosComponentes.java (**SIAEC01**)



Interfaz Componentes

Visualización de código a mejorar:

Cada color de la forma que se usó para seleccionar el fragmento del código indica las similitudes que poseen.

FormDatosComponente.java

```
291     private void textoBuscarHardwareKeyTyped(java.awt.event.KeyEvent evt) {
292         ModeloTabla modeloTablaHardware = (ModeloTabla) tablaHardware.getModel();
293         modeloTablaHardware.eliminarTotalFilas();
294         try {
295             GestionarComponenteServicio hardwareServicio = new GestionarComponenteServicio();
296             List<Hardware> listaDeHardwares = hardwareServicio.buscarHardware(textoBuscarHardware.getText().trim().toUpperCase());
297                 for (Hardware hardware : listaDeHardwares) {
298                     Fila fila = new Fila();
299                     fila.agregarValorCelda(hardware.getCodigohardware());
300                     fila.agregarValorCelda(hardware.getNombre());
301                     modeloTablaHardware.agregarFila(fila);
302                 }
303                 modeloTablaHardware.refrescarDatos();
304         } catch (Exception e) {
305             Mensaje.mostrarErrorExcepcion(this, e.getMessage());
306         }
307         int numFila = tablaHardware.getSelectedRow();
308         if(numFila>=0){
309             Hardware hardware = obtenerObjetoDeLaTablaHardware();
310             botonNuevaMarca.setEnabled(true);
311             textoDescripcionHardware.setText(hardware.getNombre());
312             comboMarca.removeAllItems();
313             listaMarca = new ArrayList();
314             for(LineaMarcaHadware lineaMarcaHadware : hardware.getLineaMarcaHadwares()){
315                 listaMarca.add(lineaMarcaHadware.getMarca());
316             }
317             for(Marca marca : listaMarca){
318                 comboMarca.addItem(marca.getNombremarca());
319             }
320             if(componente.getCodC()>0)
321                 comboMarca.setSelectedItem(componente.getM().getNombremarca());

322         }else {
323             textoDescripcionHardware.setText("");
324             comboMarca.removeAllItems();
325             botonNuevaMarca.setEnabled(false);
326         }
327     }
```

FormDatosComponente.java

```
332     private void botonBuscarHardwareActionPerformed(java.awt.event.ActionEvent evt) {
333         ModeloTabla modeloTablaHardware = (ModeloTabla) tablaHardware.getModel();
334         modeloTablaHardware.eliminarTotalFilas();
335         try {
336             GestionarComponenteServicio hardwareServicio = new GestionarComponenteServicio();
337             List<Hardware> listaDeHardwares = hardwareServicio.buscarHardware(textoBuscarHardware.getText().trim().toUpperCase());
338                 for (Hardware hardware : listaDeHardwares) {
339                     Fila fila = new Fila();
340                     fila.agregarValorCelda(hardware.getCodigohardware());
341                     fila.agregarValorCelda(hardware.getNombre());
342                     modeloTablaHardware.agregarFila(fila);
343                 }
344                 modeloTablaHardware.refrescarDatos();
345         } catch (Exception e) {
346             Mensaje.mostrarErrorExcepcion(this, e.getMessage());
347         }
348     }
349
```

FormDatosComponente.java

```
352     private void tablaHardwareKeyReleased(java.awt.event.KeyEvent evt) {
353         int numFila = tablaHardware.getSelectedRow();
354         if(numFila>=0){
355             Hardware hardware = obtenerObjetoDeLaTablaHardware();
356             botonNuevaMarca.setEnabled(true);
357             textoDescripcionHardware.setText(hardware.getNombre());
358             comboMarca.removeAllItems();
359             listaMarca = new ArrayList();
360             for(LineaMarcaHadware lineaMarcaHadware : hardware.getLineaMarcaHadwares()){
361                 listaMarca.add(lineaMarcaHadware.getMarca());
362             }
363             for(Marca marca : listaMarca){
364                 comboMarca.addItem(marca.getNombremarca());
365             }
366             if(componente.getCodC()>0)
367                 comboMarca.setSelectedItem(componente.getM().getNombremarca());
368         }else {
369             textoDescripcionHardware.setText("");
370             comboMarca.removeAllItems();
371             botonNuevaMarca.setEnabled(false);
372         }
373     }
```

FormDatosComponente.java

```
375     private void tablaHardwareMousePressed(java.awt.event.MouseEvent evt) {
376         int numFila = tablaHardware.getSelectedRow();
377         if(numFila>=0){
378             Hardware hardware = obtenerObjetoDeLaTablaHardware();
379             botonNuevaMarca.setEnabled(true);
380             textoDescripcionHardware.setText(hardware.getNombre());
381             comboMarca.removeAllItems();
382             listaMarca = new ArrayList();
383             for(LineaMarcaHadware lineaMarcaHadware : hardware.getLineaMarcaHadwares()){
384                 listaMarca.add(lineaMarcaHadware.getMarca());
385             }
386             for(Marca marca : listaMarca){
387                 comboMarca.addItem(marca.getNombremarca());
388             }
389             if(componente.getCodC()>0)
390                 comboMarca.setSelectedItem(componente.getM().getNombremarca());
391         }else {
392             textoDescripcionHardware.setText("");
393             comboMarca.removeAllItems();
394             botonNuevaMarca.setEnabled(false);
395         }
396     }
```

FormDatosComponente.java

```
398     private void tablaHardwareMouseDragged(java.awt.event.MouseEvent evt) {
399         int numFila = tablaHardware.getSelectedRow();
400         if(numFila>=0){
401             Hardware hardware = obtenerObjetoDeLaTablaHardware();
402             botonNuevaMarca.setEnabled(true);
403             textoDescripcionHardware.setText(hardware.getNombre());
404             comboMarca.removeAllItems();
405             listaMarca = new ArrayList();
406             for(LineaMarcaHadware lineaMarcaHadware : hardware.getLineaMarcaHadwares()){
407                 listaMarca.add(lineaMarcaHadware.getMarca());
408             }
409             for(Marca marca : listaMarca){
410                 comboMarca.addItem(marca.getNombremarca());
411             }
412             if(componente.getCodC()>0)
413                 comboMarca.setSelectedItem(componente.getM().getNombremarca());
414         }else {
415             textoDescripcionHardware.setText("");
416             comboMarca.removeAllItems();
417             botonNuevaMarca.setEnabled(false);
418         }
419     }
```

FormDatosComponente.java

```
426     private void botonGuardarActionPerformed(java.awt.event.ActionEvent evt) {
427         try {
428                 Hardware hardware = obtenerObjetoDeLaTablaHardware();
429                 Marca marca = obtenerObjetoMarca();
                    String descripcion = textoDescripcionComponente.getText().trim().toUpperCase();
431                 if(marca!=null && !descripcion.isEmpty()){
432                         componente.setH(hardware);
433                         componente.setM(marca);
434                         componente.setDescC(descripcion);
435                         GestionarComponenteServicio componenteServicio = new GestionarComponenteServicio()
436                         if(componente.getCodC()==0){
437                             componenteServicio.crear(componente);
438                             Mensaje.Mostrar_MENSAJE_GUARDADOEXITOSO(this);
439                             dispose();
440                         }else{
441                             componenteServicio.modificar(componente);
442                             Mensaje.Mostrar_MENSAJE_MODIFICADOEXITOSO(this);
443                             dispose();
444                         }
445                 }else
446                     Mensaje.Mostrar_MENSAJE_LLENARCAMPOSOBLIGATORIOS(this);
447         } catch (Exception e) {
448             Mensaje.mostrarErrorExcepcion(this,e.getMessage());
449         }
450     }
```

FormDatosComponente.java

```
457     private void botonNuevaMarcaActionPerformed(java.awt.event.ActionEvent evt) {
458         if(!textoDescripcionHardware.getText().isEmpty()){
459             Hardware hardware = obtenerObjetoDeLaTablaHardware();
460             FormDatosHardware datosHardware = new FormDatosHardware(null, hardware,false);
461             datosHardware.setVisible(true);
462             int numFila = tablaHardware.getSelectedRow();
463             if(numFila>=0){
464                 Hardware hardwares = obtenerObjetoDeLaTablaHardware();
465                 botonNuevaMarca.setEnabled(true);
466                 textoDescripcionHardware.setText(hardwares.getNombre());
467                 comboMarca.removeAllItems();
468                 listaMarca = new ArrayList();
469                 for(LineaMarcaHadware lineaMarcaHadware : hardwares.getLineaMarcaHadwares()){
470                     listaMarca.add(lineaMarcaHadware.getMarca());
471                 }
472                 for(Marca marca : listaMarca){
473                     comboMarca.addItem(marca.getNombremarca());
474                 }
475                 if(componente.getCodC()>0)
476                     comboMarca.setSelectedItem(componente.getM().getNombremarca());
477             }else {
478                 textoDescripcionHardware.setText("");
479                 comboMarca.removeAllItems();
480                 botonNuevaMarca.setEnabled(false);
481             }
482         }
483     }
```

b) FormDatosComponentes.java (**SIAEC02**)

Una vez aplicada la técnica de refactorización **extraer método**, el código se visualizará de la siguiente manera:

FormDatosComponente.java

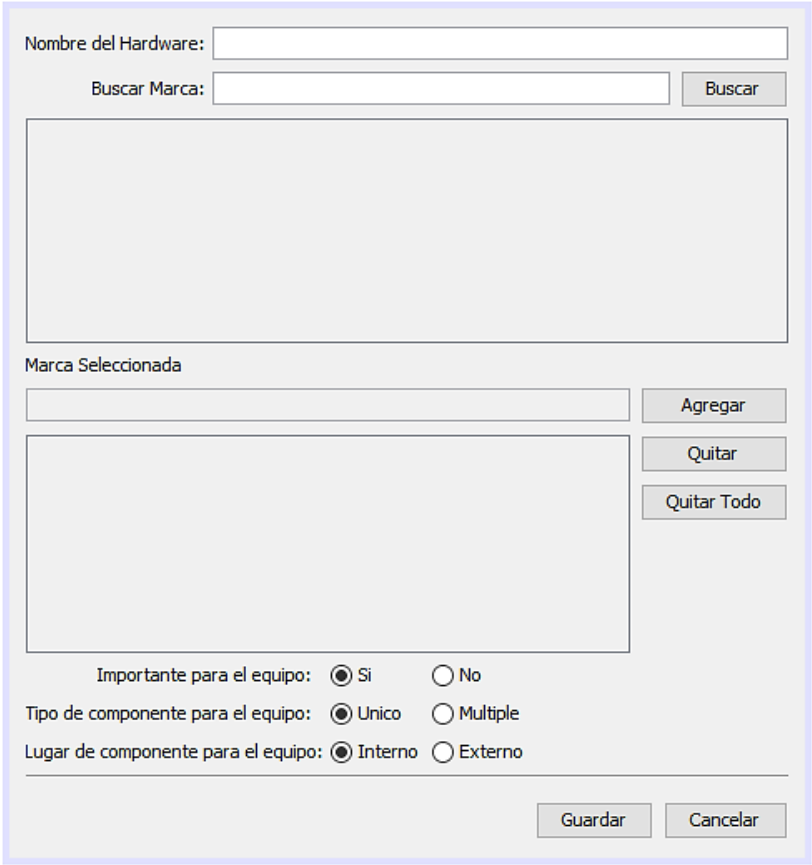
```
341     private void textoBuscarHardwareKeyTyped(java.awt.event.KeyEvent evt) {
342         buscaHardware();
343         mostrarDatosHardwareYMarca();
344     }
345
346     private void botonBuscarHardwareActionPerformed(java.awt.event.ActionEvent evt) {
347         buscaHardware();
348     }
349
350     private void tablaHardwareKeyReleased(java.awt.event.KeyEvent evt) {
351         mostrarDatosHardwareYMarca();
352     }
353
354     private void tablaHardwareMousePressed(java.awt.event.MouseEvent evt) {
355         mostrarDatosHardwareYMarca();
356     }
357
358     private void tablaHardwareMouseDragged(java.awt.event.MouseEvent evt) {
359         mostrarDatosHardwareYMarca();
360     }
361
362     private void botonGuardarActionPerformed(java.awt.event.ActionEvent evt) {
363         guardarComponente();
364     }
365
366     private void botonCancelarActionPerformed(java.awt.event.ActionEvent evt) {
367         dispose();
368     }
369
```

```
370     private void botonNuevaMarcaActionPerformed(java.awt.event.ActionEvent evt) {
371         if(!textoDescripcionHardware.getText().isEmpty()){
372             Hardware hardware = obtenerObjetoDeLaTablaHardware();
373             FormDatosHardware datosHardware = new FormDatosHardware(null, hardware,false);
374             datosHardware.setVisible(true);
375             mostrarDatosHardwareYMarca();
376         }
377     }
```

Cada método recibió un nombre que expresara lo que se está haciendo realmente.

- ✓ **buscaHardware ()**
- ✓ **mostrarDatosHardwareYMarca ()**
- ✓ **guardarComponente ()**

c) FormDatosHardware.java (**SIAEC01**)



Interfaz de Datos de hardware

Visualización de código a mejorar:

Cada color de la forma que se usó para seleccionar el fragmento del código indica las similitudes que poseen.

FormDatosHardware.java

```
424     private void textoBusquedaMarcaKeyTyped(java.awt.event.KeyEvent evt) {
425         ModeloTabla modeloTablaMarca = (ModeloTabla) tablaListaMarca.getModel();
426         modeloTablaMarca.eliminarTotalFilas();
427         try {
428             GestionarHardwareServicio hardwareServicio = new GestionarHardwareServicio();
            List<Marca> listaDeMarcas = hardwareServicio.buscarMarca(textoBusquedaMarca.getText().trim().toUpperCase());
430             for (Marca marca : listaDeMarcas) {
431                 Fila fila = new Fila();
432                 fila.agregarValorCelda(marca.getCodigomarca());
433                 fila.agregarValorCelda(marca.getNombremarca());
434                 modeloTablaMarca.agregarFila(fila);
435             }
436             modeloTablaMarca.refrescarDatos();
437         } catch (Exception e) {
438             Mensaje.mostrarErrorExcepcion(this, e.getMessage());
439         }
440         int numFila = tablaListaMarca.getSelectedRow();
441         if(numFila>=0){
442         Marca marca = obtenerObjetoDeLaTablaMarca();
443         textoDescripcionMarca.setText(marca.getNombremarca());
444         }else
445             textoDescripcionMarca.setText("");
446     }
```

FormDatosHardware.java

```
448     private void botonMarcaActionPerformed(java.awt.event.ActionEvent evt) {
449         ModeloTabla modeloTablaMarca = (ModeloTabla) tablaListaMarca.getModel();
450         modeloTablaMarca.eliminarTotalFilas();
451         try {
452             GestionarHardwareServicio hardwareServicio = new GestionarHardwareServicio();
            List<Marca> listaDeMarcas = hardwareServicio.buscarMarca(textoBusquedaMarca.getText().trim().toUpperCase());
454             for (Marca marca : listaDeMarcas) {
455                 Fila fila = new Fila();
456                 fila.agregarValorCelda(marca.getCodigomarca());
457                 fila.agregarValorCelda(marca.getNombremarca());
458                 modeloTablaMarca.agregarFila(fila);
459             }
460             modeloTablaMarca.refrescarDatos();
461         } catch (Exception e) {
462             Mensaje.mostrarErrorExcepcion(this, e.getMessage());
463         }
464         int numFila = tablaListaMarca.getSelectedRow();
465         if(numFila>=0){
466         Marca marca = obtenerObjetoDeLaTablaMarca();
467         textoDescripcionMarca.setText(marca.getNombremarca());
468         }else
469             textoDescripcionMarca.setText("");
470     }
```

FormDatosHardware.java

```
472     private void tablaListaMarcaKeyReleased(java.awt.event.KeyEvent evt) {
473         int numFila = tablaListaMarca.getSelectedRow();
474         if(numFila>=0){
475         Marca marca = obtenerObjetoDeLaTablaMarca();
476         textoDescripcionMarca.setText(marca.getNombremarca());
477         }else
478             textoDescripcionMarca.setText("");
479     }
480 
481     private void tablaListaMarcaMousePressed(java.awt.event.MouseEvent evt) {
482         int numFila = tablaListaMarca.getSelectedRow();
483         if(numFila>=0){
484         Marca marca = obtenerObjetoDeLaTablaMarca();
485         textoDescripcionMarca.setText(marca.getNombremarca());
486         }else
487             textoDescripcionMarca.setText("");
488     }
489 
490     private void tablaListaMarcaMouseDragged(java.awt.event.MouseEvent evt) {
491         int numFila = tablaListaMarca.getSelectedRow();
492         if(numFila>=0){
493         Marca marca = obtenerObjetoDeLaTablaMarca();
494         textoDescripcionMarca.setText(marca.getNombremarca());
495         }else
496             textoDescripcionMarca.setText("");
497     }
```

FormDatosHardware.java

```
499     private void botonAgregarActionPerformed(java.awt.event.ActionEvent evt) {
500         ModeloTabla modeloTablaLineaMarca = (ModeloTabla)tablaLineaMarca.getModel();
501          try {
502              if(!textoDescripcionMarca.getText().isEmpty()){
503                 Marca marca = obtenerObjetoDeLaTablaMarca();
504                 LineaMarcaHadware lineaMarcaHadware = new LineaMarcaHadware();
505                 lineaMarcaHadware.setMarca(marca);
506                 hardware.agregarLineaMarca(lineaMarcaHadware);
507                 modeloTablaLineaMarca.eliminarTotalFilas();
508                 for(LineaMarcaHadware marcaHadware: hardware.getLineaMarcaHadwares()){
509                     Fila fila = new Fila();
510                     fila.agregarValorCelda(marcaHadware.getMarca().getCodigomarca());
511                     fila.agregarValorCelda(marcaHadware.getMarca().getNombremarca());
512                     modeloTablaLineaMarca.agregarFila(fila);
513                 }
514                 modeloTablaLineaMarca.refrescarDatos();
515             }else
516                  Mensaje.mostrarErrorExcepcion(this,"Debe Buscar una marca para asignar.");
517         } catch (Exception e) {
518             Mensaje.mostrarErrorExcepcion(this,e.getMessage());
519         }
520     }
```

FormDatosHardware.java

```
545     private void botonGuardarActionPerformed(java.awt.event.ActionEvent evt) {
546         GestionarHardwareServicio gestionarHardwareServicio = new GestionarHardwareServicio();
            String nombrehardwrae= textoNombreHardware.getText().trim().toUpperCase();
548         try {
549             if(!nombrehardwrae.isEmpty()){
550                 hardware.setNombre(nombrehardwrae);
551                 if(rbunico.isSelected())
552                     hardware.setUnico(true);
553                 if(rbmultiple.isSelected())
554                     hardware.setUnico(false);
555                 if(rbIntenro.isSelected())
556                     hardware.setInterno(true);
557                 if(rbExterno.isSelected())
558                     hardware.setInterno(false);
559                 if(rbSi.isSelected())
560                     hardware.setImportante(true);
561                 if(rbNo.isSelected())
562                     hardware.setImportante(false);
563                 if(hardware.getCodigohardware()==0){
564                   gestionarHardwareServicio.crear(hardware);
565                   Mensaje.Mostrar_MENSAJE_GUARDADOEXITOSO(this);
566                   dispose();
567                 }else{
568                     gestionarHardwareServicio.modificar(hardware);
569                     Mensaje.Mostrar_MENSAJE_MODIFICADOEXITOSO(this);
570                     dispose();
571                 }
572             }else
573                 Mensaje.Mostrar_MENSAJE_LLENARCAMPOSOBLIGATORIOS(this);
574         } catch (Exception e) {
575             Mensaje.mostrarErrorExcepcion(this, e.getMessage());
576         }
```

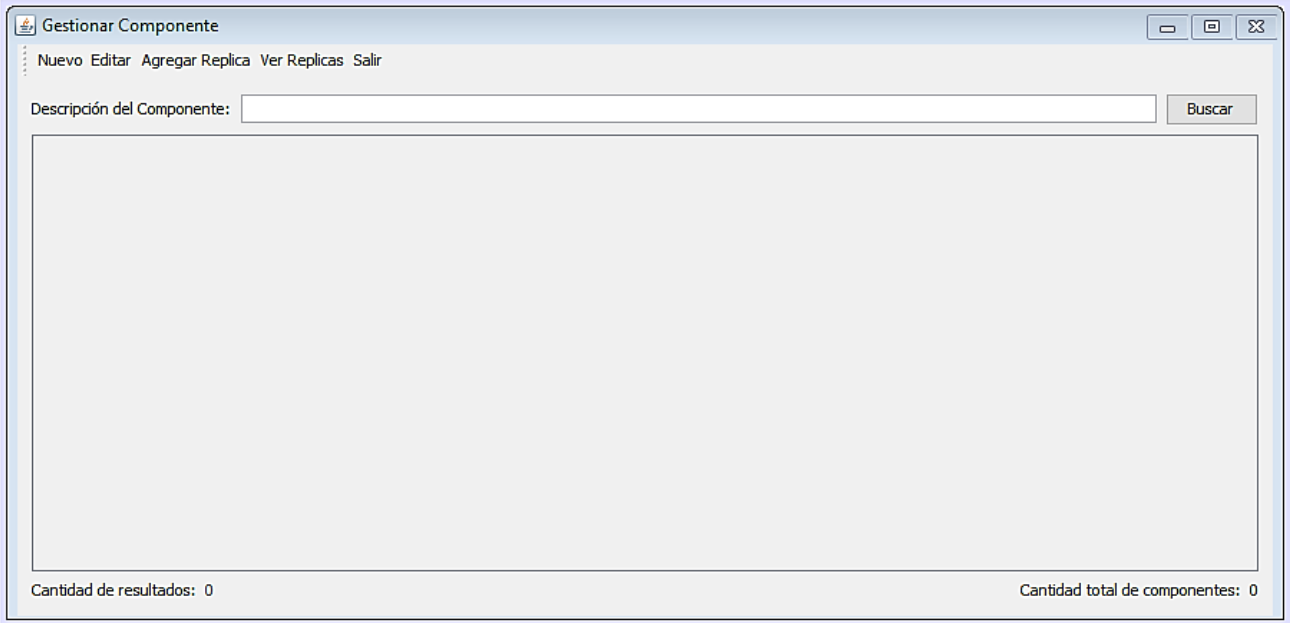
d) FormDatosHardware.java (**SIAEC02**)

Una vez aplicada la técnica de refactorización **extraer método**, el código se visualizará de la siguiente manera:

Cada método recibió un nombre que expresara lo que se está haciendo realmente.

- ✓ **buscarMarca ()**
- ✓ **mostrarDatosMarca ()**
- ✓ **agregarLineaMarca ()**
- ✓ **guardarHardware ()**

e) FormGestionarComponente.java (**SIAEC01**)



Interfaz Gestionar Componentes

Visualización de código a mejorar:

Cada color de la forma que se usó para seleccionar el fragmento del código indica las similitudes que poseen.

FormGestionarComponente.java

```
34      public FormGestionarComponente(Frame frame, Usuario usuario) {
35          initComponents();
36          this.frame=frame;
37          Tabla tabla = new Tabla();
38          tabla.agregarColumna(new Columna("CODIGO", "java.lang.Integer"));
39          tabla.agregarColumna(new Columna("DESCRIPCI\u00D3N", "java.lang.String"));
40          tabla.agregarColumna(new Columna("MARCA", "java.lang.String"));
41          tabla.agregarColumna(new Columna("HARDWARE", "java.lang.String"));
42          tabla.agregarColumna(new Columna("CANTIDAD", "java.lang.String"));
43          ModeloTabla modeloTablaArea = new ModeloTabla(tabla);
44          tablaComponente.setModel(modeloTablaArea);
45          EstiloTabla.estilodeTabla(tablaComponente);
46          TableColumn columna0,columna1, columna2, columna3, columna4;
47          //CODIGO
48          columna0 = tablaComponente.getColumnModel().getColumn(0);
49          tablaComponente.removeColumn(columna0);
50          //DESCRIPCION
51          columna1 = tablaComponente.getColumnModel().getColumn(0);
52          columna1.setPreferredWidth(700);
53          //MARCA
54          columna2 = tablaComponente.getColumnModel().getColumn(1);
55          columna2.setPreferredWidth(300);
56          //HARDWARE
57          columna3 = tablaComponente.getColumnModel().getColumn(2);
58          columna3.setPreferredWidth(350);
59          //CANTIDAD
60          columna4 = tablaComponente.getColumnModel().getColumn(3);
61          columna4.setPreferredWidth(170);
62
63          this.usuario=usuario;
64      }
```

FormGestionarComponente.java

```
275     private void botonNuevoActionPerformed(java.awt.event.ActionEvent evt) {
276         new FormDatosComponente(frame).setVisible(true);
277         ModeloTabla modeloTablaComponente = (ModeloTabla) tablaComponente.getModel();
278         modeloTablaComponente.eliminarTotalFilas();
279         try {
280             GestionarComponenteServicio componenteServicio = new GestionarComponenteServicio();
281             List<Componente> listaDeComponentes = componenteServicio.buscar
                        (textoDescripcionComponente.getText().trim().toUpperCase());
283             etiquetaResultados.setText(String.valueOf(listaDeComponentes.size()));
284             etiquetaCantidadComponentes.setText(String.valueOf(Componente.totComp(listaDeComponentes)));
285                 for (Componente componente : listaDeComponentes) {
286                     Fila fila = new Fila();
287                     fila.agregarValorCelda(componente.getCodC());
288                     fila.agregarValorCelda(componente.getDescC());
289                     fila.agregarValorCelda(componente.getM().getNombremarca());
290                     fila.agregarValorCelda(componente.getH().getNombre());
291                     fila.agregarValorCelda(String.valueOf(componente.getCantC()));
292                     modeloTablaComponente.agregarFila(fila);
293                 }
294                 modeloTablaComponente.refrescarDatos();
295             } catch (Exception e) {
296                 Mensaje.mostrarErrorExcepcion(this, e.getMessage());
297             }
298     }
```

FormGestionarComponente.java

```
300     private void menureplicaActionPerformed(java.awt.event.ActionEvent evt) {
301         Componente componente = obtenerObjetoDeLaTablaComponente();
302         new FormDatosComponenteReplica(frame,componente,usuario).setVisible(true);
303         ModeloTabla modeloTablaComponente = (ModeloTabla) tablaComponente.getModel();
304         modeloTablaComponente.eliminarTotalFilas();
305         try {
306             GestionarComponenteServicio componenteServicio = new GestionarComponenteServicio();
307             List<Componente> listaDeComponentes = componenteServicio.buscar
                        (textoDescripcionComponente.getText().trim().toUpperCase());
309             etiquetaResultados.setText(String.valueOf(listaDeComponentes.size()));
310             etiquetaCantidadComponentes.setText(String.valueOf(Componente.totComp(listaDeComponentes)));
311                 for (Componente componentes : listaDeComponentes) {
312                     Fila fila = new Fila();
313                     fila.agregarValorCelda(componentes.getCodC());
314                     fila.agregarValorCelda(componentes.getDescC());
315                     fila.agregarValorCelda(componentes.getM().getNombremarca());
316                     fila.agregarValorCelda(componentes.getH().getNombre());
317                     fila.agregarValorCelda(String.valueOf(componentes.getCantC()));
318                     modeloTablaComponente.agregarFila(fila);
319                 }
320                 modeloTablaComponente.refrescarDatos();
321             } catch (Exception e) {
322                 Mensaje.mostrarErrorExcepcion(this, e.getMessage());
323             }
324     }
```

FormGestionarComponente.java

```
326     private void botonAgregarReplicaActionPerformed(java.awt.event.ActionEvent evt) {
327         Componente componente = obtenerObjetoDeLaTablaComponente();
328         new FormDatosComponenteReplica(frame,componente,usuario).setVisible(true);
329         ModeloTabla modeloTablaComponente = (ModeloTabla) tablaComponente.getModel();
330         modeloTablaComponente.eliminarTotalFilas();
331         try {
332             GestionarComponenteServicio componenteServicio = new GestionarComponenteServicio();
333             List<Componente> listaDeComponentes = componenteServicio.buscar
                        (textoDescripcionComponente.getText().trim().toUpperCase());
335             etiquetaResultados.setText(String.valueOf(listaDeComponentes.size()));
336             etiquetaCantidadComponentes.setText(String.valueOf(Componente.totComp(listaDeComponentes)));
337                 for (Componente componentes : listaDeComponentes) {
338                     Fila fila = new Fila();
339                     fila.agregarValorCelda(componentes.getCodC());
340                     fila.agregarValorCelda(componentes.getDescC());
341                     fila.agregarValorCelda(componentes.getM().getNombremarca());
342                     fila.agregarValorCelda(componentes.getH().getNombre());
343                     fila.agregarValorCelda(String.valueOf(componentes.getCantC()));
344                     modeloTablaComponente.agregarFila(fila);
345                 }
346                 modeloTablaComponente.refrescarDatos();
347         } catch (Exception e) {
348             Mensaje.mostrarErrorExcepcion(this, e.getMessage());
349         }
350     }
```

FormGestionarComponente.java

```
352     private void textoDescripcionComponenteKeyTyped(java.awt.event.KeyEvent evt) {
353         ModeloTabla modeloTablaComponente = (ModeloTabla) tablaComponente.getModel();
354         modeloTablaComponente.eliminarTotalFilas();
355         try {
356             GestionarComponenteServicio componenteServicio = new GestionarComponenteServicio();
357             List<Componente> listaDeComponentes = componenteServicio.buscar
                        (textoDescripcionComponente.getText().trim().toUpperCase());
359             etiquetaResultados.setText(String.valueOf(listaDeComponentes.size()));
360             etiquetaCantidadComponentes.setText(String.valueOf(Componente.totComp(listaDeComponentes)));
361                 for (Componente componente : listaDeComponentes) {
362                     Fila fila = new Fila();
363                     fila.agregarValorCelda(componente.getCodC());
364                     fila.agregarValorCelda(componente.getDescC());
365                     fila.agregarValorCelda(componente.getM().getNombremarca());
366                     fila.agregarValorCelda(componente.getH().getNombre());
367                     fila.agregarValorCelda(String.valueOf(componente.getCantC()));
368                     modeloTablaComponente.agregarFila(fila);
369                 }
370                 modeloTablaComponente.refrescarDatos();
371         } catch (Exception e) {
372             Mensaje.mostrarErrorExcepcion(this, e.getMessage());
373         }
374     }
```

FormGestionarComponente.java

```
376     private void botonBuscarComponenteActionPerformed(java.awt.event.ActionEvent evt) {
377         ModeloTabla modeloTablaComponente = (ModeloTabla) tablaComponente.getModel();
378         modeloTablaComponente.eliminarTotalFilas();
379         try {
380             GestionarComponenteServicio componenteServicio = new GestionarComponenteServicio();
381             List<Componente> listaDeComponentes = componenteServicio.buscar
                        (textoDescripcionComponente.getText().trim().toUpperCase());
383             etiquetaResultados.setText(String.valueOf(listaDeComponentes.size()));
384             etiquetaCantidadComponentes.setText(String.valueOf(Componente.totComp(listaDeComponentes)));
385                 for (Componente componente : listaDeComponentes) {
386                     Fila fila = new Fila();
387                     fila.agregarValorCelda(componente.getCodC());
388                     fila.agregarValorCelda(componente.getDescC());
389                     fila.agregarValorCelda(componente.getM().getNombremarca());
390                     fila.agregarValorCelda(componente.getH().getNombre());
391                     fila.agregarValorCelda(String.valueOf(componente.getCantC()));
392                     modeloTablaComponente.agregarFila(fila);
393                 }
394                 modeloTablaComponente.refrescarDatos();
395         } catch (Exception e) {
396             Mensaje.mostrarErrorExcepcion(this, e.getMessage());
397         }
398     }
```

FormGestionarComponente.java

```
420     private void botonEditarActionPerformed(java.awt.event.ActionEvent evt) {
421         Componente componente = obtenerObjetoDeLaTablaComponente();
422         if(componente!=null){
423             FormDatosComponente datosComponente= new FormDatosComponente(null, componente);
424             datosComponente.setVisible(true);
425             ModeloTabla modeloTablaComponente = (ModeloTabla) tablaComponente.getModel();
426             modeloTablaComponente.eliminarTotalFilas();
427             try {
428                 GestionarComponenteServicio componenteServicio = new GestionarComponenteServicio();
429                 List<Componente> listaDeComponentes = componenteServicio.buscar
                            (textoDescripcionComponente.getText().trim().toUpperCase());
431                 etiquetaResultados.setText(String.valueOf(listaDeComponentes.size()));
432                 etiquetaCantidadComponentes.setText(String.valueOf(Componente.totComp(listaDeComponentes)));
433                     for (Componente componentes : listaDeComponentes) {
434                         Fila fila = new Fila();
435                         fila.agregarValorCelda(componentes.getCodC());
436                         fila.agregarValorCelda(componentes.getDescC());
437                         fila.agregarValorCelda(componentes.getM().getNombremarca());
438                         fila.agregarValorCelda(componentes.getH().getNombre());
439                         fila.agregarValorCelda(String.valueOf(componentes.getCantC()));
440                         modeloTablaComponente.agregarFila(fila);
441                     }
442                     modeloTablaComponente.refrescarDatos();
443             } catch (Exception e) {
444                 Mensaje.mostrarErrorExcepcion(this, e.getMessage());
445             }
446         }
447     }
```

FormGestionarComponente.java

```
450     private void menueditarActionPerformed(java.awt.event.ActionEvent evt) {
451         Componente componente = obtenerObjetoDeLaTablaComponente();
452         if(componente!=null){
453             FormDatosComponente datosComponente= new FormDatosComponente(null, componente);
454             datosComponente.setVisible(true);
455             ModeloTabla modeloTablaComponente = (ModeloTabla) tablaComponente.getModel();
456             modeloTablaComponente.eliminarTotalFilas();
457             try {
458                 GestionarComponenteServicio componenteServicio = new GestionarComponenteServicio();
459                 List<Componente> listaDeComponentes = componenteServicio.buscar
                            (textoDescripcionComponente.getText().trim().toUpperCase());
461                 etiquetaResultados.setText(String.valueOf(listaDeComponentes.size()));
462                 etiquetaCantidadComponentes.setText(String.valueOf(Componente.totComp(listaDeComponentes)));
463                     for (Componente componentes : listaDeComponentes) {
464                         Fila fila = new Fila();
465                         fila.agregarValorCelda(componentes.getCodC());
466                         fila.agregarValorCelda(componentes.getDescC());
467                         fila.agregarValorCelda(componentes.getM().getNombremarca());
468                         fila.agregarValorCelda(componentes.getH().getNombre());
469                         fila.agregarValorCelda(String.valueOf(componentes.getCantC()));
470                         modeloTablaComponente.agregarFila(fila);
471                     }
472                     modeloTablaComponente.refrescarDatos();
473             } catch (Exception e) {
474                 Mensaje.mostrarErrorExcepcion(this, e.getMessage());
475             }
476         }
477     }
```

f) FormGestionarComponente.java (**SIAEC02**)

Una vez aplicada la técnica de refactorización **extraer método**, el código se visualizará de la siguiente manera:

```
25      public FormGestionarComponente(Frame frame, Usuario usuario) {
26          initComponents();
27          this.frame=frame;
28          crearTabla();
29          this.usuario=usuario;
30      }
```

FormGestionarComponente.java

```
280     private void botonNuevoActionPerformed(java.awt.event.ActionEvent evt) {
281         new FormDatosComponente(frame).setVisible(true);
282         buscaComponente();
283     }
284
285     private void menureplicaActionPerformed(java.awt.event.ActionEvent evt) {
286         Componente componente = obtenerObjetoDeLaTablaComponente();
287         new FormDatosComponenteReplica(frame,componente,usuario).setVisible(true);
288         buscaComponente();
289     }
290
291     private void botonAgregarReplicaActionPerformed(java.awt.event.ActionEvent evt) {
292         Componente componente = obtenerObjetoDeLaTablaComponente();
293         new FormDatosComponenteReplica(frame,componente,usuario).setVisible(true);
294         buscaComponente();
295     }
296
297     private void textoDescripcionComponenteKeyTyped(java.awt.event.KeyEvent evt) {
298         buscaComponente();
299     }
300
301     private void botonBuscarComponenteActionPerformed(java.awt.event.ActionEvent evt) {
302         buscaComponente();
303     }
```

```
325     private void botonEditarActionPerformed(java.awt.event.ActionEvent evt) {
326         modificarComponente();
327     }
328
329     private void menueditarActionPerformed(java.awt.event.ActionEvent evt) {
330         modificarComponente();
331     }
```

```
333     private void modificarComponente() {
334         Componente componente = obtenerObjetoDeLaTablaComponente();
335         if(componente!=null){
336             FormDatosComponente datosComponente= new FormDatosComponente(null, componente);
337             datosComponente.setVisible(true);
338             buscaComponente();
339         }
340     }
```

Cada método recibió un nombre que expresara lo que se está haciendo realmente.

- ✓ **crearTabla ()**
- ✓ **buscarComponente ()**
- ✓ **modificarComponente ()**

RENOMBRAR MÉTODO, CLASE O VARIABLES

Esta técnica de refactorización se usa cuando se quiere cambiar el nombre de:

- ✓ **Campo:** Cambia la declaración y los usos del terreno para el nuevo nombre.
- ✓ **Variable local:** Cambia la declaración y los usos de la variable con el nuevo nombre.
- ✓ **Método:** Cambia el nombre del método y todas las referencias a ese método por el nuevo nombre.

Cuando cambia el nombre de un método, clase o variable local, la operación de cambio de nombre se propaga a todas las instancias del método que son en su alcance, independientemente de si el método de extensión está siendo utilizado como un método estático o un método de instancia.

Esta técnica de Refactorización nos permitirá disminuir **Code Smell** y mejorar la comprensibilidad del código fuente.

Cambios que se realizaron al prototipo SIAEC, las Clases que tomamos como ejemplo son los siguientes:

- ✓ **Area.java**
- ✓ **AsignacionEquipo.java**
- ✓ **Componente.java**

✓ **VARIABLES**

(SIAEC-01) – SOFTWARE ORIGINAL Area.java

Area.java

```
12    public class Area {
13        private int codA;
14        private String nomA;
15        private String descA;
```

(SIAEC-02) – SOFTWARE REFACTORIZADO Area.java

Area.java

```
7     public class Area {
8         private int codigoarea;
9         private String nombre;
10        private String descripcion;
```

(SIAEC-01) AsignacionEquipo.java

AsignacionEquipo.java

```
18    public class AsignacionEquipo {
19        private int codAE;
20        private Date fechAE;
21        private Usuario usuAE;
22        private Empleado empAE;
23        private String motAE;
24        private String numAE;
25        private boolean vigAE;
26        private boolean actAE;
27        private List<LineaAsignacionEquipo> lineasAE;
28        private List<LineaAsignacionComponenteReplica> lineasACR;
```

(SIAEC-02) AsignacionEquipo.java

AsignacionEquipo.java

```
12    public class AsignacionEquipo {
13        private int codigoasignacionequipo;
14        private Date fechaasignacion;
15        private Usuario usuario;
16        private Empleado empleado;
17        private String motivoasignacion;
18        private String numerosisgedo;
19        private boolean vigente;
20        private boolean activo;
21        private List<LineaAsignacionEquipo> lineasAsignacionEquipo;
22        private List<LineaAsignacionComponenteReplica> lineasAsignacionComponenteReplica;
```

(SIAEC-01) Componente.java

Componente.java

```
15    public class Componente {
16        private int codC;
17        private String descC;
18        private int cantC;
19        private Marca m;
20        private Hardware h;
21        private boolean actC;
```

(SIAEC-02) Componente.java

Componente.java

```
10    public class Componente {
11        private int codigocomponente;
12        private String descripcioncomponente;
13        private int cantidadcomponente;
14        private Marca marca;
15        private Hardware hardware;
16        private boolean activocomponente;
```

✓ **METODOS**

(SIAEC-01) – SOFTWARE ORIGINAL AsignacionEquipo.java

AsignacionEquipo.java

```
163  private void valEquip()throws Exception{
164      if(lineasAE.size()>=3)
165          throw ExcepcionRegla.cErAsiEqCant();
166  }
167
168  public void quitLACR(int codCompRep){
169      for(LineaAsignacionComponenteReplica lineaAsigCR : lineasACR){
170          if(lineaAsigCR.getComponenteReplica().getCodCR()==codCompRep){
171              lineasACR.remove(lineaAsigCR);
172              break;
173          }
174      }
175  }
```

(SIAEC-02) – SOFTWARE REFACTORIZADO AsignacionEquipo.java

AsignacionEquipo.java

```
154  private void validarCantidadEquipo()throws Exception{
155      if(lineasAsignacionEquipo.size()>=3)
156          throw ExcepcionRegla.crearErrorAsignarEquipoCantidad();
157  }
158
159  public void quitarLineaAsignacionComponenteReplica(int codigocomponentereplica){
160      for(LineaAsignacionComponenteReplica
161          lineaAsignacionComponenteReplicaquitar :
162              lineasAsignacionComponenteReplica)
163          if(lineaAsignacionComponenteReplicaquitar
164                  .getComponenteReplica()
165                  .getCodigocomponentereplica()
166                  ==codigocomponentereplica){
167              lineasAsignacionComponenteReplica
168                  .remove(lineaAsignacionComponenteReplicaquitar);
169              break;
170          }
171      }
172  }
```

(SIAEC-01) Componente.java

Componente.java

```
91      public void valCEquipo()throws Exception{
92          if(true)
93              throw ExcepcionRegla.cERCEXT();
94      }
95
96      public void valCompCase()throws Exception{
97          if(false)
98              throw ExcepcionRegla.cERRCOMPINT();
99      }
100
101     public static int totComp(List<Componente> listaC) {
102         int cant=0;
103         cant = listaC.stream().map((componente) -> componente.getCantC())
104                 .reduce(cant, Integer::sum);
105         return cant;
106     }
```

(SIAEC-02) Componente.java

Componente.java

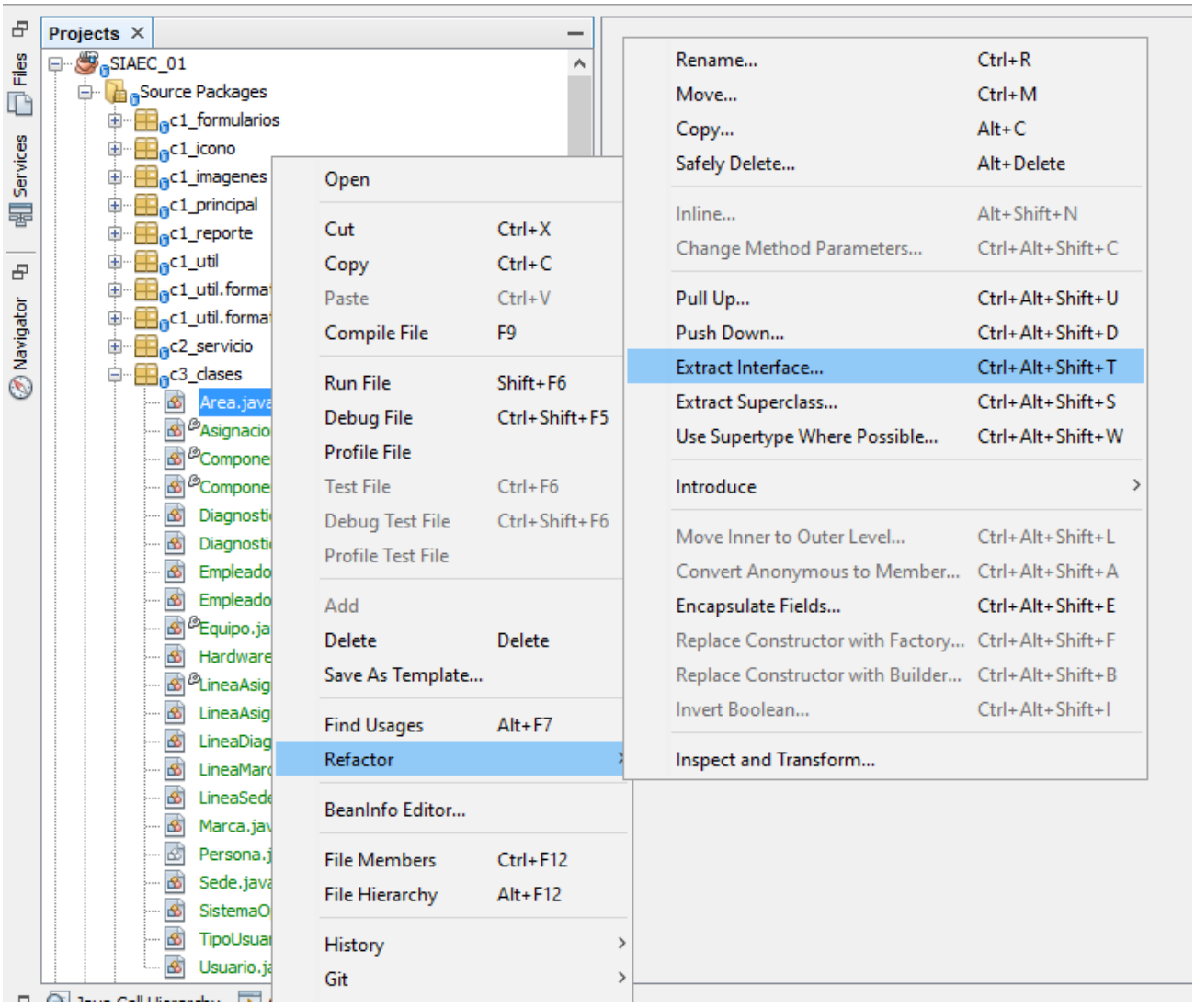
```
79      public void validarComponenteEquipo()throws Exception{
80          if(COMP_INTERNO)
81              throw ExcepcionRegla.crearErrorComponenteExterno();
82      }
83
84      public void validarComponenteCase()throws Exception{
85          if(COMP_EXTERNO)
86              throw ExcepcionRegla.crearErrorComponenteInterno();
87      }
88
89      public static int cantidadTotalComponentes(List<Componente> listaComponente){
90          int cantidadcomponente=0;
91          cantidadcomponente = listaComponente.stream()
92              .map((componente) -> componente.getCantidadComponente())
93              .reduce(cantidadcomponente, Integer::sum);
94          return cantidadcomponente;
95      }
```

EXTRAER INTERFAZ.

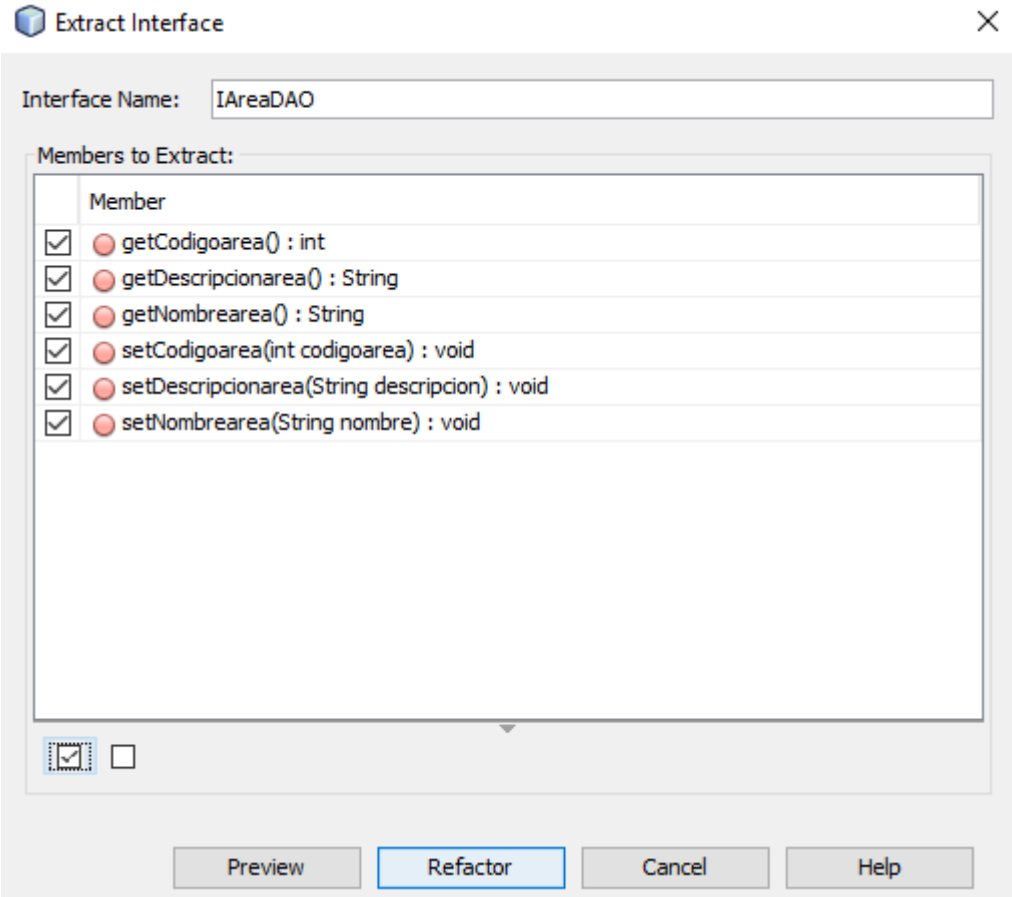
Se extraerá las Interfaces a partir de una clase.

El Sistema prototipo está desarrollada en el IDE NetBeans, este Lenguaje de Programación nos permite también aplicar esta técnica con los siguientes pasos:

a. Para obtener la Interfaz, se debe seleccionar la Clase y dar click derecho, ir a la opción Refactor y elegir la técnica Extract Interfaz.



b. Luego se mostrará una pantalla donde se debe Ingresar el nombre de la interfaz (**IAreaDAO**), debe seleccionar los métodos deseados y finalmente dar en la opción **Refactor**.



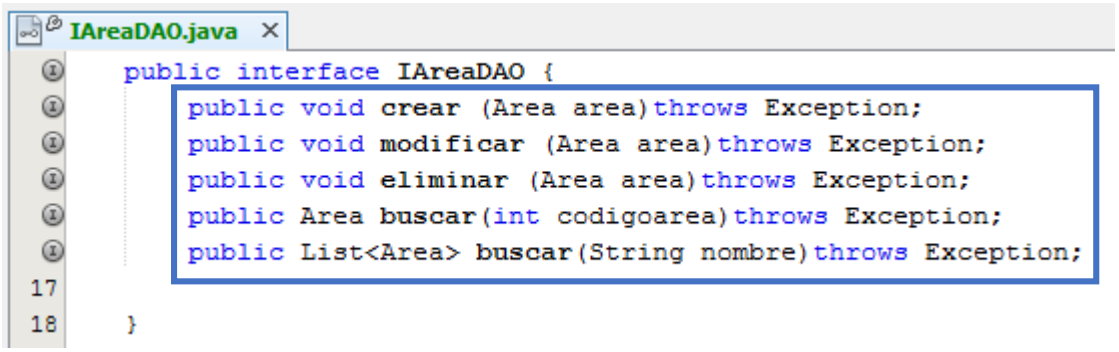
c. En la Clase **Area.java** también se hicieron modificaciones automáticamente, en este caso se eliminará la implementación (**implements IAreaDAO**) además de los @**Override**, ya que esto será implementado en otra parte del código.

```java
public class Area implements IAreaDAO{
    private int codigoarea;
    private String nombrearea;
    private String descripcionarea;

    public Area() {
        this.codigoarea=0;
    }

    public Area(String nombre, String descripcion) {
        this.nombrearea = nombre;
        this.descripcionarea = descripcion;
    }
    @Override
    public int getCodigoarea() {
        return codigoarea;
    }

    @Override
    public void setCodigoarea(int codigoarea) {
        this.codigoarea = codigoarea;
    }
```

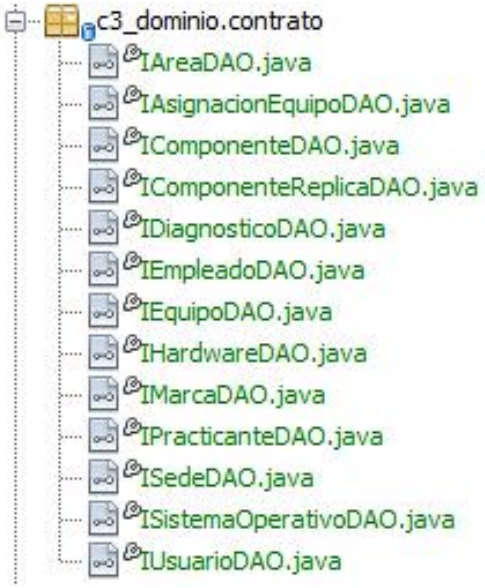
d. Luego la interfaz será creada mostrando los métodos seleccionados.

```java
public interface IAreaDAO {

    int getCodigoarea();
    String getDescripcionarea();
    String getNombrearea();
    void setCodigoarea(int codigoarea);
    void setDescripcionarea(String descripcion);
    void setNombrearea(String nombre);
}
```

e. En la Interfaz **IAreaDAO** se eliminará los métodos que se generaron por defecto y se reemplazará por los siguientes:



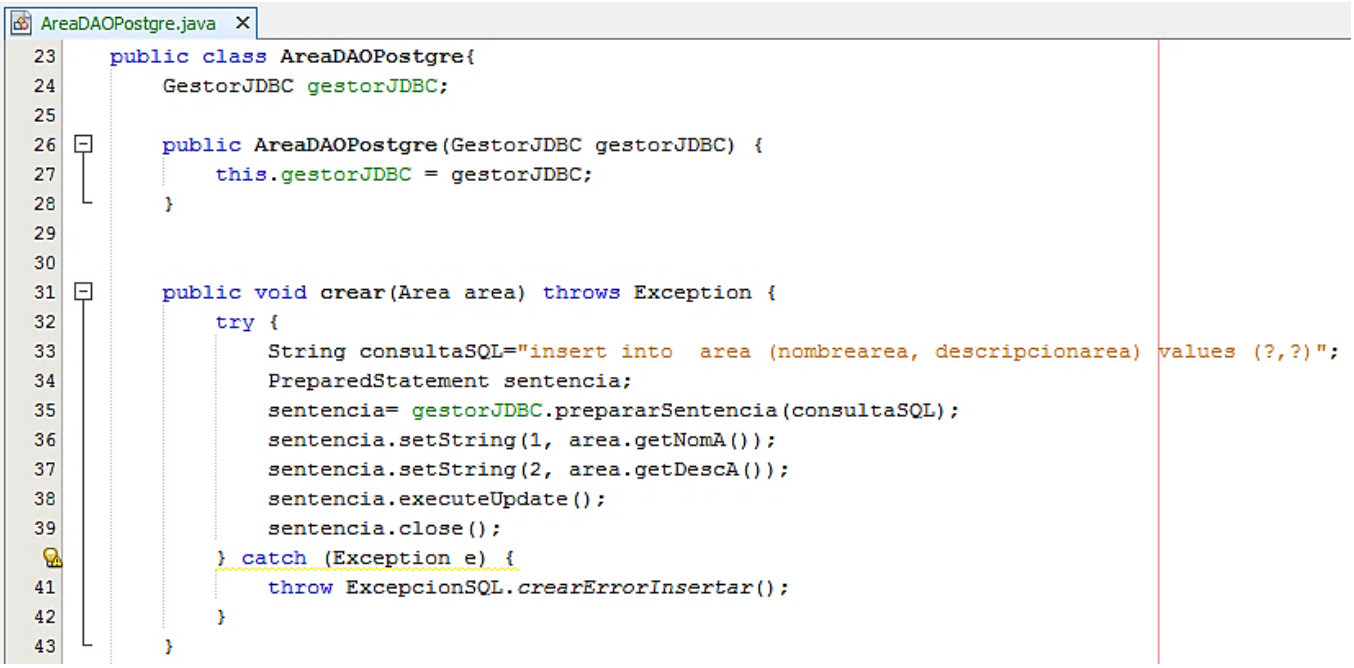
f. Cada Interfaz creada debe ser trasladada a la Capa **dominio.contrato**



g. Ir a la clase **AreaDAOPostgre.java**

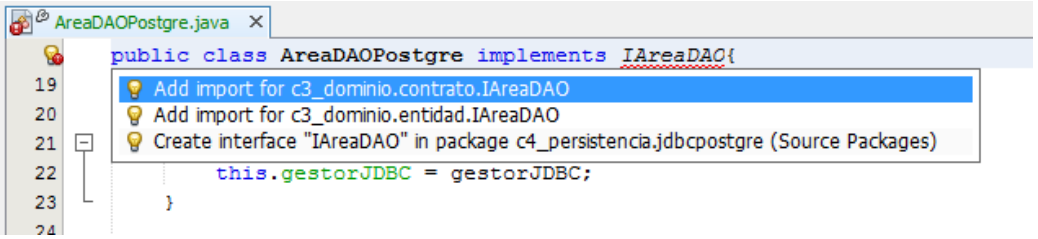
SIAEC-01

Esta es la clase donde no tiene implementado la Interfaz.

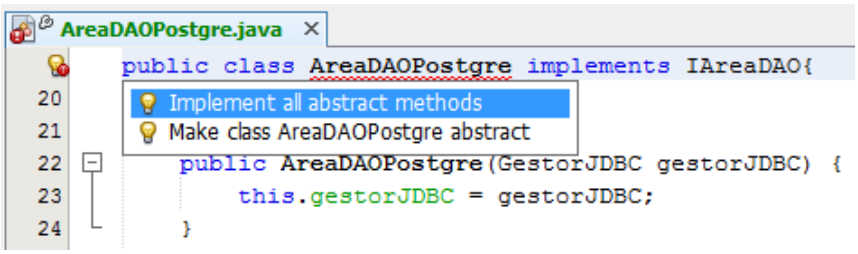


SIAEC-02

Debe implementarse la Interfaz que ha sido creado desde la clase, colocaremos lo siguiente **implements IAreaDAO**, luego se importara la librería



Una vez importado la librería, seguirá generando un error, debido a que la interfaz contiene métodos y deben ser implementadas de la siguiente manera:



AreaDAOPostgre.java

```
19  public class AreaDAOPostgre implements IAreaDAO{
20      GestorJDBC gestorJDBC;
21
22      public AreaDAOPostgre(GestorJDBC gestorJDBC) {
24          this.gestorJDBC = gestorJDBC;
24      }
25
26      @Override
27      public void crear(Area area) throws Exception {
28          throw new UnsupportedOperationException("Not supported yet.");
29      }
30
31      @Override
33      public void modificar(Area area) throws Exception {
33          throw new UnsupportedOperationException("Not supported yet.");
34      }
35
36      @Override
38      public void eliminar(Area area) throws Exception {
38          throw new UnsupportedOperationException("Not supported yet.");
39      }
40
41      @Override
43      public Area buscar(int codigoarea) throws Exception {
43          throw new UnsupportedOperationException("Not supported yet.");
44      }
45
46      @Override
48      public List<Area> buscar(String nombre) throws Exception {
48          throw new UnsupportedOperationException("Not supported yet.");
49      }
```

- ✓ **Al dar click en ese icono te llevara directo a la Interfaz.**

Como cada método ya estaba implementado, entonces colocaremos el código en cada método según corresponda.

AreaDAOPostgre.java

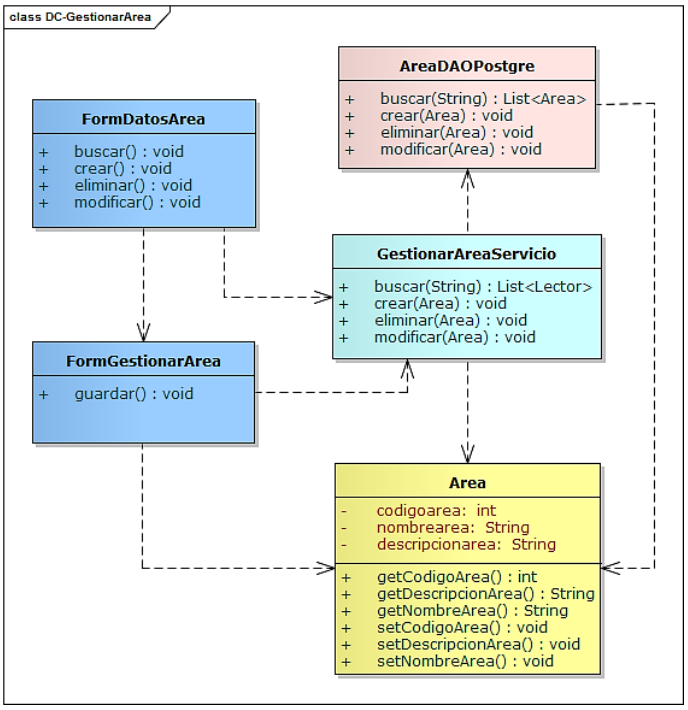
```
18  public class AreaDAOPostgre implements IAreaDAO{
19      GestorJDBC gestorJDBC;
20
21      public AreaDAOPostgre(GestorJDBC gestorJDBC) {
22          this.gestorJDBC = gestorJDBC;
23      }
24
25      @Override
        public void crear(Area area) throws Exception {
27          try {
28              String consultaSQL="insert into  area (nombrearea, descripcionarea) values (?,?)";
29              PreparedStatement sentencia;
30              sentencia= gestorJDBC.prepararSentencia(consultaSQL);
31              sentencia.setString(1, area.getNombrearea());
32              sentencia.setString(2, area.getDescripcionarea());
33              sentencia.executeUpdate();
34              sentencia.close();
            } catch (Exception e) {
36              throw ExcepcionSQL.crearErrorInsertar();
37          }
38      }
```

Todos estos pasos han sido aplicados a cada Clase existente en el Sistema prototipo SIAEC

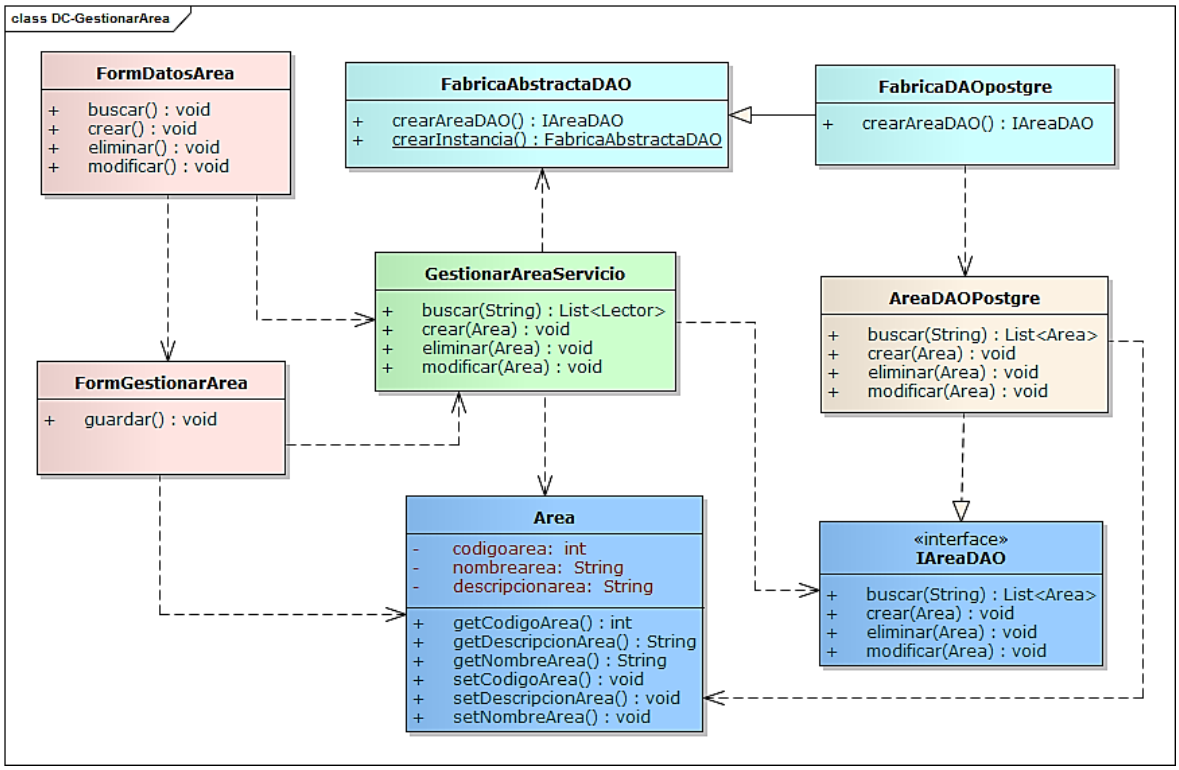
DESACOPLAMIENTO ENTRE CAPAS

Después de aplicar la refactorización de la Arquitectura, los diagramas de clase de realización de SIAEC01 (Primera Versión) y SIAEC02 (Segunda Versión), estarán compuestos de la siguiente manera:

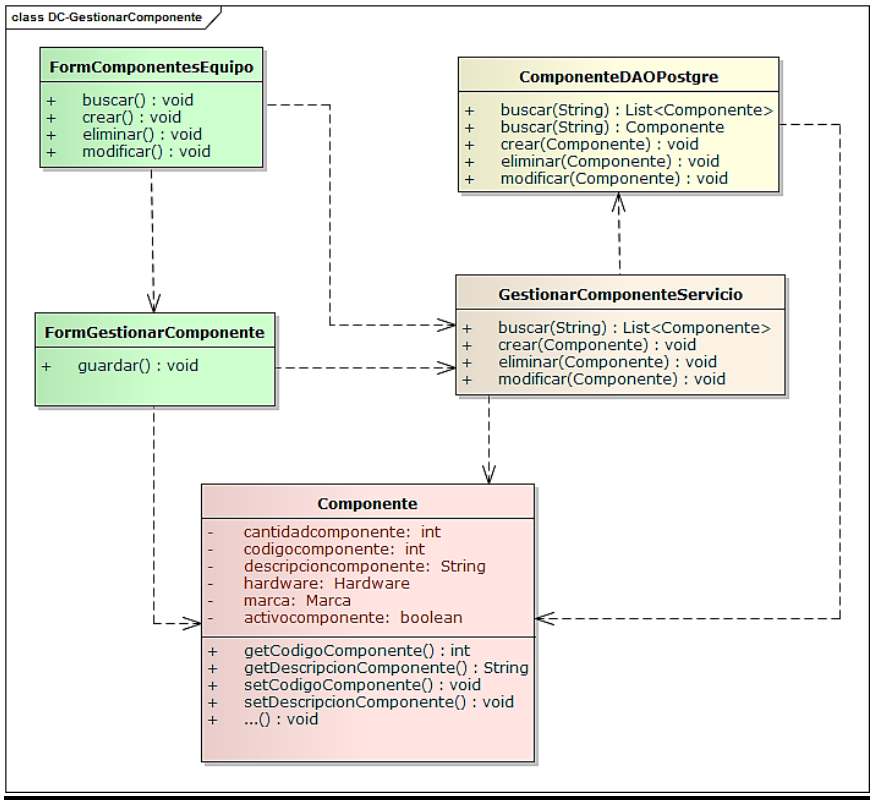
Gestionar Área SIAEC01 (Primera Versión)



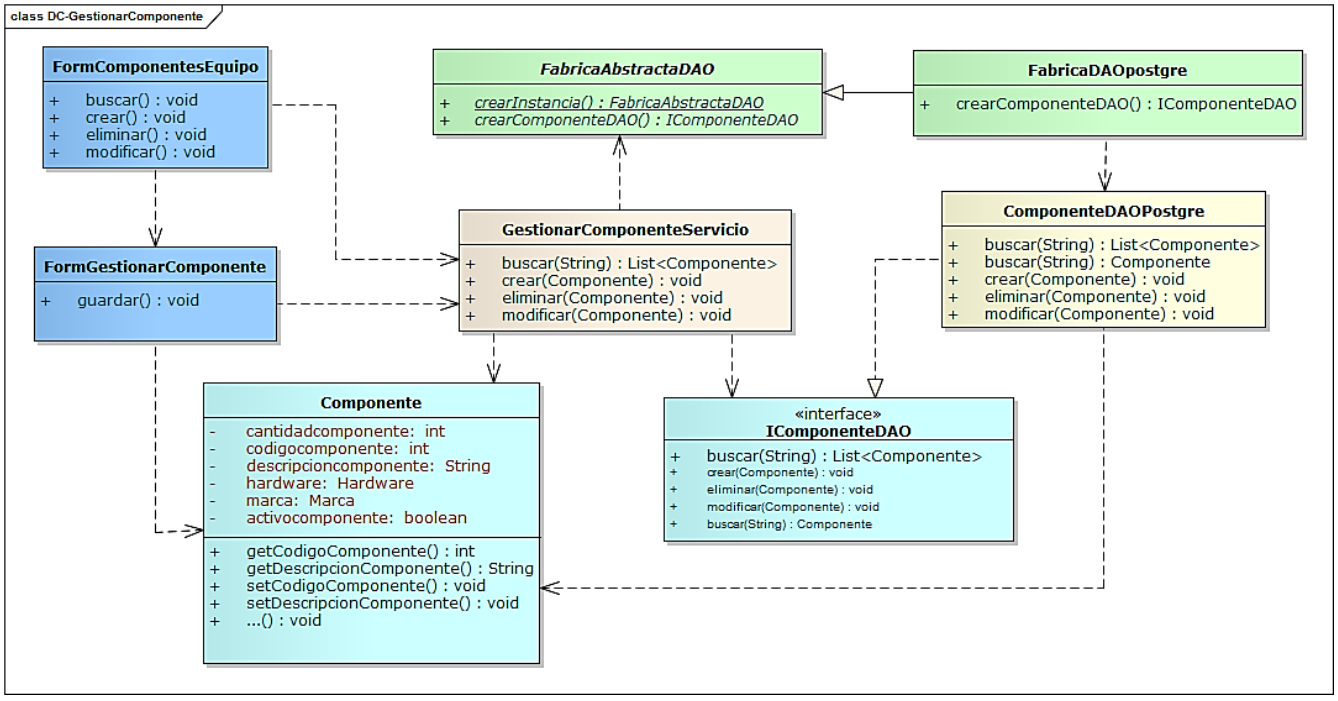
Gestionar Área SIAEC02 (Segunda Versión)



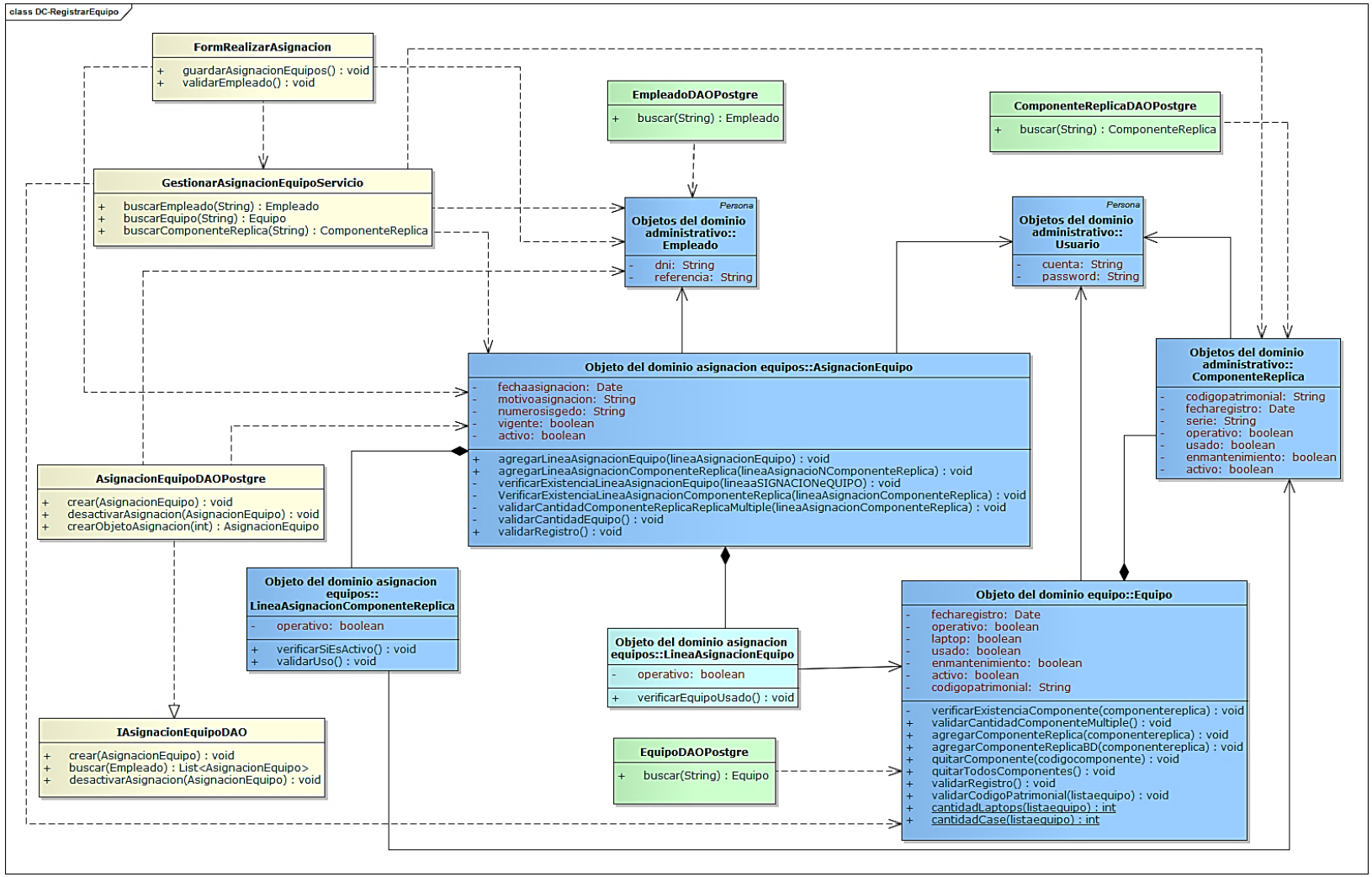
Gestionar Componente SIAEC01 (Primera Versión)



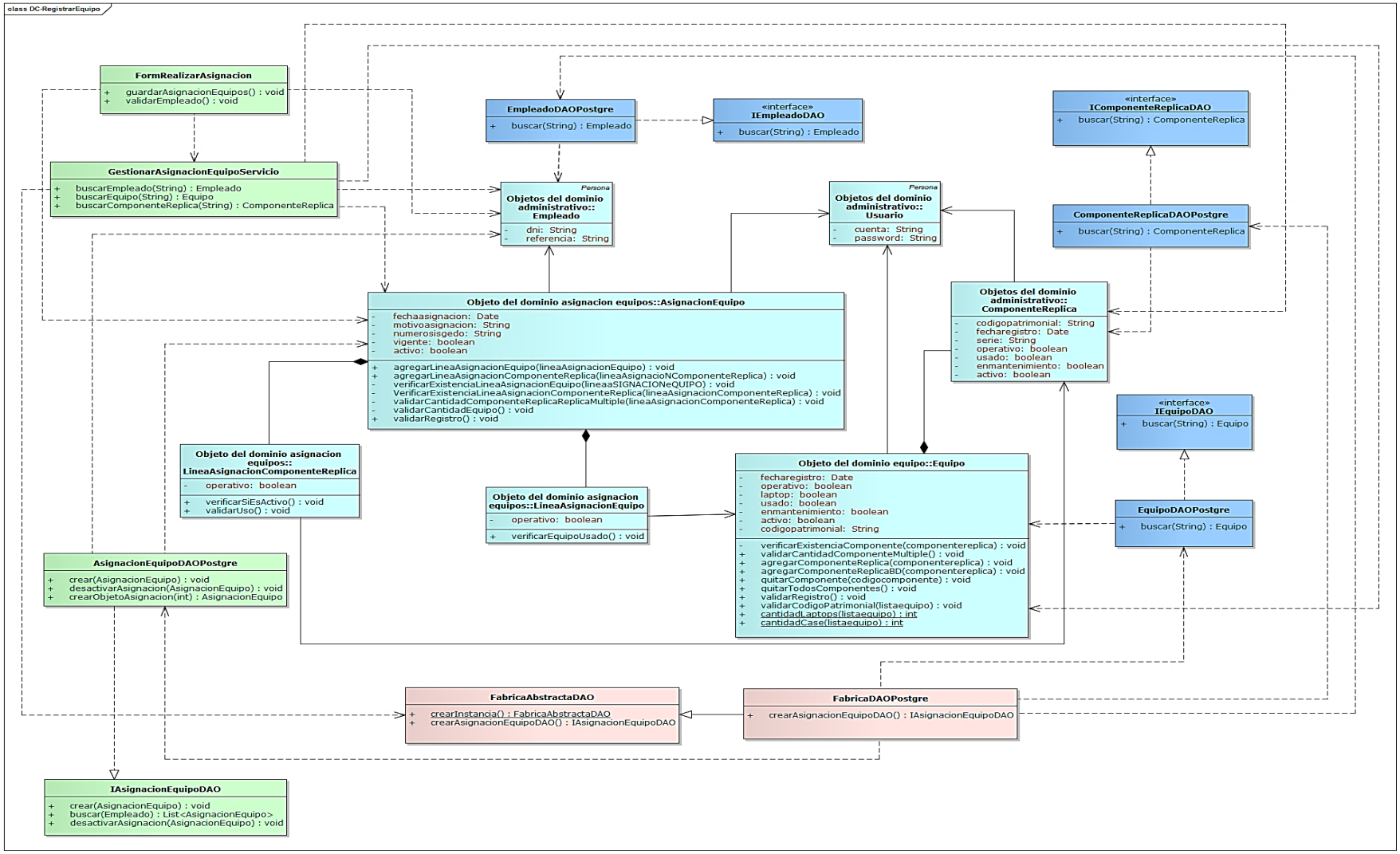
Gestionar Componente SIAEC02 (Segunda Versión)



RegistrarAsignacionEquipo SIAEC01 (Primera Versión)

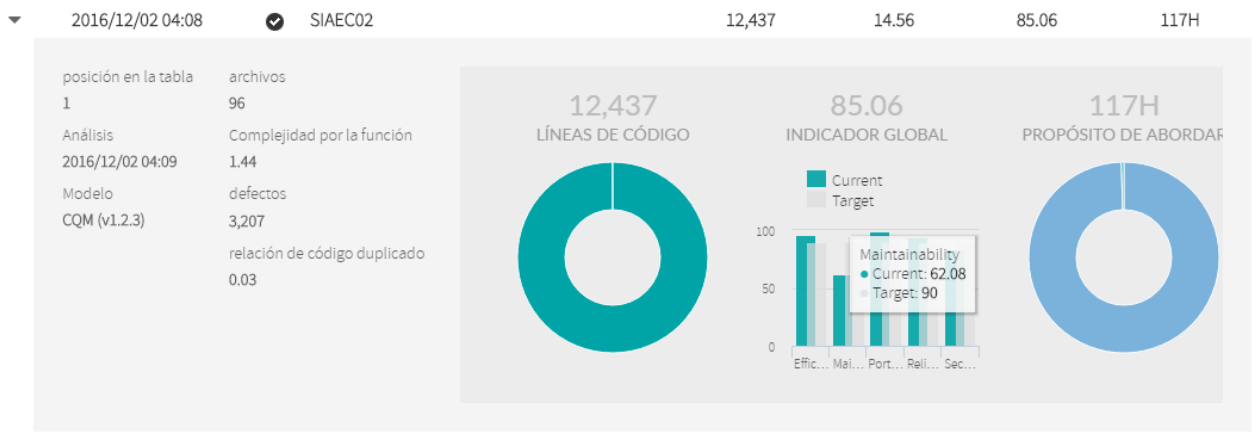


RegistrarAsignacionEquipo SIAEC02 (Segunda Versión)



RESULTADOS DE EVALUACION DE SOFTWARE POST-TEST

Los resultados obtenidos en forma de resumen son los siguientes.



Mantenibilidad:

- ✓ Objetivo Esperado: **90**
- ✓ Objetivo conseguido: **62.08**

Métricas de Aplicación:

- Relación Código Duplicado: **0.03**
- Complejidad por la función: **1.44**

✓ COMPLEJIDAD

COMPLEJIDAD POR LA FUNCIÓN

la función promedio / método de CCN. Se calcula dividiendo el CCN total por el número de funciones / métodos.

RESUMEN

Complejidad por la función	1.44
El valor máximo en un archivo	6
valor mínimo en un archivo	0
archivo de promedio por	1.37

COMPLEJIDAD

Es un normalizado (entre 0 y 100) métrica basada en la complejidad ciclomática por función, la duplicación de código y capacidad de mantenimiento de índice.

RESUMEN

Complejidad	17.89

No data to display

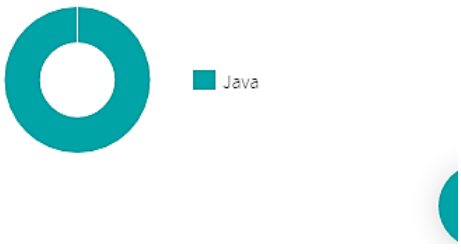
✓ DUPLICACIÓN DE CODIGO

RELACIÓN DE CÓDIGO DUPLICADO

relación de código duplicado (medido en fichas). El término "testigo" se refiere a cada uno de los elementos atómicos identificadas mediante un programa de análisis. Por ejemplo, un operador, un identificador, un número, etc .. Código duplicación, cuando es excesiva, hace que el mantenimiento más difícil, ya que cualquier cambio de la lógica de código en un clon debe hacerse en todas las instancias de la copia. Para mejorar la capacidad de mantenimiento, el código podría ser rediseñado para colocar todas las instancias de la copia en un solo lugar (por ejemplo, una nueva función, párrafo o método), y luego llamar al código refactorizada lugar.

RESUMEN

relación de código duplicado	0.03
El valor máximo en un archivo	0.28
valor mínimo en un archivo	0.05
archivo de promedio por	0.18



RATIO DE ARCHIVOS CON DUPLICACIONES

Relación de los archivos afectados por código duplicado. duplicación de código, cuando es excesiva, hace que el mantenimiento más difícil, ya que cualquier cambio de la lógica de código en un clon debe hacerse en todas las instancias de la copia. Para mejorar la capacidad de mantenimiento, el código podría ser rediseñado para colocar todas las instancias de la copia en un solo lugar (por ejemplo, una nueva función, párrafo o método), y luego llamar al código refactorizada lugar.

RESUMEN

Ratio de archivos con duplicaciones	0.1

DUPLICACIONES

número total de bloques de código duplicado. duplicación de código, cuando es excesiva, hace que el mantenimiento más difícil, ya que cualquier cambio de la lógica de código en un clon debe hacerse en todas las instancias de la copia. Para mejorar la capacidad de mantenimiento, el código podría ser rediseñado para colocar todas las instancias de la copia en un solo lugar (por ejemplo, una nueva función, párrafo o método), y luego llamar al código refactorizada lugar.

RESUMEN

duplicaciones	8

✓ RIESGO

RIESGO

indicador de riesgo, representa lo que es el riesgo de que te lleva si no hace nada para reparar sus problemas. Se calcula a partir de su indicador de GCC y el esfuerzo que tiene que pasar para alcanzar el objetivo que se configure a su aplicación. Si el indicador es mejor que su objetivo, su riesgo será 0.

RESUMEN

Riesgo	14.56

Contrastación de Indicadores y/o Métricas

COMPARACIÓN PRE-TEST Y POST-TEST

	SIAEC 01		SIAEC 02	
	Esperado	Conseguido	Esperado	Conseguido
Mantenibilidad	90	41.49	90	62.08
METRICAS				
Complejidad por Función	1.7		1.44	
Complejidad	43.35		14.89	
Relación de código Duplicado	0.06		0.03	
Radio de Archivos con Duplicación	0.15		0.1	
Duplicaciones	13		8	
Riesgo	29.07		14.56	

COMPLEJIDAD CICLOMÁTICA	EVALUACIÓN DEL RIESGO
1-10	Programa Simple, sin mucho riesgo
11-20	Más complejo, riesgo moderado
21-50	Complejo, Programa de alto riesgo
50	Programa no testeable, Muy alto riesgo

Los resultados obtenidos de la comparación de PRE-TEST y POST-TES se especifican de la siguiente manera:

La complejidad por Función se presenta como 1.7 en el PRE-TEST y presenta como 1.44 en POST-TEST indicando que se ha superado la complejidad por función en 49.82%. La complejidad tiene un indicador de 43.33 en PRE-TEST y 14.89 en POST-TEST por lo tanto se superó la Complejidad en 34.34%. La relación de código Duplicado indica 0.06 en PRE-TEST y 0.03 en POST-TEST entonces se disminuyó en 50% la Duplicación de Código, en cuanto a Duplicación de Archivos se indica un radio de 0.15 en PRE-TEST y 0.1 en POST-TEST se disminuyó el 66.6% en archivos Duplicados; y en general las Duplicaciones indican 13 en PRE-TEST y 8 en POST-TEST donde se obtiene el 60% de la disminución de duplicación en todo el Sistema Prototipo SIAEC. Referente al Riesgo indica 29.07 en PRE-TEST y 14.56 en POST-TEST donde se obtiene el 50.1% de Riesgo disminuido. Finalmente, el nivel de Mantenibilidad

conseguido en el PRE-TEST fue 41.49 indicando que el software tenía una mantenibilidad de 46.1%, luego de aplicado las técnicas de refactorización se obtuvo como nivel en POST-TEST el valor de 62.08 donde se indica que el sistema puede ser mantenible en un 68.97%, se incrementó una mejora de 22.87% para la exitosa Mantenibilidad del Sistema Prototipo SIAEC.

CAPITULO IV

DISCUSIÓN

IV. DISCUSIÓN

Esta investigación presenta como se han aplicado las técnicas de refactorización en el Sistema de Administración de Equipos de Cómputo (SIAEC), sistema que esta puesta en producción en la Gerencia Regional de Agricultura La Libertad, este estudio muestra la implementación de la lógica de reescritura.

Las ventajas en cuanto a calidad que ofrece el proceso de refactorización se ven reflejados en los resultados presentados en el Capítulo 04. Las ganancias obtenidas en las refactorizaciones de *"Renombrar clases, métodos y variables", "Extraer Método", "Extraer Interfaz", "Añadir Abstract Factory", "Renombrar paquetes"* son claramente notables, a diferencia de la investigación de (Angel, 2013) donde aplica las técnicas de refactorización haciéndolas más complejas. Respecto a la Complejidad, en la versión original de SIAEC, que fue considerada como SIAEC01 se obtuvo los siguientes resultados: Complejidad por Función se calcula dividiendo el CCN total por el número de funciones / métodos, se redujo 0.26 respectivamente de la complejidad por función; Complejidad es un normalizado (entre 0 y 100), esta métrica está basada en la complejidad ciclomática por función, la duplicación de código y capacidad de mantenimiento, se pudo reducir 28.46 de la complejidad; Relación de código duplicado indica el radio de código duplicado, se pudo reducir 0.03 el radio de duplicación redujo a la mitad; y por ultimo tenemos el indicador de Riesgo, representa lo que es el riesgo que te lleva si no hace nada para reparar sus problemas. Se calcula a partir de su indicador de GCC y el esfuerzo que tiene que pasar para alcanzar el objetivo que se configure a su aplicación, si el indicador es mejor que su objetivo, su riesgo será 0; nuestro riesgo era de 29.07 pero redujo 14.51 y se obtuvo 14.56 respectivamente, está dentro de lo moderado.

La investigación de (Marticonera Sánchez, 2013) menciona también sobre la Calidad de software donde se define la importancia de la mantenibilidad, así también como la eficiencia entre otros; por lo tanto, si se habla de refactorización el impacto tendría que ser el mismo ya que esto se enfoca en la mejora de Calidad de Software además de establecer una estructura más precisa para el debido entendimiento del Sistema.

Finalmente, el objetivo de Garantizar mayor mantenibilidad del sistema prototipo SIAEC se consigue enteramente con la aplicación de técnicas de refactorización, ya que, tras su aplicación, el programador sabrá exactamente como empezar a codificar e incluso si se encuentra con otros sistemas con el mismo problema sabrá por dónde empezar.

CAPITULO V

CONCLUSIÓN

V. CONCLUSIÓN

- ✓ Se concluye que la aplicación de Técnicas de Refactorización si logró que el sistema prototipo SIAEC puede ser mantenible para cuando se quiera añadir más requerimientos, o agregar más módulos al Sistema ya que no se invertirá demasiado tiempo para poder entender el código fuente.
- ✓ La implementación de técnicas de refactorización disminuyó la densidad de defectos en un 50.1% del sistema prototipo SIAEC.
- ✓ La implementación de técnicas de refactorización redujo determinadamente en un 49.82% la Complejidad del sistema prototipo SIAEC.
- ✓ La implementación de técnicas de refactorización incrementó la Comprensibilidad en un 68.97% del sistema prototipo SIAEC.

CAPITULO VI

RECOMENDACIONES

VI. RECOMENDACIONES

Luego de haber realizado esta investigación, se llegó a las siguientes recomendaciones:

- Se recomienda para un trabajo futuro ampliar el catálogo de refactorizaciones para cubrir otros casos que no se hayan considerado.
- Se recomienda explorar nuevas herramientas de Análisis de software como SONARQUBE, CODEPROANALITYX, etc. para comprobar si hay similitudes en los resultados obtenidos.
- En la problemática se mencionó acerca de un problema que también ocasiona la mala calidad de software y es el trabajo en equipo, si mejoramos también el trabajo en equipo el software tendría la calidad correspondiente, se recomienda usar metodologías agiles como por ejemplo SCRUM, esta metodología abarca mucho el trabajo en equipo mediante Sprint donde se designan las tareas correspondientes ya sea para definir estándares que se usaran durante el desarrollo de software, para así evitar problemas.

CAPITULO VII

REFERENCIAS

VII. REFERENCIAS

Angel, Cuenca Ortega. 2013. *Tecnicas de Refactorizacion para Maude.* Valencia : s.n., 2013. pág. 131.

Beck, Kent. 2005. *Programación eXtrema explicada: Aceptando el cambio.* s.l. : 2° Edicion, 2005.

Collazos Serrano, Víctor . 2009. *"Refactorización arquitectónica de software orientada a la detección de patrones de diseño J2EE.* Universidad Nacional de Colombia. Bogotá : s.n., 2009.

Cuenca Ortega, Angel. 2013. *Tecnicas de Refactorizacion para Maude.* Departamento de Sistemas Informaticos y Computacion, Universidad Politecnica de Valencia. Valencia : s.n., 2013. pág. 131 pags.

Fowler, Martin. 1999. *Refactoring, Mejorar el diseño de código existente.* s.l. : Addison-Wesley, 1999. pág. 320 págs.

Grady, Booch. 2001. *Analisis y Diseño Orientada a Objetos con Aplicaciones.* Mexico : S.A. Alhambra Mexicana, 2001. pág. 672.

Marticonera Sánchez, Raúl . 2013. *Refactorización sobre Programación genérica en Lenguajes Orientado a Objetos.* Universidad de Valladolid. Valladolid : s.n., 2013.

Melo, W y Abreu, F Brito. 1996. *Evaluating the impact of ObjectOriented.* Berlin : Proceedings of 3rd International, 1996. Sin especificar .

Rodriguez Candela, Alberto Sols. 2000. *Fiabilidad, mantenibilidad, efectividad: un enfoque sistémico.* Madrid : JPM Graphic, 2000. pág. 361. Vol. 12.

S. Pressman, Roger. 2002. *Ingenieria de Software un enfoque Practico.* Madrid : Concepcion Fernandez Madrid, 2002. pág. 642 pag.

Sommerville, Ian. 2011. *Ingenieria de Software.* [ed.] Luis M. Cruz Castillo. [trad.] Victor Campos Olguin. Novena. Mexico : Pearson Educacion de Mexico, 2011. pág. 792.

W. Boehm, Barry. 1979. *Software engineering; R & D Trends and defense needs.* Cambridge : P. (ed.). Cambridge, 1979.

ANEXOS

Anexo 1. Elección de Metodología de Estudio.

CRITERIO	METODOLOGIA		
	RUP	XP	MANTEMA
Información	6	5	5
Documentación	5	4	6
Tiempo y Costo	3	5	6
Aseguramiento de la Calidad	6	4	6
Total	**20**	**18**	**23**

Según los criterios seleccionados y la validación aplicada a dos expertos en desarrollo de software, la metodología a utilizar es MANTEMA ya que obtuvo un puntaje superior a las metodologías también expuestas, por lo tanto, la investigación será trabajada en base a la Metodología MANTEMA.

Anexo 2. Validación de Realidad Problemática

CRITERIO	Sistema Prototipo SIAEC
Calidad de Software	3
Fiabilidad del Software	3
Complejidad del Software	4
Comprensibilidad del Software	2
Facilidad de Evolución de Software	2
TOTAL	**14**

Según los criterios seleccionados y la Validación aplicada a un Experto, quien cual ha trabajado durante 15 años en el desarrollo de Software, obtuvo como resultado que la Calidad, Fiabilidad de software no son muy favorables ya que es considerado como (Algo) en cuanto a la Valoración; también se tiene la comprensibilidad y Facilidad de Evolución de Software se encuentra en la Valorización (Poco), nos indica que el Software tiene problemas en la Comprensibilidad y Facilidad de Evolución; y por último se indica que la Complejidad del Sistema es un problema muy sobresaliente ya que indica cómo (Bastante); en otras palabras, la complejidad de software es mayor de lo que se esperaba.

Anexo 3. Plantilla de Validación de metodología de desarrollo de software

FACULTAD DE INGENIERÍA
ESCUELA ACADÉMICO PROFESIONAL DE INGENIERÍA DE SISTEMAS

ENCUESTA PARA LA VALIDACIÓN DE LA REALIDAD PROBLEMÁTICA

Tema: "Aplicación de Técnicas de Refactorización para mejorar la Mantenibilidad del Sistema Prototipo SIAEC en su proceso de Evolución".
Finalidad: Validar que los problemas de la realidad problemática sean ciertos.

La información que usted brinde es muy importante para la investigación en curso. Sea muy honesto al momento de brindar la información.

Apellidos y Nombre: .....

Profesión: .....

Cargo: .....

Experiencia (años): .....

Cuadro de Valoración: Para la validación de la realidad problemática se utilizará la siguiente valoración.

Valoración	Escala
Nada	1
Poco	2
Algo	3
Bastante	4
Mucho	5

Calificación de los problemas descritos en la realidad problemática en base a los criterios de valoración.

Criterio	Software
Calidad de Software	
Fiabilidad del Software	
Complejidad de Software	
Comprensibilidad de Software	
Facilidad de Evolución de Software	

Anexo 4. Validación de metodología de desarrollo de software

FACULTAD DE INGENIERÍA
ESCUELA ACADÉMICO PROFESIONAL DE INGENIERÍA DE SISTEMAS

ENCUESTA PARA LA VALIDACIÓN DE LA REALIDAD PROBLEMÁTICA

Tema: "Aplicación de Técnicas de Refactorización para mejorar la Mantenibilidad del Sistema Prototipo SIAEC en su proceso de Evolución".
Finalidad: Validar que los problemas de la realidad problemática sean ciertos.

La información que usted brinde es muy importante para la investigación en curso. Sea muy honesto al momento de brindar la información.

Apellidos y Nombre: Cárdenas Escalante, Lain.

Profesión: Ing. Computación y Sistemas.

Cargo: Docente UCV

Experiencia (años): 15

Cuadro de Valoración: Para la validación de la realidad problemática se utilizará la siguiente valoración.

Valoración	Escala
Nada	1
Poco	2
Algo	3
Bastante	4
Mucho	5

Calificación de los problemas descritos en la realidad problemática en base a los criterios de valoración.

Criterio	Software
Calidad de Software	3
Fiabilidad del Software	3
Complejidad de Software	4
Comprensibilidad de Software	2
Facilidad de Evolución de Software	2

Anexo 5. Plantilla de validación de metodología de desarrollo de software

FACULTAD DE INGENIERÍA
ESCUELA ACADÉMICO PROFESIONAL DE INGENIERÍA DE SISTEMAS

ENCUESTA PARA LA ELECCIÓN DE LA METODOLOGÍA DE DESARROLLO

Tema: "Aplicación de Técnicas de Refactorización para mejorar la Mantenibilidad del Sistema Prototipo SIAEC en su proceso de Evolución",
Finalidad: Obtener una metodología la cual me ayudara a realizar la investigación.

La información que usted brinde es muy importante para la investigación en curso. Sea muy honesto al momento de brindar la información.

Apellidos y Nombre: .....

Profesión: .....

Años de Experiencia: .....

Criterios de Evaluación de la Metodología: La elección de la metodología se realizará bajo los siguientes criterios de evaluación.

- **Información:** Se refiere a las guías o los ejemplos que cuenta cada metodología para su implementación.
- **Documentación:** Se refiere a que todos los procesos cuentan con una documentación clara y precisa en el proceso de desarrollo de software.
- **Tiempo y Costo:** Se refiere al tiempo y al costo que se invierten al utilizar la metodología.
- **Aseguramiento de la Calidad:** Se refiere a la capacidad que cuenta cada metodología para poder garantizar la calidad del producto.

Cuadro de Valoración: Para la elección de la metodología se utilizará la siguiente valoración.

Valoración	Escala
Deficiente	1
Regular	2
Bueno	3

Calificación de las Metodología en base a los criterios de valoración

CRITERIO	METODOLOGÍAS		
	RUP	XP	MANTEMA
Información			
Documentación			
Tiempo y Costo			
Aseguramiento de la calidad			

Anexo 6. Validación de metodología de desarrollo de software

FACULTAD DE INGENIERÍA
ESCUELA ACADÉMICO PROFESIONAL DE INGENIERÍA DE SISTEMAS

ENCUESTA PARA LA ELECCIÓN DE LA METODOLOGÍA DE DESARROLLO

Tema: "Aplicación de Técnicas de Refactorización para mejorar la Mantenibilidad del Sistema Prototipo SIAEC en su proceso de Evolución",
Finalidad: Obtener una metodología la cual me ayudara a realizar la investigación.

La información que usted brinde es muy importante para la investigación en curso. Sea muy honesto al momento de brindar la información.

Apellidos y Nombre: Cárdenas Escalante, Lain.

Profesión: Ing. Computación y Sistemas.

Años de Experiencia: 15.

Criterios de Evaluación de la Metodología: La elección de la metodología se realizará bajo los siguientes criterios de evaluación.

- **Información:** Se refiere a las guías o los ejemplos que cuenta cada metodología para su implementación.
- **Documentación:** Se refiere a que todos los procesos cuentan con una documentación clara y precisa en el proceso de desarrollo de software.
- **Tiempo y Costo:** Se refiere al tiempo y al costo que se invierten al utilizar la metodología.
- **Aseguramiento de la Calidad:** Se refiere a la capacidad que cuenta cada metodología para poder garantizar la calidad del producto.

Cuadro de Valoración: Para la elección de la metodología se utilizará la siguiente valoración.

Valoración	Escala
Deficiente	1
Regular	2
Bueno	3

Calificación de las Metodología en base a los criterios de valoración

CRITERIO	METODOLOGÍAS		
	RUP	XP	MANTEMA
Información	3	2	2
Documentación	3	2	3
Tiempo y Costo	1	2	3
Aseguramiento de la calidad	3	2	3

Anexo 7. Validación de metodología de desarrollo de software

UCV
UNIVERSIDAD CÉSAR VALLEJO

FACULTAD DE INGENIERÍA
ESCUELA ACADÉMICO PROFESIONAL DE INGENIERÍA DE SISTEMAS

ENCUESTA PARA LA ELECCIÓN DE LA METODOLOGÍA DE DESARROLLO

Tema: "Aplicación de Técnicas de Refactorización para mejorar la Mantenibilidad del Sistema Prototipo SIAEC en su proceso de Evolución",
Finalidad: Obtener una metodología la cual me ayudara a realizar la investigación.

La información que usted brinde es muy importante para la investigación en curso. Sea muy honesto al momento de brindar la información.

Apellidos y Nombre: Jaracito Gómez Yosip

Profesión: Ingeniero de Sistemas

Años de Experiencia: 14 años

Criterios de Evaluación de la Metodología: La elección de la metodología se realizará bajo los siguientes criterios de evaluación.

- **Información:** Se refiere a las guías o los ejemplos que cuenta cada metodología para su implementación.
- **Documentación:** Se refiere a que todos los procesos cuentan con una documentación clara y precisa en el proceso de desarrollo de software.
- **Tiempo y Costo:** Se refiere al tiempo y al costo que se invierten al utilizar la metodología.
- **Aseguramiento de la Calidad:** Se refiere a la capacidad que cuenta cada metodología para poder garantizar la calidad del producto.

Cuadro de Valoración: Para la elección de la metodología se utilizará la siguiente valoración.

Valoración	Escala
Deficiente	1
Regular	2
Bueno	3

Calificación de las Metodología en base a los criterios de valoración

CRITERIO	METODOLOGÍAS		
	RUP	XP	MANTEMA
Información	3	3	3
Documentación	2	2	3
Tiempo y Costo	2	3	3
Aseguramiento de la calidad	3	2	3

Anexo 8. Validación de Métricas de Mantenibilidad

UCV
UNIVERSIDAD CÉSAR VALLEJO

PLANTILLAS PARA LA EVALUACIÓN DE INSTRUMENTOS DE RECOLECCIÓN DE DATOS

1. IDENTIFICACION DEL EXPERTO

NOMBRE DEL EXPERTO: Lain Cárdenas Escalante.
DNI: 18133704 PROFESION: Ing. Computación y Sistemas
LUGAR DE TRABAJO: UCV
CARGO QUE DESEMPEÑA: Docente.
DIRECCION:
TELEFONO FIJO: MOVIL:
DIRECCION ELECTRONICA: laincardenas@gmail.com.
FECHA DE EVALUACIÓN: 12/07/2016

FIRMA DEL EXPERTO

2. PLANILLA DE VALIDACION DEL INSTRUMENTO

CRITERIOS	APRECIACION CUALITATIVA			
	EXCELENTE (4)	BUENO (3)	REGULAR (2)	DEFICIENTE (1)
Presentación del instrumento	✓			
Claridad en la redacción de los ítems	✓			
Pertinencia de las variables con los indicadores	✓			
Relevancia del contenido	✓			
Factibilidad de la aplicación	✓			

APRECIACION CUALITATIVA:

OBSERVACIONES:

3. JUICIO DE EXPERTOS

En líneas generales, considera Ud. que los indicadores de las variables están inmersos en su contexto teórico de forma:

SUFICIENTE	MEDIANAMENTE SUFICIENTE	INSUFICIENTE
✓		

OBSERVACION:

Considera que los reactivos del cuestionario miden los indicadores seleccionados para la variable de manera:

SUFICIENTE	MEDIANAMENTE SUFICIENTE	INSUFICIENTE
✓		

OBSERVACION:

El instrumento diseñado mide la variable de manera:

SUFICIENTE	MEDIANAMENTE SUFICIENTE	INSUFICIENTE
✓		

OBSERVACION:

El instrumento diseñado es:

4. PVALIDACION DEL INSTRUMENTO

ITEMS	ESCALA				OBSERVACIONES
	DEJAR	MODIFCAR	ELIMINAR	INCLUIR	
01	✓				
02	✓				
03	✓				
04	✓				
05	✓				

DESEARIA INCLUIR	COMO LO MODIFICARIA