Universidad de las Ciencias Informáticas

Trabajo de diploma para optar por el título de Ingeniero en Ciencias Informáticas

Título:

Componente para la comparación de plantillas de minucias de huellas dactilares.

Autor:

Luis Yasiel Silva García

Tutores:

Dr.C Ramón Santana Fernández

MSc. Adrian A. Machado Cento

La Habana, junio de 2017

Agradecimientos

A Todas las personas que de una forma u otra han influido en el resultado de este trabajo de diploma. A mi familia por siempre estar, por nunca dudar de mí y confiar en que podría hoy estar aquí y decirle al mundo soy ingeniero. A mi mamá por no soltarme la mano cuando la necesite, por ser esa madre cariñosa que todos deseamos tener por su espíritu y voluntad incansable. A mi papá, gracias por tu educación, por tu enseñanza, por tu sabiduría, por todo. A mi hermosa tia Francis por todo el amor, todo el cariño, por cada consejo,por toda la ayuda, gracias por ser como otra madre para mi.A mis hermanos Mariela, Pedrito, Yuslaidy y mi querido primo hermano Lachi simplemente por ser ellos mismos, por estar y quererme tal y como soy, los quiero.

A los profesores y profesionales que me han brindado su ayuda para enriquecer mis conocimientos en el transcurso de mi carrera y por ello estar hoy aquí, especialmente mis tutores por ser grandes profesionales y mejores personas.

A todos… MUCHAS GRACIAS!!!

Dedicatoria

A mis queridos padres con todo mi amor y esfuerzo por ser mi pilar impulsor en la vida.

A mi país, a la universidad, a la informática, la ciencia y el conocimiento.

Declaración de Autoría

Declaro por este medio que yo Luis Yasiel Silva García, con carné de identidad 93022006665 soy el autor principal del trabajo titulado “Componente para la comparación de plantillas de minucias de huellas dactilares” y autorizo a la Universidad de las Ciencias Informáticas a hacer uso de la misma en su beneficio, así como los derechos patrimoniales con carácter exclusivo.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

Ramón Santana Fernández

_____

Firma del Tutor

Adrian A. Machado Cento

_____

Firma del Co-Tutor

Luis Yasiel Silva García

_____

Firma del Autor

Datos del contacto

Tutor: Dr.C Ramón Santana Fernández.

Ingeniero en Ciencias Informáticas. Especialista asociado al Centro de Identificación y Seguridad Digital (CISED), graduado en el curso 2010-2011. Tiene 7 años de experiencia en el campo dedicado al reconocimiento de personas mediante huellas dactilares. Ha participado en varios eventos científicos relacionados con la biometría en la universidad, a nivel nacional e internacional obteniendo buenos resultados. Actualmente es autor de varias publicaciones asociadas a la reconstrucción de huellas dactilares y a la protección de plantillas de minucias de huellas dactilares.

Correo electrónico: rsfernandez@uci.cu

Tutor: Ms C. Adrián Alberto Machado Cento.

Graduado de Ingeniería en Ciencias Informáticas en la UCI en el año 2006, Máster en Informática Aplicada en la UCI en el año 2008, Jefe del departamento de Desarrollo del Aplicaciones del Centro de Identificación y Seguridad Digital. Ha investigado en el área de sistemas de identificación y biometría.

Correo electrónico: amachado@uci.cu

Resumen

Los identificadores biométricos a diferencia de los identificadores tradicionales no pueden ser reemplazados ni cambiados; por lo que la huella dactilar como identificador biométrico es muy utilizado a nivel mundial en el reconocimiento de personas, ya que el usuario empleará una de sus huellas dactilares para ser reconocido. Estos sistemas biométricos son fáciles de mantener y en general no requiere la intervención de más agentes que el propio usuario para hacerlo funcionar.

La presente investigación tiene como objetivo general desarrollar un componente para la comparación de plantillas de minucias de huellas dactilares en texto claro, utilizando estructuras complejas con la meta de mejorar las tasas de falsos rechazados y falso aceptados arrojadas por el componente que utiliza actualmente el Centro de Investigación y Seguridad Digital.

Como principales resultados se obtiene un componente que realiza la comparación de plantillas de minucias de huellas dactilares, mejorando los resultados de falsos rechazados y falsos aceptados. Las pruebas realizadas demostraron una mejoría en las tasas del componente realizado en comparación con las dadas por las pruebas realizadas al componente utilizado actualmente en el centro.

Palabras Claves: sourceAFIS, plantillas, huellas dactilares, minucias, sistemas biométricos.

Índice General

Índice de Tablas

Índice de Figuras

Introducción

En la actualidad los sistemas de reconocimiento biométrico han cobrado gran relevancia en entornos que requieren la identificación de usuarios o accesos restringidos. En comparación con los métodos clásicos comúnmente utilizados, como llaves o claves, los rasgos biométricos no pueden, en general, ser robados o copiados fácilmente. El usuario empleará un rasgo fisiológico o morfológico para ser reconocido. Esta clase de sistemas suele ser fácil de mantener y en general no requiere la intervención de más agentes que el propio usuario para funcionar. La biometría hace referencia a la aplicación automatizada de técnicas biométricas para la certificación, autentificación e identificación de personas en lugares donde se requiere seguridad. Las técnicas biométricas se utilizan para medir características corporales o de comportamiento de las personas con el objeto de establecer una identidad (1).

En la biometría se distinguen dos grupos de rasgos biométricos, los fisiológicos o morfológicos y los conductuales. Los fisiológicos o morfológicos son aquellos que se rigen por características físicas inalterables que se encuentran presentes en la mayoría de los seres humanos tales como, la huella dactilar, geometría de la mano, características del iris, patrones vasculares de la retina, mano, entre otros. Y los conductuales son aquellos que se rigen por las características de la conducta del ser humano tales como, las pulsaciones del teclado, discurso, dinámica de la firma, entre otros (2).

La huella dactilar de un individuo ha sido un patrón aceptable para determinar su identidad de forma inequívoca, ya que existe una probabilidad muy baja de que dos dedos posean huellas similares, esta posibilidad incluye a gemelos y a los dedos de una misma persona. Por lo que la presente investigación se centrará en unos de los rasgos fisiológicos más utilizados “Las huellas dactilares” (3).

Estas son la reproducción de la epidermis de la parte posterior de los dedos de la mano conformada por un conjunto de líneas que se denominan crestas (líneas oscuras) y valles (líneas claras). Este conjunto de líneas que forman las huellas dactilares pueden asemejarse a patrones o texturas que se pueden analizar de diferentes maneras dependiendo del grado de detalle, como se muestra en la figura 1 (4).

Figura 1: Huella Dactilar.

El reconocimiento de la huella dactilar, a pesar de haber recibido un importantísimo impulso en las últimas décadas gracias al empleo de nuevas tecnologías, escáneres, procesado digital de imágenes, algoritmos de reconocimiento y comparación, se han venido usando desde la antigüedad. Una de las primeras muestras utilizadas para la comparación de personas data de 1858, donde Sir William Herschel, trabajador del servicio civil de la India, imprimió la huella de la mano al reverso del contrato de cada trabajador, para distinguir los empleados de otros que intentaran suplantar a los trabajadores el día de pago (5).

En 1918 Edmond Locard escribió que si 12 puntos principales de la huella de los 40 propuestos por el antropólogo británico Sir Francis Galton coinciden en una comparación de dos huellas dactilares era suficiente para una identificación positiva. Con la llegada de las computadoras y un subconjunto de los puntos Galton llamadas minucias (rasgos específicos de las huellas dactilares), el Buró Federal de Investigaciones (FBI) en 1975 consolidó el uso de escáneres y tecnología para la extracción de minucias, que llevó al desarrollo de un lector que recolectaba estas minucias y posteriormente las almacenaba (5).

Durante las siguientes décadas, una de las primeras empresas en trabajar en el tema fue el Instituto Nacional de Estándares y Tecnología (NIST) enfocándose en el desarrollo de métodos automáticos para digitalizar las huellas dactilares en tinta y los efectos que tendría la compresión de imagen respecto a la calidad, la clasificación, la extracción de minucias y la comparación. El trabajo del NIST condujo el desarrollo del algoritmo denominado M40, el primer algoritmo operacional utilizado en el FBI para estrechar la búsqueda de humanos. En la actualidad, el uso de este rasgo biométrico no sólo se restringe al ámbito policial, o de seguridad, sino que se ha extendido y forma parte de la vida cotidiana hasta el punto de que la mayoría de los portátiles, y algunas memorias USB llevan lectores y algoritmos de comparación dactilar integrados (6).

Históricamente, el fraude de identidad es un problema ético que atenta contra la seguridad de los sistemas informáticos. Existe una serie de fuentes de ataques que a lo largo de los años han intentado burlar la seguridad de acceso de los sistemas biométricos.

En los últimos tiempos se han realizados muchas investigaciones acerca de las vulnerabilidades que tienen los sistemas basado en el reconocimiento dactilar frente a posibles ataques. En los cuales se encuentran los ataques dirigidos al sensor (llevados a cabo utilizando rasgos biométricos sintéticos como dedos de goma, moldes de plastilina, silicona, gelatina), y ataques indirectos dirigidos a una parte interna del sistema(llevado a cabo contra módulos internos del sistema, robo o modificación de la base de datos o plantilla) (7). Por lo que las disimiles entidades que trabajan acerca del tema, han centrado su atención en buscar nuevos métodos y soluciones para hacer el proceso de extracción y comparación de características de las huellas dactilares en sistemas biométricos más robustos y seguros.

En la actualidad en el Centro de Identificación y Seguridad Digital se utiliza un motor de reconocimiento biométrico denominado SourceAfis para realizar la autenticación de personas mediante huellas dactilares. El algoritmo de comparación utilizado para conocer si una huella dactilar pertenece a una persona no se encuentra descrito en la bibliografía lo que dificulta la comprensión del mismo, el cual se observó en el código del componente que está formado por:

1. Método geométrico utilizado para la alineación de los datos.
2. Método utilizado para la consolidación de los datos.
3. Proceso de comparación.

Lo expuesto anteriormente dificulta la realización de cambios para mejorar los resultados obtenidos del proceso de comparación. Las tasas de falso aceptado[1](FAR) y falso rechazo[2](FRR) expuestos por el autor del motor de reconocimiento son 0.01 FAR y 10.9 FRR (8) las cuales no coinciden con las tasas obtenidas durante el proceso de pruebas realizado en el centro, siendo estas de 0.07866 FAR y 14.11765 FRR (9).

Este componente está desarrollado utilizando tecnologías privativas, lo cual va en contra de las políticas

[1] falsos aceptados: Métrica que consiste en que el conjunto de características de dos dedos diferentes es aceptado como de la misma persona cuando debería haber sido rechazado.

[2] falsos rechazos: Métrica que consiste en que el conjunto de características del mismo dedo es rechazado cuando debería haber sido aceptado.

de software libre de la universidad:

1. Afecta la comercialización del producto.
2. Disminuye el área de mercado.

En el centro se implementó el algoritmo Minutia Cylinder Code como parte de una tesis de grado (10). Este algoritmo esta implementado bajo el estándar de tarjetas inteligentes y al realizar los cambios pertinentes para que pueda ser utilizado con propósito general disminuye el rendimiento biométrico lo cual dificulta la autenticación de personas.

Teniendo en cuenta la **situación problemática** descrita anteriormente se enuncia el siguiente **problema de la investigación**: ¿Cómo mejorar el rendimiento biométrico[3] del proceso de comparación de plantillas de minucias de huellas dactilares en el Centro de Identificación y Seguridad Digital?

Teniendo como **Objetivo general:** Desarrollar un componente para la comparación de plantillas de minucias de huellas dactilares, que utilice estructuras complejas como estructura base de la comparación, para mejorar los resultados de este proceso en el Centro de Identificación y Seguridad Digital.

Se determina como **Objeto de Estudio** el proceso de comparación de plantillas de minucias de huellas dactilares y el **campo de acción** se ha delimitado en el proceso de comparación de plantillas de minucias de huellas dactilares utilizando estructuras complejas.

Para dar solución al objetivo general se plantearán las siguientes **Objetivos específicos:**

1. Analizar los referentes teórico-metodológicos asociados al dominio de la investigación para mejorar la comprensión del proceso de comparación de plantillas de minucias de huellas dactilares.
2. Realizar análisis y diseño del componente de comparación de plantillas de minucias de huellas dactilares para guiar el proceso de desarrollo.
3. Implementar el componente propuesto para la comparación de plantillas de minucias de huellas dactilares.

[3] Rendimiento biométrico: se define generalmente como la medida para decidir cuan efectivo es el sistema, mientras las tasas de falsos aceptado y rechazados sean más bajas, el sistema será más exacto.

4. Validar el funcionamiento del componente desarrollado mediante la ejecución de pruebas de rendimiento biométrico.

Hipótesis de la investigación.

El desarrollo de un algoritmo de comparación de plantillas de minucias de huellas dactilares, basado en estructuras geométricas, mejorará las tasas de falsos aceptados y falsos rechazos.

Obteniendo como **Posibles resultados:**

1. Un componente para la comparación de plantillas de minucias de huellas dactilares.

Métodos Teóricos.

- **Los Métodos Científicos** utilizados son:
  - **Analítico Sintético:** Se realizó el análisis de múltiples documentos acerca de las huellas dactilares, permitiendo el descubrimiento y entendimiento de sus principales características para poder establecer una relación entre ellas.
  - **Análisis Histórico–Lógico:** Se puso en práctica en el uso cronológico del estudio de la evolución y desarrollo que han tenido los sistemas biométricos de huellas dactilares a través de la historia.
  - **Hipotético–Deductivo:** Se utiliza este método para elaborar una hipótesis que sustente todo el proceso investigativo.
  - **Observación:** Se puso en práctica este método para observar el código del componente de comparación de plantillas de minucias de huellas dactilares que utiliza el centro, para poder comprender la conformación del algoritmo de comparación.
- **Los Métodos Empíricos** utilizados son:
  - **Experimento:** Se emplea este método con el propósito de comprobar la funcionalidad del proceso de comparación de minucias de huellas dactilares según son implementadas.

El presente trabajo de diploma consta de tres capítulos estructurados de la siguiente forma:

- **Capítulo I: Fundamentación teórica:** se abordan los principales elementos teóricos vinculados a la investigación, se realiza el estudio del estado del arte de disimiles algoritmos de comparación de plantillas de minucias de huellas dactilares y el análisis de las metodologías de desarrollo, tecnologías y herramientas a utilizar para el desarrollo del componente.

- **Capítulo II: Características del componente:** se describe la solución propuesta y los principales aspectos relacionados con su diseño. Se describe el componente a través de las historias de usuario, la arquitectura y otros artefactos generados por la metodología seleccionada.
- **Capítulo III: Implementación y prueba:** se describen los artefactos relacionados con la implementación y las pruebas realizadas al componente con el objetivo de validar su correcto funcionamiento y su correspondencia con los requerimientos especificados.

Capítulo 1: Fundamentación Teórica.

Introducción.

En el presente capítulo se analizan los referentes teóricos metodológicos asociado al proceso de comparación de plantillas de minucias de huellas dactilares. Se realiza un análisis de los métodos de comparación descritos en la bibliografía y se selecciona el método a implementar. Se analizan las principales metodologías, tecnologías y herramientas para seleccionar el ambiente de desarrollo a utilizar en el desarrollo del componente.

1.1 Conceptos asociados al dominio del problema

1.1.1 Huellas dactilares.

Las huellas dactilares son patrones constituidos por las crestas papilares de los dedos de las manos que aparecen visibles en la epidermis (Piel). Se forman en el período fetal a partir del sexto mes de vida y se mantienen invariantes durante toda la vida de cada persona (11).

Propiedades.

- **Perennidad.**

Las crestas papilares se forman a partir del sexto mes de vida, dentro de la intrauterina de la madre y desde ese momento el dibujo papilar es el mismo para toda la vida del ser humano (11).

- **Inmutabilidad.**

Es la capacidad que tiene el tejido de reconstruirse y reaparecer el mismo diseño que antes describían los surcos y crestas papilares (11).

- **Diversidad.**

Las huellas dactilares son diversiformes y no pueden encontrarse dos semejantes en una serie de sesenta y cuatro mil millones (11).

1.1.2 Características estructurales.

Las huellas dactilares se encuentran conformadas por:

- **Valles o Surcos y Crestas.**

Son las rugosidades que forman salientes y depresiones. Las depresiones son denominadas surcos inter-papilares o valles y las salientes crestas papilares. Una cresta está definida como un segmento de una curva y un valle es la región entre dos crestas adyacentes, como se muestra en la figura 2, (12).

Figura 2: Valles y Crestas en una huella dactilar.

1.1.3 Minucia.

Es el punto de interés de una huella dactilar. Inicialmente se realiza la selección de un punto central $A(\mathbf{x}, \mathbf{y})$ dentro del conjunto de minucias representadas como: minucia $= \{\mathbf{x}, \mathbf{y}, \boldsymbol{\theta}, \mathbf{t}\}$ donde $\mathbf{x}$ e $\mathbf{y}$ son la posición de la imagen de la huella dactilar, $\boldsymbol{\theta}$ es el ángulo de orientación de la minucia y $\mathbf{t}$ el tipo de minucia.

Las minucias se clasifican en diferentes tipos como se muestra en la figura 3. Los tipos de mayor valor identificativos son las terminaciones y bifurcaciones, el resto es considerado por diferentes expertos como la combinación de estas (13).

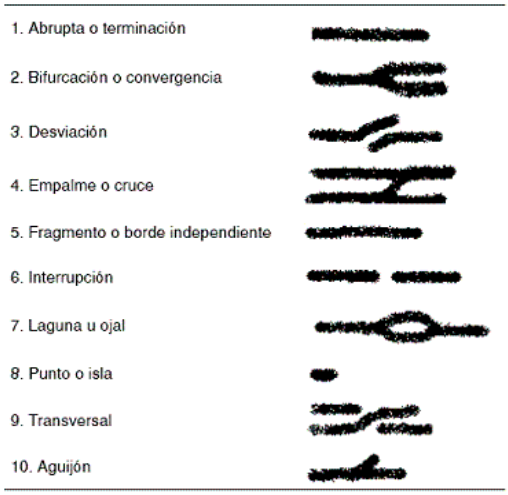

Figura 3: Tipos de minucias dactilares.

- **Núcleo.**

Es el punto de máxima curvatura de la cresta, como se muestra en la figura 4.

- **Delta.**

Es el centro de un conjunto de crestas inferiores al núcleo que dibuja varios triángulos concéntricos como se muestra en la figura 4, (14).

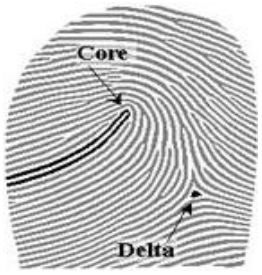

Figura 4: Núcleo o Core y Delta.

1.1.4 Clasificación de las huellas dactilares.

Las huellas dactilares son únicas en cada individuo y estas se pueden clasificar en 5 clases: Arco, Arco tendido, Lazo izquierdo, Lazo derecho y Espiral, como se muestra en la figura 5.

- **Arco.**

Los patrones de arco tienen líneas que empiezan en un lado de la huella dactilar, van hacia el centro curvándose hacia arriba y salen del otro lado de la huella dactilar, formando lo que se asemeja a una fila de arcos apilados, no cuentan con un punto delta.

- **Arco Tendido.**

Semeja a un arco. Es un tipo de línea que rápidamente nace y se pierde en un paso de ángulo.

- **Lazo derecho.**

Se caracterizan porque las crestas que forman su núcleo nacen en el costado izquierdo de la huella dactilar y hacen su recorrido hacia la derecha, para luego dar vuelta sobre sí mismas y regresar al mismo punto de partida. Estas tienen un punto delta que se encuentra ubicado al lado derecho de la huella dactilar.

- **Lazo izquierdo.**

Al igual que el lazo derecho, estas tienen un punto delta, pero éste se ubica del lado izquierdo de la huella dactilar. Las crestas papilares que forman el núcleo nacen a la derecha y su recorrido es a la izquierda para dar vuelta sobre sí mismas y regresar al mismo punto de partida.

- **Espiral (Anillo de cresta).**

Los patrones en forma de anillos tienen muchos círculos que salen de un punto de la huella dactilar, las mismas cuentan con dos puntos deltas (12).

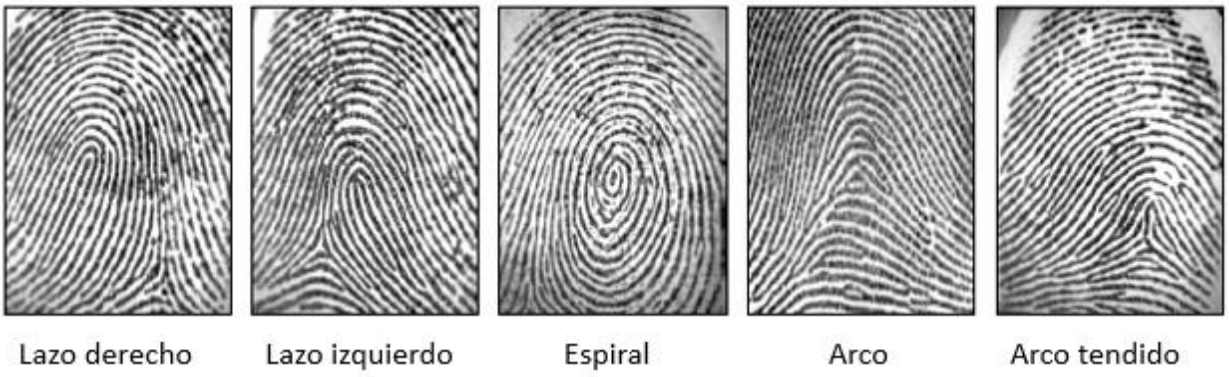

Figura 5: Clasificaciones de las huellas dactilares.

1.1.5 Tripletas.

Es un triángulo formado por la unión de tres puntos de minucias. Una tripleta válida tiene que cumplir una serie de restricciones relacionadas con los lados del triángulo, los ángulos interiores y las orientaciones referentes a cada punto de minucia (15).

1.2 Análisis del estado del arte.

El proceso de comparación es una de las fases críticas en un sistema de autenticación de personas mediante huellas dactilares. El proceso de comparación puede ser muy complejo ya que dos tomas del mismo identificador biométrico sufren desplazamientos, rotaciones, distorsiones y deformaciones no lineales. Un algoritmo de comparación de plantillas de minucias de huellas dactilares debe ser robusto frente a estas variaciones. En la presente investigación se pretende desarrollar un componente para la comparación de plantillas de minucias de huellas dactilares, para ello se hace necesario realizar un estudio de los métodos y algoritmos disponibles en la bibliografía tanto a nivel nacional como a nivel internacional.

1.2.1 Algoritmo de coincidencia basado en el cálculo de la distancia euclídea.

Este método utiliza la representación basada en la investigación pionera de Salil Prabhakar en donde se analizan las huellas dactilares como patrones de texturas orientadas. Desde donde se extraen representaciones de texturas invariantes combinando la información discriminatoria global y local.

1.2.1.1 Extracción de características.

Primero se determina el punto central en la imagen de la huella dactilar y una región circular alrededor de este se define como la región de interés, la cual es dividida en sectores; cada sector se normaliza a una media y varianza constante, la región de interés normalizada es filtrada usando un banco de filtros de Gabor 2D; la varianza de los niveles de gris en un sector cuantifica la subyacente estructura de cresta y se define como una característica. La unión de estas características constituye un vector, denominado FingerCode[4]. Así, las características capturan la información local y la enumeración ordenada en la sectorización captura las relaciones globales invariantes entre los patrones locales.

1.2.1.2 Proceso de comparación.

La coincidencia de las plantillas de minucias de huellas dactilares está basada en la determinación de la distancia euclídea entre los correspondientes FingerCodes, la invariancia a la traslación es establecida por la identificación del punto de referencia. Sin embargo, la aproximación para la invariancia a la rotación es lograda por la rotación cíclica de las características en el vector. Un único paso de rotación corresponde a un vector de características rotado 22.5°, una rotación por R pasos corresponde a un vector de características rotado Rx 22.5°. El FingerCode obtenido después de R pasos de rotación es dado por las expresiones1, 2 y 3 rotada en los ángulos que se representan en 4 y 5:

$$\mathrm{V}_i^R\,\theta = \mathrm{V}_i{}'\theta' \quad (1)$$

$$i' = \frac{(i+k-R)}{k} + L\left(\frac{i}{k}\right) * Jk \quad (2)$$

$$S' = \frac{(S+180°+22.5°*(-\mathrm{R}))}{1800} \quad (3)$$

$$\text{i E [0,1,2,L ,79], (4)}$$

$$\text{S E [0° ,22.5° ,45° ,67.5° ,90° ,112.5° ,135° ,157.5°] (5)}$$

$\mathrm{V}_i^R\,\theta$ es el FingerCode rotado, y $\mathrm{V}_i{}'\theta'$ es el FingerCode original

Para cada huella dactilar en la base de datos, se almacena cinco plantillas correspondiente a las siguientes cinco rotaciones del correspondiente FingerCode: $\mathrm{V}_i^{-2}\,\theta$, $\mathrm{V}_i^{-3}\,\theta$, $\mathrm{V}_i^0\,\theta$, $\mathrm{V}_i^1\,\theta$, $\mathrm{V}_i^2\,\theta$. El vector de entrada es comparado con las cinco plantillas almacenadas en la base de datos para obtener cinco diferentes puntuaciones de coincidencia. La mínima de estas cinco puntuaciones de coincidencia corresponde al mejor alineamiento de la huella dactilar de entrada con las huellas dactilares de la base

[4] FingerCode: Es un vector que se obtiene al extraer características de las huellas dactilares.

de datos. Puesto que la generación de plantillas y almacenamiento en la base de datos es un proceso off-line la tarea de coincidencia es extremadamente rápida, el tiempo de verificación todavía depende del tiempo tomado para generar una plantilla para la imagen de prueba (16).

1.2.2 Algoritmo de coincidencia Bozorth.

Este algoritmo de comparación se desarrolló para el FBI, posteriormente el NIST lo adoptó para mejorarlo. Este tipo de algoritmo puede usarse para la identificación ("uno a muchos") o verificación ("uno a uno") y recibe el nombre de *matcher.* En el funcionamiento del mismo es importante destacar que las características de las minucias están exclusivamente limitadas a posición y orientación, representadas como {x, y, t} respectivamente. Este está diseñado para ser invariante a rotación y traslación. Se compone mayoritariamente de tres pasos que mencionaremos a continuación (17):

- Construcción de las tablas de comparación internas de impresiones dactilares: se construye una tabla para la huella dactilar tomada y otra para la que se encuentra en la base de datos con la que se comparará (17).

Primeramente, se computa las medidas relativas entre cada minucia en una huella dactilar y el resto de las minucias de la misma. Estas medidas relativas se guardan en una tabla de comparación de minucias y es lo que hace que el algoritmo sea invariante a rotación y traslación. Para tener en cuenta la posición relativa de traslación, la distancia euclidiana se computa entre la posición de ambas minucias. Esta distancia permanecerá invariante independientemente de lo que se pueda girar o trasladar la imagen.

En cuanto a la orientación, el objetivo es medir el ángulo entre la línea que marca la orientación de las minucias y la línea que une ambas minucias. De esta manera, los ángulos permanecen constantes a dicha línea, independientemente de lo que se gire la imagen. Para cada par de minucias comparadas, se introducirá una nueva entrada en la tabla de comparación que contendrá la información sobre la distancia entre minucias, los ángulos, y las propias minucias. Las entradas se ordenan en la tabla por distancia creciente hasta que se alcanza un límite superior prefijado, momento en el que la tabla se corta. Cada vez que se quieran comparar dos huellas dactilares, habrá que realizar una tabla de comparación para cada una de ellas.

- Construcción de la tabla de compatibilidad entre impresiones dactilares: compara ambas tablas y genera una nueva tabla de compatibilidad (17).

Se comparan las tablas de comparación de dos huellas dactilares separadas y se buscan entradas compatibles entre ambas. Si el resultado es positivo, entonces se efectúa una nueva entrada en la tabla de compatibilidad con la información relativa entre ambas minucias. Es por ello que en cada entrada de la tabla de compatibilidad se incluyen dos pares de minucias.

- Recorrido de la tabla de compatibilidad: se recorre la tabla y se unen sus entradas en grupos. Se combinan los grupos compatibles y se calcula una puntuación (17).

En este proceso, ya construida la tabla de compatibilidad que consiste en una lista de asociaciones compatibles entre dos pares de minucias. Estas asociaciones representan enlaces en un gráfico de compatibilidad, para determinar cómo de bien encajan entre ellos. Este se diseñó para que los recorridos de la tabla de compatibilidad se inicien desde distintos puntos de partida. A medida que los recorridos avanzan, porciones del gráfico de compatibilidad son creadas mediante la unión de las entradas de la tabla. Una vez que los recorridos han concluido, las porciones compatibles se combinan y el número de entradas de la tabla enlazadas. A través de esta combinación de porciones se suma para computar la puntuación de semejanza. Cuanto mayor sea el número de enlaces entre asociaciones compatibles, mayor será esta puntuación y mayor es la posibilidad de que las dos huellas dactilares pertenezcan a la misma persona.

1.2.3 Algoritmo de coincidencia basado en estructura local y global.

Para la realización de este algoritmo inicialmente se coteja el vector de entrada, con el vector patrón de las minucias de la huella almacenada en la base de datos. El resultado del algoritmo de comparación es el porcentaje (%) de coincidencias existente entre ambos vectores (18).

La comparación no se puede hacer directamente, y se debe tener en cuenta una serie de consideraciones previas sobre la posición de la huella dactilar, a saber:

- Desplazamiento.
- Rotación.
- Escalabilidad.

El proceso de cotejo de ambos vectores se realiza en dos fases: análisis local y global de los patrones de minucias (18).

1.2.3.1 Análisis local.

Se analiza el nivel local de la vecindad de cada una de los puntos característicos. Cada minucia es identificada y caracterizada a partir de la disposición espacial relativa a ella misma con sus doce minucias vecinas más próximas. Cada minucia es parametrizada con los siguientes datos: tipo *ti* de la minucia objeto de estudio (bifurcación o final/inicio de cresta), tipos tiv de las minucias vecinas, distancia relativa *d* entre la minucia de referencia o central y sus minucias vecinas y, por último, ángulos relativos ∅ y γ entre la orientación que toma la cresta de la huella dactilar en la minucia objeto de estudio βi y los ejes que unen a ésta con sus minucias vecinas.

Una vez caracterizadas a nivel local, tanto la minucia recién adquirida como la minucia patrón almacenada, se realiza un estudio de correlación entre ambas estructuras locales. Para ello se construye una matriz de similitud que asocia el nivel de semejanza de la estructura local de cada minucia de la huella dactilar capturada con las minucias de la huella dactilar patrón donde:

$\boldsymbol{p1 \ldots pn}$ **= Son las minucias del matriz patrón.**

$\boldsymbol{q1 \ldots qn}$ **= Son las minucias de la matriz de entrada.**

$\boldsymbol{Sij}$ **= Nivel de similitud entre las estructuras locales qi y pi.**

El grado de similitud se calcula de la siguiente forma (tanto para la minucia central del vector de entrada como del patrón):

- Las 12 minucias vecinas se ordenan de menor a mayor según su distancia con respecto a la minucia central.
- Atendiendo al orden resultante, la minucia vecina ubicada en el puesto j entrada (j<=12) de entrada y su equivalente ubicada también en la posición j patrón se comparan del siguiente modo:

1. Si son del mismo tipo se suma un punto al grado de similitud.

2. Para el vecino *j* (tanto de entrada como patrón), se calcula la distancia con respecto a la minucia central *d*. Si la diferencia entre las distancias obtenidas para los vectores entrada y patrón es menor que un umbral (5 pixeles) se suma un punto al grado de similitud.

3. Se repite el paso 2 pero en este caso para los ángulos ∅ y γ sumándose un punto si la diferencia es menor de 15 grados.

4. El proceso se repite para las 12 minucias vecinas.

1.2.3.2 Análisis global.

Obtenidas las minucias centrales el siguiente paso consiste en repetir el análisis espacial, pero en este caso a nivel global. Las minucias centrales se considerarán como puntos de referencia para corregir los errores introducidos por traslaciones y rotaciones de la huella dactilar. La forma de emparejar las minucias de las imágenes de entrada y patrón es la siguiente:

- Se toma la minucia k del vector de entrada y se calcula su distancia *de* con respecto a la minucia central.
- Se calcula la distancia d_p de todas las minucias del vector patrón con respecto a la minucia central de dicho vector. Se toman como candidatas a ser emparejadas aquellas cuya diferencia $(d_p - d_e)$ sea inferior a un umbral (20 pixeles en valor absoluto). Las minucias candidatas reciben una puntuación igual a $1 - abs(d_p - d_e)/20$. Este umbral trata de corregir los errores de escalado entre huellas.
- Para las minucias candidatas se calculan nuevamente los ángulos $\emptyset_e$ y γ_e, para el vector de entrada, y $\emptyset_d$ y $\gamma_d$para el vector patrón. Si la diferencia entre los ángulos obtenidos para la minucia de entrada $(\emptyset_e - \emptyset_d \; y \; \gamma_e - \gamma_d)$ y el patrón es inferior a un umbral (15 grados para $\emptyset_d$ y 20 grados para γ_d) se suma a la puntuación anterior los valores $1 - abs(\emptyset_e - \emptyset_d)/15$ y $1 - abs(\gamma_e - \gamma_d)/20$, respectivamente. La minucia que haya obtenido la mayor puntuación es la emparejada.
- El proceso se repite para todas las minucias del vector de entrada.

1.2.4 Algoritmo basado en coincidencia cruzada.

El proceso de comparación de este método se basa en la formación de estructuras complejas, que no son más que la unión de dos estructuras de minucias y las 5 vecindades más cercanas a una minucia central. Al descomponer una estructura compleja se crean un conjunto de tripletas de minucias que son formadas a partir de las relaciones entre la minucia central y sus 5 vecindades más cercanas, donde a cada tripleta se le adiciona un descriptor denominado tripleta primaria (Cuando uno de sus vértices coincide con la minucia central) o secundaria (Cuando ninguno de sus vértices coinciden con la minucia central) (13).

El proceso de comparación se realiza mediante el cálculo de similitud S que existe entre cada estructura compleja. Para ello se realiza una comparación cruzada donde para cada $d(i,j),\ i = 1 \dots n$ y $j =$

$1 \ldots n$ donde n es el número de estructuras en cada conjunto y $d(i,j)$ representa la comparación entre dos estructuras. El cálculo de S se realiza por la expresión 6:

$$\mathrm{S} = \frac{tp(d(i,j))}{100} + \frac{ts(d(i,j))}{100} \quad (6)$$

Donde tp representa los datos pertenecientes a las tripletas primarias y ts los datos pertenecientes a las tripletas secundarias. La comparación de cada tripleta de minucias se realiza según el índice de similitud que presentan los datos en comparación con un umbral definido. En el caso de los lados se define el descriptor desfasamiento, el cual es la diferencia en pixeles de la longitud de un lado con respecto al otro. En el caso de los ángulos internos y la diferencia de los ángulos de minucias adyacentes a un lado, se define el descriptor grados de libertad como la diferencia existente entre dos ángulos a comparar. Ambos descriptores actúan como umbrales para calcular el índice de similitud entre dos estructuras (13).

Inicialmente para la extracción de características identificativas de las plantillas de minucias se utiliza la estructura compleja como estructura de minucias. Para la formación de estas estructuras se requieren la utilización de 3 algoritmos.

1. Algoritmo para la formación de estructuras complejas de minucias.
   a) Formación de la estructura 5 vecindades más cercanas a una minucia.
   b) Extracción de las tripletas que pueden formarse utilizando el centro y las minucias vecinas de la estructura.
2. Algoritmo para la extracción de características invariantes a rotación y traslación provenientes de las tripletas.
3. Algoritmo para la clasificación de las características extraídas.

El algoritmo utilizado para la formación de las estructuras complejas comienza con la selección de un punto central $A(x,y)$ dentro de un conjunto de minucias $E(x,y,\alpha,t)$ donde (x,y) representan las coordenadas en el espacio cartesiano, α representa el ángulo de la minucia y t el tipo de minucia, utilizado en la expresión 7:

$$\sum_{i=0}^{n} \frac{x_i}{n}; \sum_{i=0}^{n} \frac{y_i}{n} \quad (7)$$

Al seleccionar el punto central se puede establecer un orden en la creación de las estructuras de minucias y teniendo como principal objetivo disminuir el costo y el tiempo computacional durante el proceso de comparación. Este punto central calcula el centro de la colección de minucias. Partiendo del punto $A(x, y)$ se busca la minucia $Z(x, y)$ más cercana a él utilizando la función de distancia 8:

$$\lambda = \min_{z_i \in E} d(A, Z_i) \quad (8)$$

Donde i adquiere valores de $0\ a\ n$, siendo n la cantidad de minucias y d la distancia euclidiana entre dos puntos (minucias). Finalmente se ordenan las minucias en contra de las manecillas del reloj. Para la formación de las 5 vecindades más cercanas se toma como centro la minucia z_0 obtenida de la expresión anterior (8) y se calculan las 5 vecindades más cercanas V utilizando la función 9:

$$\lambda = \min_{z_i \in E} d(z_0, z_i) \text{ con } z_0 \neq Z_i \quad (9)$$

Donde λ representa la distancia mínima entre una minucia z_i y la minucia central z_0. Obtenidas las 5 vecindades, a partir de la minucia central, se procede al cálculo del ángulo que forma cada una de las vecindades en relación con el centro. Este ángulo se calcula con respecto al eje x y es de gran importancia durante el proceso de comparación para poder calcular las probabilidades de que una tripleta contenga o no minucias válidas. Esto permite saber si una tripleta está formada por minucias que coinciden con el conjunto original, lo que mejora el rendimiento biométrico.

La descomposición en tripletas de minucias se realiza a partir de la formación de todas las tripletas posibles dado una estructura de 5 vecindades más cercanas. Para describir el tipo de dato que contiene cada tripleta y la relación entre las minucias se adiciona un descriptor a cada tripleta denominado primario o secundario:

1. Primario: Cuando uno de sus vértices coincide con la minucia central.

2. Secundario: Cuando ninguno de sus vértices coincide con la minucia central.

Luego se realiza el proceso de selección de tripletas de calidad el cual está dado por la eliminación de tripletas donde la suma de dos de sus ángulos sea mayor que 150 grados. Esto se debe a que se ha observado que la formación de este tipo de tripletas es menor en plantillas de minucias que pertenecen a un mismo rasgo biométrico. La estructura compleja queda como se muestra en la figura 6.

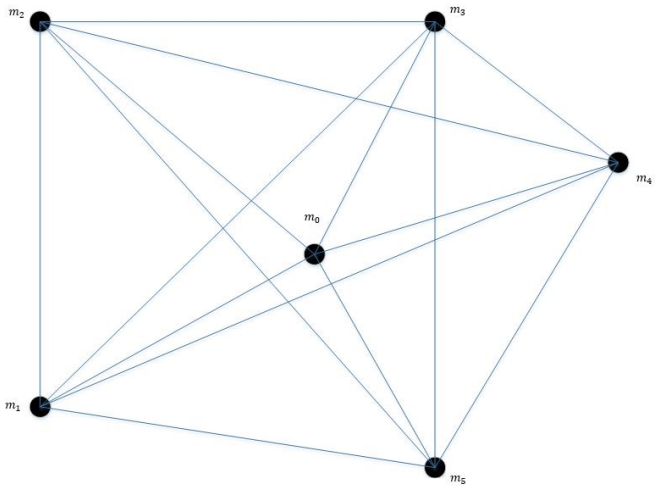

Figura 6: Estructura compleja.

De cada tripleta formada de la estructura compleja se extraen las características identificativas, las cuales son:

1. Longitud de los lados del triángulo.
2. Amplitud de los ángulos internos del triángulo.
3. Diferencia entre dos ángulos adyacentes a un lado.

Para la extracción de las características identificativas invariantes a rotación y traslación se utiliza el teorema del coseno, quedando el vector de características de la siguiente forma:

$$S_1, S_2, S_3; \alpha_1, \alpha_2, \alpha_3; \Delta\sigma_1, \Delta\sigma_2, \Delta\sigma_3$$

La longitud de los lados se representan como S_1, S_2, S_3, la de los ángulos internos como $\alpha_1, \alpha_2, \alpha_3$, la diferencia de los ángulos de las minucias adyacentes a un lado como $\Delta\sigma_1, \Delta\sigma_2, \Delta\sigma_3$ y se calcula mediante la expresión 10:

$$\Delta\boldsymbol{\sigma_1} = \mathbf{dif}(\mathbf{angulo}\boldsymbol{c_{i2}}, \boldsymbol{angulo c_{i1}}) \quad (10)$$

Para la realización de la comparación de características identificativas se define como pre condición que ambas plantillas se encuentren transformadas. Para poder hacer uso de los descriptores, primario y secundario, definidos en el método de representación y extracción de características identificativas. Este proceso de comparación de plantillas protegidas se desarrolla mediante:

1. Algoritmo para la creación de estructuras.
2. Algoritmo para la comparación de estructuras.
   a) Cálculo de la probabilidad de ocurrencia.
   b) Comparación de estructuras primarias.
   c) Comparación de estructuras secundarias.
3. Algoritmo para la consolidación de los resultados.

El método de comparación de estructuras se divide en tres algoritmos.

1. El algoritmo para el cálculo de la probabilidad de ocurrencia consiste en la comparación de los ángulos centrales de las estructuras complejas. Este algoritmo utiliza un umbral de rotación (Ur) para representar la máxima rotación permitida que pueden experimentar las minucias en cada muestra. Como objetivo se persigue conocer qué tripletas de minucias presentes en el conjunto de muestra fueron formadas a partir de minucias que no estaban en el conjunto de minucias de la plantilla de prueba.

2. El algoritmo de comparación de estructuras complejas (primarias y secundarias) define el descriptor desfasamiento (d), el cual constituye la diferencia, en pixeles, entre la longitud de los lados de cada tripleta. El descriptor grados de libertad (gl) se define como la diferencia entre dos ángulos a comparar y se encuentra asociado a los ángulos internos (gli) y la diferencia de los ángulos de dos minucias adyacentes a un lado (gld). Ambos descriptores actúan como umbrales para calcular el índice de similitud entre dos estructuras. La comparación de los descriptores consiste en hallar el módulo de la resta de los valores de cada uno de ellos como se muestra en las expresiones 11,12,13:

- $\mathbf{d} = \left|\mathbf{d_{muestra}} - \mathbf{d_{prueba}}\right|$ (11)

- $\mathbf{gli} = \left|\mathbf{gli_{muestra}} - \mathbf{gli_{prueba}}\right|$ (12)
- $\mathbf{gld} = \left|\mathbf{gld_{muestra}} - \mathbf{gld_{prueba}}\right|$ (13)

La comparación de estructuras con mayor probabilidad de ocurrencia se realiza mediante la comparación de las tripletas de primer orden y las tripletas de segundo orden de cada estructura compleja. Las tripletas que no dan un resultado positivo se comparan sin tener en cuenta si son primarias o secundarias. El cálculo de similitud entre las estructuras complejas se realiza utilizando la expresión 14:

$$\mathbf{I} = \left(\frac{\mathbf{P}}{\mathbf{P_t}} \times \mathbf{0.6}\right) + \left(\frac{\mathbf{S}}{\mathbf{S_t}} \times \mathbf{0.4}\right) \text{ (14)}$$

Donde P representa la cantidad de tripletas primarias que se comparan positivamente, P_t la cantidad de tripletas primarias en total, S la cantidad de tripletas secundarias que comparan positivamente y S_t la cantidad de tripletas secundarias total.

3. El algoritmo de consolidación de los datos consiste en calcular el índice de similitud general que tienen las estructuras complejas que comparan de manera global. Para conocer si una plantilla de prueba y una plantilla de muestra en el dominio protegido pertenecen a una misma persona se calcula el índice de similitud i entre ambas a partir de la expresión 15:

$$\mathbf{i} = (\mathbf{m}/\mathbf{n}) \times \mathbf{100} \text{ (15)}$$

Donde m representa la cantidad de estructuras complejas que aparecen en ambas plantillas y n el total de estructuras complejas.

Cada uno de los métodos descritos anteriormente representa una alternativa para el proceso de comparación de plantillas de minucias de huellas dactilares, pero para la implementación del componente se decide utilizar el algoritmo basado en coincidencia cruzada. En el método se decide seguir el enfoque de Jeffers y Arakala (19), para la extracción de características de la huella dactilar mediante las formaciones de estructuras complejas el cual tiene como objetivo simplificar el proceso de comparación y eliminar la necesidad de realizar el proceso de alineación de los datos debido a la selección de estructuras invariantes a rotación y traslación. Para el proceso de comparación se utiliza el método

basado en coincidencia cruzada, en el cual se definen umbrales para realizar la comparación de cada estructura.

1.3 Metodologías, Tecnologías y Herramientas.

1.3.1 Metodología de desarrollo de software.

Las metodologías para el desarrollo de software es un modo sistemático para realizar, gestionar y administrar un proyecto. Seleccionar una buena metodología será transcendental para el éxito de un producto. El papel principal de una metodología es guiar y organizar actividades que conlleven a las metas trazadas.

Para elegir una metodología de desarrollo de software se debe tener en cuenta dos factores fundamentales: el tipo de proyecto que se desea desarrollar y el tiempo que se dispone para desarrollar el mismo. En la actualidad no se puede afirmar que existe una metodología que funcione de manera universal, son más bien concebidas como marcos metodológicos que deben ajustarse a cada organización y tipo de proyecto a desarrollar, por lo que existen dos grandes enfoques de metodologías: las tradicionales y las ágiles. Las metodologías de desarrollo se clasifican en dos clases: las metodologías tradicionales o robustas y las ágiles o ligeras. En la siguiente tabla se presenta una comparación entre los dos tipos de metodologías, como se muestra en la figura 7, (20).

Metodologías Ágiles	Metodologías Tradicionales
Basadas en heurísticas provenientes de prácticas de producción de código	Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo
Especialmente preparados para cambios durante el proyecto	Cierta resistencia a los cambios
Impuestas internamente (por el equipo)	Impuestas externamente
Proceso menos controlado, con pocos principios	Proceso mucho más controlado, con numerosas políticas/normas
No existe contrato tradicional o al menos es bastante flexible	Existe un contrato prefijado
El cliente es parte del equipo de desarrollo	El cliente interactúa con el equipo de desarrollo mediante reuniones
Grupos pequeños (<10 integrantes) y trabajando en el mismo sitio	Grupos grandes y posiblemente distribuidos
Pocos artefactos	Más artefactos
Pocos roles	Más roles
Menos énfasis en la arquitectura del software	La arquitectura del software es esencial y se expresa mediante modelos

Figura 7: Comparativa entre las metodologías agiles y las tradicionales.

1.3.2 Metodologías tradicionales.

Las metodologías tradicionales se centran especialmente en el control del proceso, estableciendo rigurosamente las actividades involucradas, los artefactos que se deben producir, y las herramientas y notaciones que se usarán. Estas propuestas han demostrado ser efectivas y necesarias en un gran número de proyectos. Entre las principales metodologías tradicionales se encuentran RUP (Proceso unificado de modelado) y MSF (Microsoft Solution Framework).

1.3.2.1 Proceso unificado de modelado (RUP).

El Proceso Unificado de modelado es un proceso de ingeniería de software. Proporciona un acercamiento disciplinado a la asignación de tareas y responsabilidades en una organización de desarrollo. Su propósito es asegurar la producción de software de alta calidad que se ajuste a las necesidades de sus usuarios finales con unos costos y calendario predecibles.

Es una metodología tradicional de desarrollo que intenta integrar todos los aspectos a tener en cuenta durante todo el ciclo de vida del software, con el objetivo de hacer abarcables tanto pequeños como grandes proyectos, además proporciona herramientas para todos los pasos del desarrollo así como documentación en línea para sus clientes (21).

- El ciclo de vida de **RUP** se caracteriza por:

Guiado/Manejado por casos de uso.

La razón de ser de un sistema software es servir a usuarios ya sean humanos u otros sistemas; un caso de uso es una facilidad que el software debe proveer a sus usuarios. Los casos de uso reemplazan la antigua especificación funcional tradicional y constituyen la guía fundamental establecida para las actividades a realizar durante todo el proceso de desarrollo incluyendo el diseño, la implementación y las pruebas del sistema (21).

Centrado en arquitectura.

La arquitectura involucra los elementos más significativos del sistema y está influenciada entre otros por plataformas de software, sistemas operativos, manejadores de bases de datos, protocolos, consideraciones de desarrollo como sistemas heredados y requerimientos no funcionales. Se representa mediante varias vistas que se centran en aspectos concretos del sistema, abstrayéndose de lo demás. Todas las vistas juntas forman el llamado modelo 4+1 de la arquitectura, recibe este nombre porque lo

forman las vistas lógica, de implementación, proceso y despliegue, más la de casos de uso que es la que da cohesión a todas (21).

Iterativo e Incremental.

Para hacer más manejable un proyecto se recomienda dividirlo en ciclos. Para cada ciclo se establecen fases de referencia, cada una de las cuales debe ser considerada como un mini proyecto cuyo núcleo fundamental está constituido por una o más iteraciones de las actividades principales básicas de cualquier proceso de desarrollo. En concreto RUP divide el proceso en cuatro fases, dentro de las cuales se realizan varias iteraciones en número variable según el proyecto y en las que se hace un mayor o menor hincapié en las distintas actividades.

- **RUP** divide el proceso de desarrollo del software en cuatro fases (21).

Inicio.

La fase de inicio trata de responder a estas preguntas: ¿Cuál es el objetivo? ¿Es factible? ¿Lo construimos o lo compramos? ¿Cuánto va a costar? y a otras más. Sin embargo, se pretende hacer una estimación precisa o la captura de todos los requisitos. Más bien se trata de explorar el problema lo justo para decidir si vamos a continuar o a dejarlo.

Elaboración.

El propósito de la fase de elaboración es analizar el dominio del problema, establecer los cimientos de la arquitectura, desarrollar el plan del proyecto y eliminar los mayores riesgos. Cuando termina esta fase se llega al punto de no retorno del proyecto.

Construcción.

La finalidad principal de esta fase es alcanzar la capacidad operacional del producto de forma incremental a través de las sucesivas iteraciones. Durante esta fase todas los componentes, características y requisitos, que no lo hayan sido hecho hasta ahora, han de ser implementados, integrados y testeados, obteniéndose una versión del producto que se pueda poner en manos de los usuarios (una versión beta).

Transición.

La finalidad de la fase de transición es poner el producto en manos de los usuarios finales, para lo que típicamente se requerirá desarrollar nuevas versiones actualizadas del producto, completar la documentación, entrenar al usuario en el manejo del producto, y en general tareas relacionadas con el ajuste, configuración, instalación y usabilidad del producto.

1.3.2.2 Microsoft Solution Framework (MSF):

El modelo de proceso de MSF combina el concepto de la administración de proyectos tradicional (cascada), con los modelos en espiral (mejora continua) para capitalizar en las fortalezas de cada uno de estos enfoques. MSF combina los beneficios de la planeación en cascada basado en el alcance de hitos, con los entregables iterativos e incrementales del modelo en espiral (22).

Componentes de MSF (22)**.**

Principios:

- Fortalecer el equipo brindándoles capacitación
- Asignación de responsabilidades y autoridad
- Comunicaciones abiertas
- Agregar valor
- Calidad
- Aprender experiencias

Disciplinas:

- Gestión de proyectos
- Control de riesgos
- Control de cambios

Modelos:

- Modelo de equipo de trabajo
- Modelo del proceso
- Faces de la metodología

Metodologías agiles.

La filosofía de las metodologías ágiles es dar mayor valor al individuo, a la colaboración con el cliente y al desarrollo incremental del software con iteraciones muy cortas. Este enfoque está mostrando su efectividad en proyectos con requisitos muy cambiantes y cuando se exige reducir drásticamente los tiempos de desarrollo pero manteniendo una alta calidad (20).

1.3.2.3 Programación Extrema (Extreme Programming, XP).

Es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en el desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. Se basa en realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios. Es especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico (20).

- **Características esenciales de XP** (20).

Historias de Usuario: Son la técnica utilizada para especificar los requisitos del software. Se trata de tarjetas de papel en las cuales el cliente describe brevemente las características que el sistema debe poseer, sean requisitos funcionales o no funcionales. El tratamiento de las historias de usuario es muy dinámico y flexible. Cada historia de usuario es lo suficientemente comprensible y delimitada para que los programadores puedan implementarla en unas semanas.

Roles XP:

- **Programador:** El programador escribe las pruebas unitarias y produce el código del sistema.
- **Cliente:** Escribe las historias de usuario y las pruebas funcionales para validar su implementación. Además, asigna la prioridad a las historias de usuario y decide cuáles se implementan en cada iteración centrándose en aportar mayor valor al negocio.
- **Encargado de pruebas:** Ayuda al cliente a escribir las pruebas funcionales. Ejecuta las pruebas regularmente, difunde los resultados en el equipo y es responsable de las herramientas de soporte para pruebas.
- **Encargado de seguimiento:** Proporciona realimentación al equipo. Verifica el grado de acierto entre las estimaciones realizadas y el tiempo real dedicado, para mejorar futuras estimaciones. Realiza el seguimiento del progreso de cada iteración.
- **Entrenador:** Es responsable del proceso global. Debe proveer guías al equipo de forma que se apliquen las prácticas XP y se siga el proceso correctamente.
- **Consultor:** Es un miembro externo del equipo con un conocimiento específico en algún tema necesario para el proyecto, en el que puedan surgir problemas.

- **Gestor:** Es el vínculo entre clientes y programadores, ayuda a que el equipo trabaje efectivamente creando las condiciones adecuadas. Su labor esencial es de coordinación.

Procesos XP: El ciclo de desarrollo consiste en los siguientes pasos.

1. El cliente define el valor de negocio a implementar.
2. El programador estima el esfuerzo necesario para su implementación.
3. El cliente selecciona qué construir, de acuerdo con sus prioridades y las restricciones de tiempo.
4. El programador construye ese valor de negocio.
5. Vuelve al paso 1.

En todas las iteraciones de este ciclo tanto el cliente como el programador aprenden. No se debe presionar al programador a realizar más trabajo que el estimado, ya que se perderá calidad en el software o no se cumplirán los plazos. De la misma forma el cliente tiene la obligación de manejar el ámbito de entrega del producto, para asegurarse que el sistema tenga el mayor valor de negocio posible con cada iteración.

Prácticas XP: La principal suposición que se realiza en XP es la posibilidad de disminuir la mítica curva exponencial del costo del cambio a lo largo del proyecto, lo suficiente para que el diseño evolutivo funcione. Esto se consigue gracias a las tecnologías disponibles para ayudar en el desarrollo de software y a la aplicación disciplinada de las siguientes prácticas.

- **El juego de la planificación**.
- **Entregas pequeñas**.
- **Metáfora**.
- **Diseño simple**.
- **Pruebas**.
- **Refactorización (*Refactoring*)**.
- **Programación en parejas**.
- **Propiedad colectiva del código.**
- **Integración continúa**.
- **40 horas por semana**.
- **Cliente in-situ**.
- **Estándares de programación**.

1.3.2.4 SCRUM.

Es un marco de trabajo por el cual las personas pueden acometer problemas complejos adaptativos, a la vez que entregar productos del máximo valor posible productiva y creativamente (23). Scrum es:

- Ligero
- Fácil de entender
- Extremadamente difícil de llegar a dominar

Está especialmente indicada para proyectos con un rápido cambio de requisitos. Sus principales características se pueden resumir en dos. El desarrollo de software se realiza mediante iteraciones, denominadas *Sprint*, con una duración de 30 días. El resultado de cada *sprint* es un incremento ejecutable que se muestra al cliente. La segunda característica importante son las reuniones a lo largo proyecto, entre ellas destaca la reunión diaria de 15 minutos del equipo de desarrollo para coordinación e integración.

El Equipo de Desarrollo

El Equipo de Desarrollo consiste en los profesionales que desempeñan el trabajo de entregar un Incremento de producto "Terminado", que potencialmente se pueda poner en producción, al final de cada Sprint. Solo los miembros del Equipo de Desarrollo participan en la creación del Incremento.

Los Equipos de Desarrollo son estructurados y empoderados por la organización para organizar y gestionar su propio trabajo. La sinergia resultante optimiza la eficiencia y efectividad del Equipo de Desarrollo (23).

Los Equipos de Desarrollo tienen las siguientes características (23):

- Son autorganizados. Nadie (ni siquiera el Scrum Master) indica al Equipo de Desarrollo cómo convertir elementos de la Lista del Producto en Incrementos de funcionalidad potencialmente desplegables.
- Los Equipos de Desarrollo son multifuncionales, contando como equipo con todas las habilidades necesarias para crear un Incremento de producto.
- Scrum no reconoce títulos para los miembros de un Equipo de Desarrollo, todos son Desarrolladores, independientemente del trabajo que realice cada persona; no hay excepciones a esta regla.

- Scrum no reconoce sub-equipos en los equipos de desarrollo, no importan los dominios particulares que requieran ser tenidos en cuenta, como pruebas o análisis de negocio; no hay excepciones a esta regla.
- Los Miembros individuales del Equipo de Desarrollo pueden tener habilidades especializadas y áreas en las que estén más enfocados, pero la responsabilidad recae en el Equipo de Desarrollo como un todo.

Artefactos de Scrum

Los artefactos de Scrum representan trabajo o valor en diversas formas que son útiles para proporcionar transparencia y oportunidades para la inspección y adaptación. Los artefactos definidos por Scrum están diseñados específicamente para maximizar la transparencia de la información clave, que es necesaria para asegurar que todos tengan el mismo entendimiento del artefacto (23).

1.3.2.5 Variación de AUP para la UCI (AUP-UCI).

Al no existir una metodología de software universal, ya que toda metodología debe ser adaptada a las características de cada proyecto (equipo de desarrollo, recursos, etc.) exigiéndose así que el proceso sea configurable. Se decide hacer una variación de la metodología AUP, de forma tal que se adapte al ciclo de vida definido para la actividad productiva de la UCI.

Descripción de las fases.

De las 4 fases que propone AUP (Inicio, Elaboración, Construcción, Transición) se decide para el ciclo de vida de los proyectos de la UCI mantener la fase de Inicio, pero modificando el objetivo de la misma, se unifican las restantes 3 fases de AUP en una sola, a la que llamaremos Ejecución y se agrega la fase de Cierre (24).

Inicio.

Durante el inicio del proyecto se llevan a cabo las actividades relacionadas con la planeación del proyecto. En esta fase se realiza un estudio inicial de la organización cliente que permite obtener información fundamental acerca del alcance del proyecto, realizar estimaciones de tiempo, esfuerzo y costo y decidir si se ejecuta o no el proyecto.

Ejecución.

En esta fase se ejecutan las actividades requeridas para desarrollar el software, incluyendo el ajuste de los planes del proyecto considerando los requisitos y la arquitectura. Durante el desarrollo se modela el negocio, obtienen los requisitos, se elaboran la arquitectura y el diseño, se implementa y se libera el producto.

Cierre.

En esta fase se analizan tanto los resultados del proyecto como su ejecución y se realizan las actividades formales de cierre del proyecto.

Descripción de las disciplinas.

AUP propone 7 disciplinas (Modelo, Implementación, Prueba, Despliegue, Gestión de configuración, Gestión de proyecto y Entorno), se decide para el ciclo de vida de los proyectos de la UCI tener 7 disciplinas también, pero a un nivel más atómico que el definido en AUP. Los flujos de trabajos: Modelado de negocio, Requisitos y Análisis y diseño en AUP están unidos en la disciplina Modelo, en la variación para la UCI se consideran a cada uno de ellos disciplinas. Se mantiene la disciplina Implementación, en el caso de Prueba se desagrega en 3 disciplinas: Pruebas Internas, de Liberación y Aceptación. Las restantes 3 disciplinas de AUP asociadas a la parte de gestión para la variación UCI se cubren con las áreas de procesos que define CMMI DEV v1.3 para el nivel 2, serían CM (Gestión de la configuración), PP (Planeación de proyecto) y PMC (Monitoreo y control de proyecto) (24).

Modelado de negocio.

El Modelado del Negocio es la disciplina destinada a comprender los procesos de negocio de una organización. Se comprende cómo funciona el negocio que se desea informatizar para tener garantías de que el software desarrollado va a cumplir su propósito. Para modelar el negocio se proponen las siguientes variantes:

1. **Casos de Uso del Negocio (CUN).**
2. **Descripción de Proceso de Negocio (DPN).**
3. **Modelo Conceptual (MC).**

A partir de las variantes anteriores se condicionan cuatro escenarios para modelar el sistema en la disciplina **Requisitos**.

Requisitos.

El esfuerzo principal en la disciplina Requisitos es desarrollar un modelo del sistema que se va a construir. Esta disciplina comprende la administración y gestión de los requisitos funcionales y no funcionales del producto. Existen tres formas de encapsular los requisitos [Casos de Uso del Sistema (CUS), Historias de usuario (HU) y Descripción de requisitos por proceso (DRP)], agrupados en cuatro escenarios condicionados por el Modelado de negocio.

Análisis y diseño.

En esta disciplina, si se considera necesario los requisitos, estos pueden ser refinados y estructurados para conseguir una comprensión más precisa de estos, y una descripción que sea fácil de mantener y ayude a la estructuración del sistema (incluyendo su arquitectura). En esta disciplina se modela el sistema y su forma (incluida su arquitectura) para que soporte todos los requisitos, incluyendo los requisitos no funcionales. Los modelos desarrollados son más formales y específicos que el de análisis.

Implementación.

En la implementación, a partir de los resultados del Análisis y Diseño se construye el sistema.

Pruebas internas.

En esta disciplina se verifica el resultado de la implementación probando cada construcción, incluyendo tanto las construcciones internas como intermedias, así como las versiones finales a ser liberadas. Se deben desarrollar artefactos de prueba como: diseños de casos de prueba, listas de chequeo y de ser posibles componentes de prueba ejecutables para automatizar las pruebas.

Pruebas de liberación.

Pruebas diseñadas y ejecutadas por una entidad certificadora de la calidad externa, a todos los entregables de los proyectos antes de ser entregados al cliente para su aceptación.

Pruebas de Aceptación.

Es la prueba final antes del despliegue del sistema. Su objetivo es verificar que el software está listo y que puede ser usado por usuarios finales para ejecutar aquellas funciones y tareas para las cuales el software fue construido.

Características por escenarios (24)**.**

Escenario No 1.

Aplica a los proyectos que hayan evaluado el negocio a informatizar y como resultado obtengan que puedan modelar una serie de interacciones entre los trabajadores del negocio/actores del sistema (usuario), similar a una llamada y respuesta respectivamente, donde la atención se centra en cómo el usuario va a utilizar el sistema. Es necesario que se tenga claro por el proyecto que los casos de uso del negocio muestran como los procesos son llevados a cabo por personas y los activos de la organización.

Escenario No 2.

Aplica a los proyectos que hayan evaluado el negocio a informatizar y como resultado obtengan que no es necesario incluir las responsabilidades de las personas que ejecutan las actividades, de esta forma modelarían exclusivamente los conceptos fundamentales del negocio. Se recomienda este escenario para proyectos donde el objetivo primario es la gestión y presentación de información.

Escenario No 3.

Aplica a los proyectos que hayan evaluado el negocio a informatizar y como resultado obtengan un negocio con procesos muy complejos, independientes de las personas que los manejan y ejecutan, proporcionando objetividad, solidez, y su continuidad. Se debe tener presente que este escenario es muy conveniente si se desea representar una gran cantidad de niveles de detalles y la relaciones entre los procesos identificados.

Escenario No 4.

Aplica a los proyectos que hayan evaluado el negocio a informatizar y como resultado obtengan un negocio muy bien definido. El cliente estará siempre acompañando al equipo de desarrollo para convenir los detalles de los requisitos y así poder implementarlos, probarlos y validarlos. Se recomienda en proyectos no muy extensos, ya que una HU no debe poseer demasiada información.

Todas las disciplinas antes definidas (desde Modelado de negocio hasta Pruebas de Aceptación) se desarrollan en la Fase de Ejecución, de ahí que en la misma se realicen Iteraciones y se obtengan resultados incrementales.

Descripción de los roles.

AUP propone 9 roles (Administrador de proyecto, Ingeniero de procesos, Desarrollador, Administrador de BD, Modelador ágil, Administrador de la configuración, Stakeholder, Administrador de pruebas, Probador), se decide para el ciclo de vida de los proyectos de la UCI tener 10 roles, manteniendo algunos de los propuestos por AUP y unificando o agregando otros (24).

- Jefe de proyecto
- Planificador
- Analista Arquitecto de información (Opcional)
- Desarrollador
- Administrador de la configuración
- Stakeholder (Cliente/Proveedor de requisitos)
- Administrador de calidad
- Probador
- Arquitecto de software (Sistema)
- Administrador de BD

Selección de la metodología a utilizar.

Luego del análisis de las características de diferentes metodologías de desarrollo de software, se decide que AUP-UCI sea la que dirija el proceso de desarrollo del componente en cuestión puesto que:

- Esta metodología consiste centrar la atención en actividades que son esenciales para el desarrollo, no en todas las que forman parte del proyecto. Por lo que el equipo de trabajo no va a leer detalladamente el proceso de documentación, ya que se sabe en lo que se está trabajando.
- Con el uso de esta metodología se puede utilizar cualquier conjunto de herramientas, pero lo aconsejable son las simples, por ser fáciles de manejar y entender o las de código abierto.
- Es una metodología de fácil adaptación, siempre satisfaciendo las necesidades propias de sus usuarios, por lo que no es necesario comprar una herramienta especial o tomar un curso para poder adaptar un proyecto utilizando AUP-UCI.

1.3.2 Tecnologías y Herramientas.

El proceso de desarrollo de software se sustenta en el uso de diferentes herramientas y tecnologías, las cuales, unidas a la metodología seleccionada, conforman el ambiente de desarrollo de un sistema. Por este motivo se decide estudiar tecnologías y herramientas actuales para seleccionar aquellas que apoyarán el ciclo de vida del desarrollo del componente.

1.3.3 Lenguaje de Modelación.

Un lenguaje para el modelado de objetos es un conjunto estandarizado de símbolos y de modos de disponerlos para modelar un diseño de software orientado a objetos.

1.3.3.1 UML

El Lenguaje Unificado de Modelado **(UML)** es un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software. Captura decisiones y conocimiento sobre los sistemas que se deben construir. Se usa para entender, diseñar, configurar, mantener, y controlar la información sobre tales sistemas. Está pensado para usarse con todos los métodos de desarrollo, etapas del ciclo de vida, medios y dominios de aplicación. El lenguaje de modelado pretende unificar la experiencia pasada sobre técnicas de modelado e incorporar las mejores prácticas actuales en un acercamiento estándar.

UML incluye conceptos semánticos, notación, y principios generales. Tiene partes estáticas, dinámicas, de entorno y organizativas. Está pensado para ser utilizado en herramientas interactivas de modelado visual que tengan generadores de código, así como generadores de informes. La especificación de UML no define un proceso estándar, pero está pensado para ser útil en un proceso de desarrollo iterativo. Pretende dar apoyo a la mayoría de los procesos de desarrollo orientados a objetos.

1.3.4 Herramientas para el diseño.

Herramienta CASE (Ingeniería del software asistida por computadoras).

Son diversas aplicaciones informáticas o programas informáticos destinadas a aumentar la productividad en el desarrollo de software reduciendo el costo de las mismas en términos de tiempo y de dinero. Estas herramientas pueden ayudar en todos los aspectos del ciclo de vida de desarrollo del software en tareas como el proceso de realizar un diseño del proyecto, cálculo de costos, implementación de parte del código automáticamente con el diseño dado, compilación automática, documentación o detección de errores entre otras.

1.3.4.1 Rational Rose 8.1.

Rational Rose es una herramienta de producción y comercialización establecida por *Rational Software Corporation*. Utiliza el Lenguaje Unificado (UML) para facilitar la captura de dominio de la semántica, la arquitectura y el diseño. Este software tiene la capacidad de crear, ver, modificar y manipular los componentes de un modelo. No es gratuito, admite la integración con otras herramientas de desarrollo. Habilita asistentes para crear clases y provee plantillas de código que pueden aumentar significativamente la cantidad de código fuente generada, además de tener la capacidad de integrarse con otras herramientas de desarrollos (IDEs) (25).

1.3.4.2 Visual Paradigm 8.0.

Visual Paradigm para UML es una herramienta CASE que soporta el modelado mediante UML y proporciona asistencia a los analistas, ingenieros de software y desarrolladores, durante todos los pasos del Ciclo de Vida de desarrollo de un Software (26).

Las ventajas que proporciona Visual Paradigm son (26)**:**

- **Dibujo:** facilita el modelado de UML, ya que proporciona herramientas específicas para ello. Esto también permite la estandarización de la documentación, ya que la misma se ajusta al estándar soportado por la herramienta.
- **Corrección sintáctica:** controla que el modelado con UML sea correcto.
- **Coherencia entre diagramas:** al disponer de un repositorio común, es posible visualizar el mismo elemento en varios diagramas, evitando duplicidades.
- **Integración con otras aplicaciones:** permite integrarse con otras aplicaciones, como herramientas ofimáticas, lo cual aumenta la productividad.
- **Trabajo multiusuario:** permite el trabajo en grupo, proporcionando herramientas de compartición de trabajo.
- **Reutilización:** facilita la reutilización, ya que disponemos de una herramienta centralizada donde se encuentran los modelos utilizados para otros proyectos.
- **Generación de código:** permite generar código de forma automática, reduciendo los tiempos de desarrollo y evitando errores en la codificación del software.
- **Generación de informes:** permite generar diversos informes a partir de la información introducida en la herramienta.

1.3.5 Lenguajes de programación.

Es un idioma artificial diseñado para expresar computaciones que pueden ser llevadas a cabo por máquinas como las computadoras. Pueden usarse para crear programas que controlen el comportamiento físico y lógico de una máquina, para expresar algoritmos con precisión, o como modo de comunicación humana. Está formado de un conjunto de símbolos y reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones. Al proceso por el cual se escribe, se prueba, se depura, se compila y se mantiene el código fuente de un programa informático se le llama programación.

1.3.5.1 Lenguaje C# 6.0.

C# es un lenguaje de programación diseñado por Microsoft para su plataforma .NET. Aunque es posible escribir código para la plataforma .NET en muchos otros lenguajes, C# es el único que ha sido diseñado específicamente para ser utilizado en ella, por lo que programarla usando C# es mucho más sencillo e intuitivo que hacerlo con cualquiera de los otros lenguajes. Por esta razón, se suele decir que C# es el lenguaje nativo de .NET**.** La sintaxis y estructuración de C# es muy parecida a la de C++ o Java, puesto que la intención de Microsoft es facilitar la migración de códigos escritos en estos lenguajes a C# y facilitar su aprendizaje a los desarrolladores habituados a ellos. Su sencillez y el alto nivel de productividad son comparables con los de Visual Basic (27).

1.3.5.2 Lenguaje Java 7

Java es un lenguaje de programación de alto nivel, orientado a objetos, multihebra, usado para escribir tanto programas auto contenidos como "applets"[5], estos últimos como parte de los documentos HTML (hypertext markup language)[6] que se encuentran disponibles en páginas WWW. Los applets permiten a las páginas de red tener un contenido ejecutable accesible con un navegador habilitado con Java (28).
Es un lenguaje independiente de la plataforma, lo que significa que los programas desarrollados en java correrán en cualquier sistema sin cambios. Esta independencia de plataforma se logró usando un formato especial para los programas compilados en java. Este formato de archivos puede ser leído y compilado por cualquier computadora que tenga intérprete java (29).

Principales características de Java (28):

- Una vez que el compilador e intérprete de Java han sido portados a una nueva plataforma, un programa Java puede ejecutarse en tal plataforma sin cambios.
- Java es un lenguaje fuertemente tipificado: el compilador hace varias verificaciones, tal como comparar los tipos de los argumentos que se pasan a los métodos para asegurar que se ajustan a los tipos de los parámetros formales.
- El intérprete hace muchas verificaciones durante el tiempo de ejecución, tal como asegurar que el ´índice de un arreglo no salga de sus límites, o que una referencia a un objeto no sea nula.

[5] Applet: es un componente de una aplicación que se ejecuta en el contexto de otro programa.
[6] HTML (hypertext markup language): hace referencia al lenguaje de marcado para la elaboración de páginas web.

- No hay aritmética de apuntadores: las ´únicas operaciones permitidas sobre las referencias a un objeto son asignación de otra referencia a objeto y comparación con otra referencia. Por lo tanto, el programador tiene menor posibilidad de escribir código susceptible a errores que modifique un apuntador.

1.3.5.3 Lenguaje C++ 5.1.

Bjarnes Stroutstrup diseñó y desarrolló C++ en 1983 buscando un lenguaje con las opciones de programación orientada a objetos. En el año 1995, se incluyeron algunas bibliotecas de funciones al lenguaje C, y con base en ellas, se pudo en 1998 definir el estándar de C++.

C++ es de nivel medio, pero no porque sea menos potente que otro, sino porque combina la programación estructurada de los lenguajes de alto nivel con la flexibilidad del ensamblador. Al referirnos a lenguaje estructurado debemos pensar en funciones, y también a sentencias de control (if, while, etc.), por lo que se dice que es también es orientado a objetos, como en el caso de Java.

C++ es un súper conjunto de C, cualquier compilador de C++ debe ser capaz de compilar un programa en C. De hecho, la mayoría admite tanto código en C como en C++ en un archivo. Por esto, la mayoría de desarrolladores compilan con C++ su código escrito en C, incluso hay quienes, siendo código en C ponen la extensión CPP (extensión de los archivos de código C++) y lo compilan con C++. Algunas personas podrían pensar que entonces C++ desplazó a C, y en algunos aspectos podría ser cierto, pero también es cierto que algunas soluciones a problemas requieren de la estructura simple de C más que la de C++ (30).

1.3.6 Entornos integrados de desarrollo.

Muchas personas aprenden a programar utilizando un editor de texto simple. Pero la mayoría, finalmente, terminan haciendo uso de algún entorno de desarrollo integrado IDE para crear aplicaciones.
Un entorno integrado de desarrollo IDE es un tipo de software compuesto por un conjunto de herramientas de programación. En concreto se compone de:

- Editor de código de programación.
- Compilador.
- Intérprete.
- Depurador.
- Constructor de interfaz gráfico.

Normalmente, un IDE está dedicado a un determinado lenguaje de programación. No obstante, las últimas versiones de los IDEs tienden a ser compatibles con varios lenguajes (por ejemplo, Eclipse, NetBeans, Microsoft Visual Studio...) mediante la instalación de plugins adicionales (31).

1.3.6.1 Eclipse 4.6.

Eclipse es una plataforma de desarrollo *open source* basada en Java. Es un desarrollo de IBM cuyo código fuente fue puesto a disposición de los usuarios. En sí mismo Eclipse es un marco y un conjunto de servicios para construir un entorno de desarrollo a partir de componentes conectados (*plug-in*). Hay plug-ins para el desarrollo de Java (JDT Java Development Tools) así como para el desarrollo en C/C++, COBOL, etc (32).

Fue concebida desde sus orígenes para convertirse en una plataforma de integración de herramientas de desarrollo. No tiene en mente un lenguaje específico, sino que es un IDE genérico, aunque goza de mucha popularidad entre la comunidad de desarrolladores del lenguaje Java (33).

1.3.6.2 NetBeans 8.2.

NetBeans IDE es un software libre desarrollado inicialmente por Sun Microsystems. Es una herramienta muy útil para desarrollar aplicaciones para internet, dispositivos móviles, animaciones, soluciones de negocios, etc. usando lenguajes de programación como Java, PHP, C/C++, JavaScript, Groovy y Ruby. Puede ser instalado en sistemas operativos como Windows, Mac, Linux y Solaris.

NetBeans es un proyecto público que consiste en un entorno integrado de desarrollo, de código abierto con una gran base de usuarios y con una comunidad en constante crecimiento (34).

Características de NetBeans IDE (34)**.**

Gran soporte: Cualquier novedad del lenguaje es rápidamente soportada por NetBeans.

Asistentes: Dispone de asistentes para la creación y configuración de distintos proyectos, incluida la elección de algunos “frameworks”.

Buen editor de código: Con el habitual coloreado y sugerencias de código, acceso a clases pinchando en el código, control de versiones, localización de ubicación de la clase actual, comprobaciones sintácticas y semánticas, plantillas de código, herramientas de refactorización.

Simplifica la gestión de grandes proyectos: Con el uso de diferentes vistas estructura la visualización de manera ordenada.

Herramientas para depurado de errores: El “Debugger” que incluye el IDE es bastante útil para encontrar dónde fallan las cosas. Se pueden definir puntos de ruptura en la línea de código que nos interese y monitorizar en tiempo real los valores de propiedades y variables.

Optimización de código: Por su parte, el “Profiler” ayuda a optimizar las aplicaciones e intenta hacer que se ejecuten más rápido y con el mínimo uso de memoria. Se puede ver el comportamiento de nuestra aplicación y obtener indicadores e información de cómo y cuántos recursos consume, cuántos objetos se crean y obtener capturas del estado del sistema en diferentes momentos.

Acceso a base de datos: Desde el propio NetBeans podemos conectarnos a distintos sistemas gestores de bases de datos, como pueden ser Oracle, MySQL y demás, ver las tablas, realizar consultas y modificaciones, y todo ello Integrado en el propio IDE.

1.3.6.3 Framework Qt Creator 5.4.1.

Qt Creator es un Entorno Integrado de Desarrollo o IDE (editor + compilador + depurador) moderno, potente, fácil de manejar, eficiente, abierto y gratuito, que permite el desarrollo rápido de aplicaciones en entornos MS Windows, Mac OS y Linux. Algunos ejemplos de programas creados con las librerías Qt: Adobe Photoshop Album, Google Earth, KDE, Opera, Skype, VLC media player (35).

Características fundamentales de **Qt Creator** (35):

- Utiliza el lenguaje de programación orientado a objetos C++.
- Se basa en Qt, una librería multiplataforma y gratuita para la creación de interfaces gráficos, programación web, multihilo, bases de datos.
- Permite realizar programación visual y programación dirigida por eventos.
- Características avanzadas de IDE: sintaxis coloreada, compleción automática de código, ayuda sensible al contexto, inspector de objetos, diseñador visual, compilador y depurador integrados.

Programación visual:

El programador se centrará en diseñar el aspecto gráfico de la aplicación, la distribución de los elementos visuales (llamados ***widgets***: formularios, botones, menús, cuadros de texto, etc.), la interacción entre los mismos, los distintos tipos de ventanas existentes.

Un entorno de programación visual se asemeja a un programa de dibujo, donde la imagen es una **ventana** (o **formulario**), y los elementos para dibujar son botones, etiquetas de texto; por lo que el programador diseñará el aspecto gráfico que tendrá la aplicación (35).

Programación dirigida por eventos:
El programador escribirá el código que se ejecutará en respuesta a determinados eventos (llamados ***slots***: pulsar un botón, elegir una opción del menú, abrir o cerrar una ventana (35).

- No existe la idea de un control de flujo secuencial en el programa, sino que el programador toma el control cuando se dispara un evento.
- La labor del programador es asociar a cada evento el comportamiento adecuado.

1.4 Propuesta de las herramientas, metodologías y tecnologías a utilizar.

Como herramientas y tecnologías a utilizar para el desarrollo del componente se propone que:

- El modelado del software se realice usando los lenguajes UML en su versión 6.4 y para especificar, construir y definir de forma gráfica y documental el diseño de la solución, proporcionándole un soporte concreto a la metodología seleccionada.
- La herramienta CASE a utilizar para el proceso de modelado sea Visual Paradigm Enterprise Edition en su versión 8.0 ya que esta constituye un software de alta eficiencia que permite realizar ingeniería tanto inversa como directa, es de software libre y permite la generación de documentación de forma automática en diferentes formatos.
- El lenguaje de programación sea C++ porque combina la programación estructurada de los lenguajes de alto nivel con la flexibilidad del ensamblador y brinda beneficios sobre otros lenguajes de programación.
- Como entorno integrado de desarrollo se usará Qt Creator en su versión 5.4.1 porque es la herramienta que permite obtener la productividad de los desarrolladores, haciendo la programación en C++ más veloz, fácil e intuitivo, ya que un solo código fuente permite menor tiempo invertido en mantenimientos, multiplicando los resultados de los esfuerzos de desarrollo. El acceso completo a su código fuente, permite adaptar y extender Qt a necesidades particulares.

1.5 Conclusiones parciales.

1. La recopilación de los referentes teóricos asociados al proceso de comparación de plantillas de minucias de huellas dactilares posibilitó un mejor entendimiento del contexto de la investigación y de la problemática a resolver.
2. El análisis de los algoritmos y métodos de comparación de plantillas de minucias de huellas dactilares facilitó la selección del método de comparación a implementar en el componente.
3. El análisis de las metodologías, tecnologías y herramientas posibilitaron la selección del ambiente de desarrollo adecuado para el desarrollo del componente.

Capítulo 2: Descripción de la solución.

Introducción

En el presente capítulo se exponen las principales características de la solución informática a desarrollar. Se realiza la captura de los requisitos funcionales y no funcionales, se conforman las historias de usuario correspondientes a estos. Se describe la arquitectura del módulo, así como los patrones de diseño a aplicar y el diagrama de clases del diseño a utilizar, logrando un mejor entendimiento para el desarrollo del sistema.

2.1 Descripción del problema.

En el proceso de comparación de plantillas dactilares tanto el algoritmo extracción de minucias como el de comparación de las mismas inciden en la eficiencia, rapidez y aceptación de un sistema biométrico de huellas dactilares. Ambos algoritmos deben ser robustos y precisos, puesto que se pueden extraer y comparar minucias falsas dando paso al aumento de las tazas de falsos aceptados y falsos rechazados. En el centro CISED se está desarrollando un componente de comparación de plantillas de minucias de huellas dactilares. Con el objetivo de mejorar los resultados obtenidos por el actual componente utilizado por el centro, ya que no son los mismo que se encuentran documentados.

2.2 Modelo de dominio.

En el modelo de dominio se muestran los principales conceptos con los que trabaja la aplicación en desarrollo, lo que ayuda a comprender el entorno del sistema. Este modelo contribuirá a describir las clases más importantes dentro del contexto de la aplicación.

A través de este modelo se observa:

- ✓ **Usuario:** Es el encargado de seleccionar la plantilla de minucias para iniciar el proceso de comparación. Carga una o varias plantillas de minucias y a partir de estas el usuario inicia el proceso para comparar una o varias plantillas dactilares.
- ✓ **Plantilla_huella_Dactilar:** Almacena información de la huella dactilar, está compuesta por un conjunto de minucias en estándar ANSI/INCITS 378.
- ✓ **Estructura_compleja:** Resultado que se genera al formar relaciones entre las minucias que conforman una vecindad. Este resultado es la conformación de tripletas.
- ✓ **Caract_invariante_traslacion_rotacion:** Es el resultado que se forma de extraer un conjunto de características identificativas mediante ecuaciones matemáticas aplicadas a cada tripleta.

✓ **BD_plantillas_dactilares:** Banco de datos donde se almacenan plantillas dactilares con todas sus características identificativas.

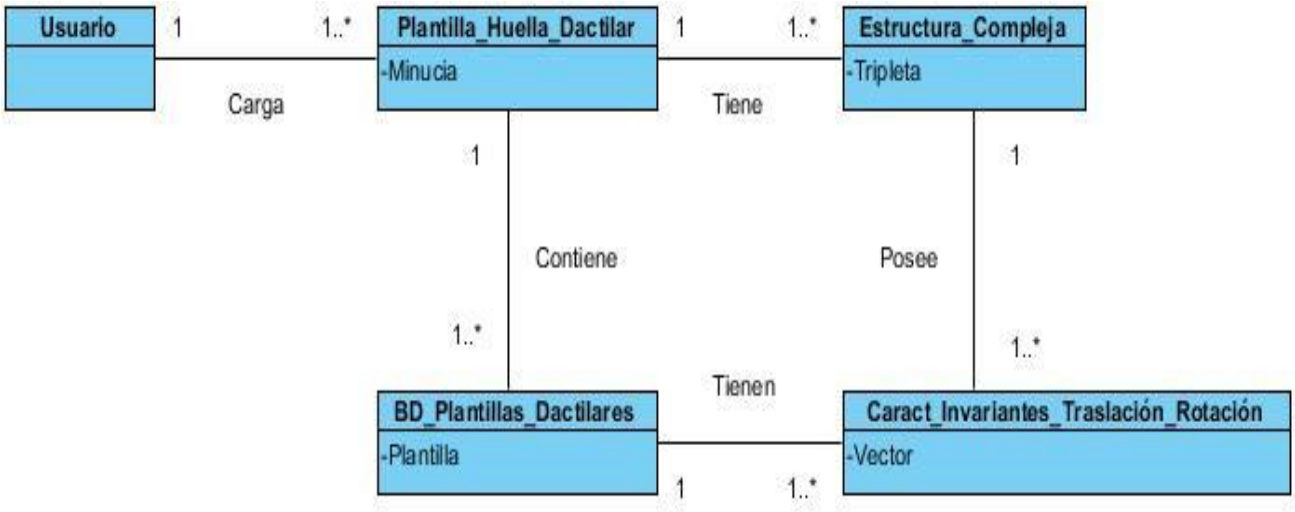

Figura 8: Modelo de dominio o conceptual.

2.3 Propuesta de solución.

Para el desarrollo de la presente investigación se propone desarrollar un componente de comparación de plantillas de minucias de huellas dactilares. El componente debe implementar el proceso de comparación descrito en (13). El proceso comienza con la lectura de dos plantillas de minucias de huellas dactilares en formato ANSI/INCITS 378-2004. Se implementa el algoritmo propuesto en (13) para la extracción de características invariantes a rotación y traslación y resistentes a deformación no lineal y superposición parcial. En este paso se forman las estructuras complejas compuestas por las cinco vecindades más cercanas a una minucia y las tripletas de minucias que se forman entre las vecindades.

Luego de formar las estructuras complejas se extrae la información de la estructura. En la presente investigación se decidió tomar la longitud de los lados, la amplitud de los ángulos internos y la diferencia entre los ángulos adyacentes a un lado. Una vez extraídos estos datos se conforma un vector de nueve posiciones el cual es utilizado para realizar la comparación cruzada con los vectores almacenados en la base de datos biométrica. Para cada estructura se calcula un índice de similitud el cual describe en qué medida se ve reflejada una estructura de la plantilla de muestra en la plantilla de prueba. Finalmente se

calcula el índice de similitud global el cual describe el nivel de similitud que existe entre ambas plantillas de minucias de huellas dactilares. El proceso se puede apreciar en la figura 9.

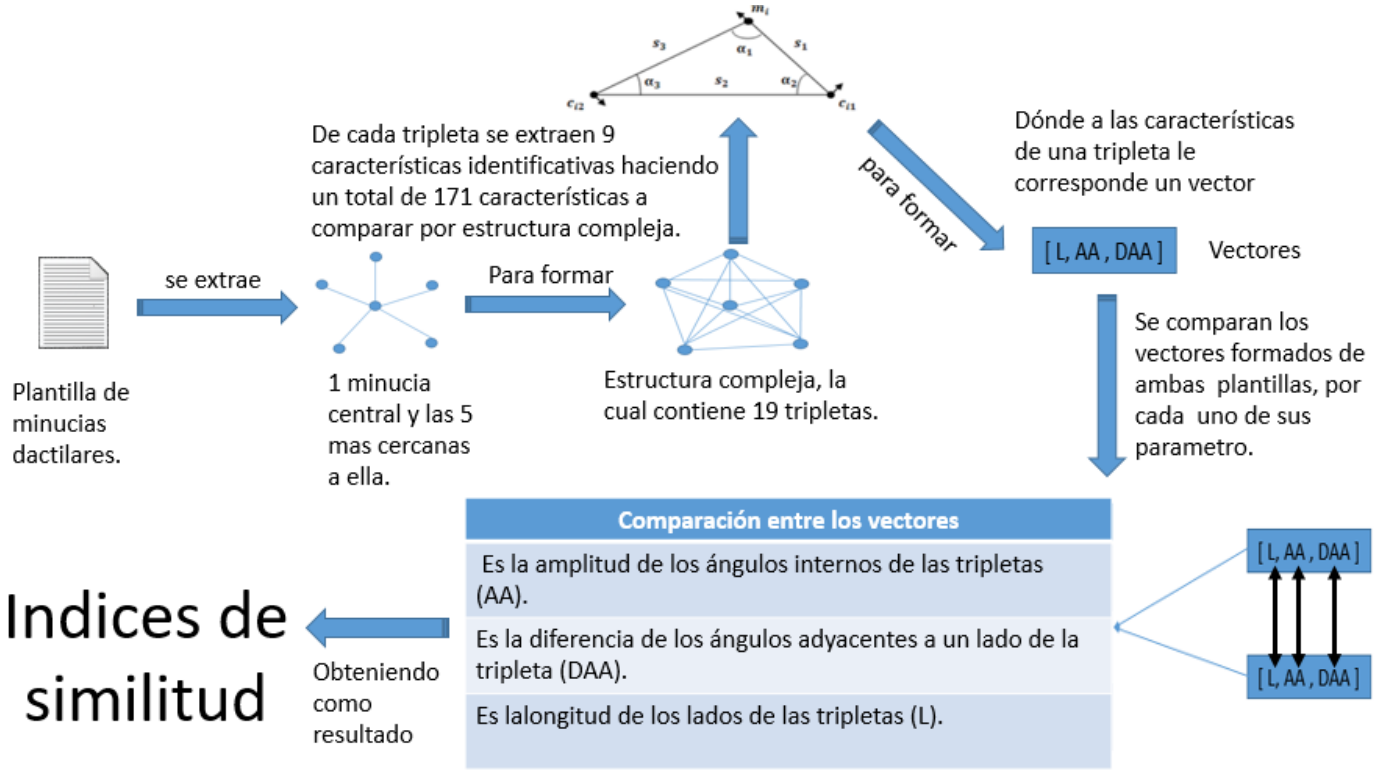

Figura 9: Representación de la propuesta de solución

La comparación de cada vector se realiza según el índice de similitud que presentan los datos en comparación con un umbral definido. En el caso de los lados se define un descriptor, el cual será la diferencia en pixeles de la longitud de un lado con respecto al otro. En el caso de la amplitud de los ángulos internos y la diferencia de los ángulos de minucias adyacentes a un lado, también se le definen sus descriptores, ya que estos actúan como umbrales para calcular el índice de similitud entre dos plantillas.

2.4 Captura de requisitos.

La captura de requisitos es uno de los pasos cruciales en el desarrollo de cualquier software. Esta tarea está encaminada a identificar lo que el cliente quiere, analizar las necesidades y especificar los requerimientos de la solución sin ambigüedades.

Requisitos funcionales.

- ✓ **RF-1 Cargar plantilla biométrica.**
- ✓ **RF-2 Formación de la estructura compleja de minucias.**

a) Formación de la estructura 5 vecindades más cercanas a una minucia.

b) Creación de tripletas a partir de las vecindades.

c) Agregar un descriptor dependiendo el tipo de tripleta.

- ✓ **RF-3 Extraer características invariantes a rotación y traslación.**

a) Extracción de características de cada tripleta para formar vectores.

b) Agregar el descriptor a cada vector, según su descriptor en la tripleta.

- ✓ **RF-4 Comparar las plantillas biométricas.**

a) Realización del cálculo de similitud existente entre cada estructura compleja.

- ✓ **RF-5 Consolidar resultados de la comparación.**
- ✓ **RF-6 Calcular umbral de similitud.**

Requisitos no funcionales.

- ✓ **RNF-1 Software.**

a) IDE de desarrollo Qt-Creator 5.4.1

b) Sistema operativo: Windows o Linux.

- ✓ **RNF-2 Restricciones en el Diseño y en la Implementación.**

a) La realización del componente debe ser desarrollada en C++.

2.5 Historias de usuario (HU).

Las historias de usuario son la técnica utilizada en AUP-UCI para especificar los requisitos del software. Se trata de tarjetas de papel en las cuales el cliente describe brevemente las características que el sistema debe poseer. Las estimaciones de esfuerzo asociado a la implementación de las historias la establecen los programadores utilizando como medida el punto. Un punto, equivale a una semana ideal de programación. La redacción de las mismas se hace bajo la terminología del cliente, de forma que sea sencilla, clara y no profundicen los detalles.

Tabla 1: HU_Cargar plantillas biométricas.

Historia de Usuario	
Número: HU_1	**Usuario:** Sistema.
Nombre de historia: Cargar plantilla biométrica.	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alto
Puntos estimados: 3	**Iteraciones asignadas:**1 y 2
Programador responsable: Luis Yasiel Silva García.	
Descripción: El sistema debe cargar plantillas biométricas de una ubicación en carpeta.	
Observaciones: Se debe tener una plantilla biométrica en la ubicación de carpeta disponible.	

Tabla 2: HU_Formación de la estructura compleja de minucias.

Historia de Usuario	
Número: HU_2	**Usuario:** Sistema.
Nombre de historia: Formación de la estructura compleja de minucias.	
Prioridad en negocio: Muy alta	**Riesgo en desarrollo:** Alto
Puntos estimados: 5	**Iteraciones asignadas:** 2
Programador responsable: Luis Yasiel Silva García.	
Descripción: El componente de formar una estructura compleja con las minucias de la plantilla. Para ello debe: • Obtener una minucia central dentro del conjunto de minucias y las 5 minucias más cercanas a ella, para formar vecindades de 6 minucias cada una. • Descomponer las vecindades para formar tripletas, que estas están compuestas por las relaciones entre las minucias de la vecindad. • Agregar un descriptor a cada tripleta. Descriptor (primario) si una de las minucias de la tripleta es la minucia central o Descriptor (secundario) si ninguna de sus minucias es la minucia central.	
Observaciones: Se debe haber cargado con éxito una la plantilla biométrica.	

Tabla 3: HU_Extraer características invariantes a rotación y traslación.

Historia de Usuario	
Número: HU_3	**Usuario:** Sistema.
Nombre de historia: Extraer características invariantes a rotación y traslación	
Prioridad en negocio: Muy alta	**Riesgo en desarrollo:** Alto
Puntos estimados: 5	**Iteraciones asignadas:** 3
Programador responsable: Luis Yasiel Silva García.	
Descripción: El componente debe extraer características invariantes a rotación y traslación para cada tripleta. Para ello se debe: • Extraer longitud de los lados, valor de ángulos internos y la diferencia de los ángulos de las minucias adyacentes de cada tripleta para formar vectores. • Agregar un descriptor a cada vector. Descriptor (primario) si la tripleta que le corresponde tiene como descriptor el primario o secundario en caso contrario.	
Observaciones: Se debe disponer de una plantilla biométrica y previamente se debe haber determinado la estructura compleja.	

Tabla 4: HU_Comparar las plantillas de minucias biométricas.

Historia de Usuario	
Número: HU_4	**Usuario:** Sistema.
Nombre de historia: Comparar las plantillas biométricas.	
Prioridad en negocio: Muy alta	**Riesgo en desarrollo:** Alto
Puntos estimados: 3	**Iteraciones asignadas:** 4
Programador responsable: Luis Yasiel Silva García.	
Descripción: Para obtener la coincidencia de plantillas biométricas se debe: • Realizar una comparación cruzada mediante la fórmula matemática de similitud $S = \frac{tp(d(i,j))}{100} + \frac{ts(d(i,j))}{100}$.	
Observaciones: Debe tenerse las características identificativas de ambas plantillas biométricas.	

2.6 Diseño.

El papel del diseño en el ciclo de vida de un software es facilitar la comprensión de su funcionamiento y proveer una representación o modelo del mismo con el propósito de definirlo con los suficientes detalles como para permitir su realización física. El modelo de diseño provee una representación arquitectónica del software que sirva de punto de partida para las tareas de implementación, dando al traste con los requisitos del sistema.

2.6.1 Descripción de la arquitectura.

La IEEE Std 1471-2000 define que la Arquitectura de Software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución (36).

Por lo que la arquitectura de software es una forma de representar sistemas mediante el uso de la abstracción, de forma que aporte el más alto nivel de comprensión de los mismos. Esta representación incluye los componentes fundamentales del software, su comportamiento y formas de interacción para satisfacer los requisitos del sistema.

Para el presente trabajo se propone la arquitectura **3-capas:**

El estilo arquitectural en 3 capas se basa en una distribución jerárquica de los roles y las responsabilidades para proporcionar una división efectiva de los problemas a resolver. Los roles indican el tipo y la forma de la interacción con otras capas y las responsabilidades la funcionalidad que implementan. Podemos decir que actualmente la programación por capas es un estilo de programación en la que el objetivo principal es separar la lógica de negocios de la lógica de diseño, un ejemplo básico de esto es separar la capa de datos de la capa de negocios y ésta a su vez de la capa de presentación al usuario (37).

En la práctica, las capas suelen ser entidades complejas, compuestas de varios paquetes o subsistemas por lo que la arquitectura propuesta para el desarrollo del componente de comparación de plantillas de minucias de huellas dactilares identifica como capas:

- **Capa de Presentación:** Representa una pequeña interfaz de prueba para el componente de comparación de plantillas de minucias de huellas dactilares.

- **Capa de Negocio:** Es donde residen los programas que se ejecutan durante la ejecución de la aplicación. Esta capa se comunica con la de presentación, para recibir las peticiones y presentar los resultados, y con la de acceso a datos, para interactuar con la base de datos, consultando y modificando sus informaciones. En la misma se encuentran las clases del negocio que no son más que aquellas que permiten la gestión de usuarios y de personas, sin olvidar las clases que conformar el motor de reconocimiento, que son las encargadas del realizar todo el proceso de identificación de la persona, como la extracción y comparación de características de la huella dactilar.
- **Capa de datos:** Representa las clases persistentes del sistema (plantillas de minucias en estándar ANSI/INCITS 378, a partir de estas se inicia el proceso de extracción y comparación de características).

Figura 10: Arquitectura 3-Capas

2.6.2 Estilo arquitectónico.

El estilo arquitectónico propuesto es Tuberías y Filtros para la infraestructura de comunicación, ya que provee una estructura para procesar flujos de datos. El sistema se percibe como una sucesión de transformaciones que sufre una serie de datos de entrada. Los datos ingresan al sistema y fluyen a través de los componentes uno a uno hasta que se asignan a un destino final: salida o almacenamiento. El sistema se divide en varios pasos secuenciales de procesamiento; los pasos se conectan con flujos de

datos; cada filtro consume y procesa sus datos en forma incremental; el destino y los filtros se conectan con tubos que implementan el flujo de datos entre los pasos de procesamiento.

2.6.3 Patrones de diseño.

Los patrones son los que comprimen el conocimiento de experiencias anteriores y pueden utilizarse en crear nuevas soluciones en contextos similares, los patrones tienen en esencia una base empírica, generalmente no son creados sino detectados, por lo que la principal fuente de patrones será tanto la aportación de los expertos como el proceso inductivo de los diseñadores.

Por lo que un patrón de diseño en el desarrollo de software es la descripción de las clases y objetos que se comunicarán entre sí de manera que puedan resolver un problema general de diseño en un contexto particular. En un contexto informático este es similar a conceptos como biblioteca de clases, frameworks, técnicas y herramientas de refactorización o programación extrema (38).

En el diseño de la aplicación se usarán los patrones **GRASP** (*General Responsibility Asignment Software Patterns*) que significa patrones generales de software para asignar responsabilidades. Dentro de los patrones de diseño GRASP se utilizó:

- ✓ **Patrón Controlador.**

Es el encargado de gestionar un evento de entrada al sistema. Es un intermediario entre la capa de presentación y el núcleo de las clases donde reside la lógica del sistema. El controlador coordina la actividad de otros objetos. Por ejemplo, en la aplicación se hace uso del patrón controlador definiendo la clase Controladora la cual gestiona todo el flujo de datos de la aplicación, y maneja varias instancias de objetos.

- ✓ **Patrón Experto.**

Se usa más que cualquier otro al asignar responsabilidades, es un principio básico que suele utilizarse en el diseño orientado a objetos. Consiste en la asignación de una responsabilidad a la clase que cuenta con la información necesaria para llevarla a cabo. El uso de este patrón da pie a un bajo acoplamiento y una alta cohesión, lo que favorece al hecho de tener sistemas más robustos y de fácil mantenimiento. El cumplimiento de una responsabilidad requiere a menudo información distribuida en varias clases de objetos. Por ejemplo, este patrón se pone de manifiesto en la clase Tripleta, la cuál es la encargada de formar vectores a partir de la misma información que contiene ella.

- ✓ **Patrón Creador.**

El patrón creador muestra la guía de cómo crear objetos cuando sus creaciones requieren tomar decisiones. Estas decisiones normalmente serán resueltas dinámicamente decidiendo que clases instanciar o sobre que objetos un objeto delegará responsabilidades (39). Se hace uso de este patrón cuando es necesario que las clases creen instancias con el nivel necesario de información para acceder a los datos almacenados, de acuerdo a la ejecución de una determinada acción. Por ejemplo, en el componente se pone en práctica en la clase Controladora, donde se crea un objeto de la clase Tripleta, en la cual para poder crear una instancia de ella hay que pasarle la información necesaria, o sea, tres puntos de minucias.

2.6.4 Diagrama de clases del diseño.

Los diagramas de clases se utilizan para modelar la visión estática de un sistema. Esta visión soporta los requisitos funcionales del sistema, en concreto, los servicios que el sistema debería proporcionar a sus usuarios finales. Normalmente contienen: clases, interfaces y relaciones entre ellas: de asociación, de dependencia y/o de generalización (40). Donde cada clase es una descripción de conjunto de objetos que comparten los mismos atributos, operaciones, métodos, relaciones y semántica.

La estructura por la cual se regirá el desarrollador para crear el componente de comparación de plantillas de minucias de huellas dactilares tendrá como base el siguiente diagrama de clases del diseño:

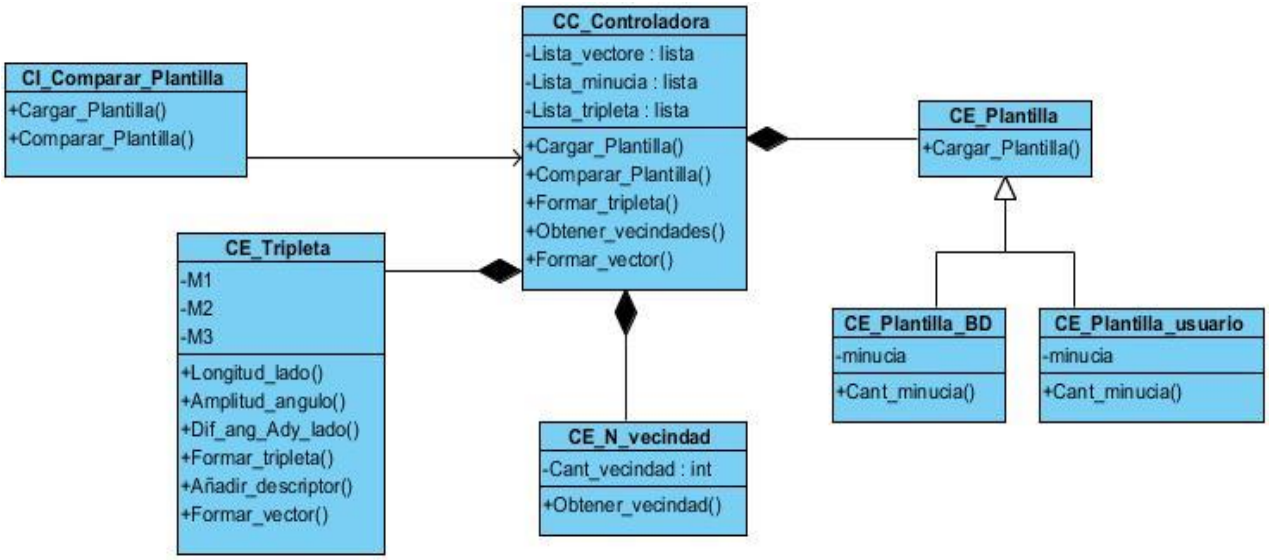

Figura 11: Diagrama de clases del diseño.

2.7 Conclusiones parciales.

1. La definición del modelo de dominio permitió identificar los requerimientos funcionales del componente.
2. El análisis y diseño de la solución facilitó la comprensión de los elementos técnicos que preceden a la implementación del mismo.
3. La elaboración de las historias de usuario mejoro la comprensión de los requisitos funcionales obtenidos en la fase de análisis de la solución.
4. La definición de la arquitectura 3-capas y algunos de los patrones GRASP, permitieron una mejor organización de la solución, de manera que las clases identificadas y sus relaciones sentaron las bases para su implementación.

Capítulo 3: Validación de la solución.

A la vez que se ha diseñado el sistema y definida la arquitectura, posteriormente se procede a la realización del componente para dar cumplimiento a los requisitos planteados. En el presente capítulo se exponen las especificaciones asociadas a la implementación del software, se describen las pautas de codificación utilizadas y las pruebas a las cual se ha sometido el componente con el objetivo de validar el correcto funcionamiento de los requerimientos del sistema.

3.1 Estándares de codificación.

Para conseguir que el código pueda ser entendido fácilmente por miembros de cualquier equipo de trabajo encargado de mantener o extender el componente desarrollado, es imprescindible el uso de estilos de codificación. Una codificación bien definida logra que todo el equipo se sienta cómodo con el código escrito por cualquier otro programador, por lo cual es indispensable que se sigan ciertos estándares de programación que provean de legibilidad al código. Los estilos definidos en este trabajo son los siguientes:

- Las variables se escriben en minúsculas y utilizando el carácter '_' para los espacios entre las palabras si es necesario. Ejemplo: "double menor_distancia".
- Los métodos se escribirán con mayúscula y se utilizará el carácter "_" como separador, si es necesario. Ejemplo: "Distancia_M(ISO_MINUTIA m1, ISO_MINUTIA m2)".
- Las clases tendrán el mismo nombre que los ficheros que las contienen, el inicio de cada palabra que conforme su nombre se escribirá con mayúscula. Ejemplo: "Plantilla".

3.2 Tareas de ingeniería.

En la etapa de implementación y prueba el artefacto que se encuentra es el de las tareas ingenieriles, la cual está encaminada a puntualizar detalladamente la realización de las historias de usuario. De cada una se obtienen al menos una tarea de ingeniería, donde cada una de estas muestras una sucesión de pasos lógicos realizados por los programadores. Cada tarea tiene exclusivamente relacionada una única historia de usuarios, un tiempo estimado, un programador asignado, fecha de inicio, fecha de fin y una breve descripción de la misma.

Tabla 5: TI_Cargar plantillas biométricas.

Tarea de ingeniería	
Número:1	**Número de la HU:**1
Nombre de la Tarea: Cargar plantilla biométrica.	
Fecha de Inicio: 4 de enero	**Fecha de Fin:** 19 de enero
Tipo de Tarea: Desarrollo	**Puntos Estimados:**1
Programador responsable: Luis Yasiel Silva García.	
Descripción: **1.** Búsqueda del directorio donde se encuentra la plantilla de minucias. **2.** Obtención de la plantilla para el procesado.	

Tabla 6: TI_Formación de la estructura compleja de minucias.

Tarea de ingeniería	
Número:2	**Número de la HU:**2
Nombre de la Tarea: Formación de la estructura compleja de minucias.	
Fecha de Inicio: 20 de enero	**Fecha de Fin:** 31 de enero
Tipo de Tarea: Desarrollo	**Puntos Estimados:**1
Programador responsable: Luis Yasiel Silva García.	
Descripción: **1.** Formación de la estructura 5 vecindades más cercanas a una minucia central. **2.** Descomposición de la estructura compleja para la formación de tripletas. **3.** Agregar descriptor identificativo a las tripletas, dependiendo si unas de sus minucias es la minucia central de la vecindad.	

Tabla 7: TI_Extraer características invariantes a rotación y traslación.

Tarea de ingeniería	
Número:3	**Número de la HU:**3
Nombre de la Tarea: Extraer características invariantes a rotación y traslación.	
Fecha de Inicio: 1 de febrero	**Fecha de Fin:** 15 de febrero
Tipo de Tarea: Desarrollo	**Puntos Estimados:**2
Programador responsable: Luis Yasiel Silva García.	
Descripción: **1.** Extracción de características identificativas de las tripletas. -Longitud de lados -Ángulos interiores -Diferencia de los ángulos de la minucia adyacente a un lado **2.** Formación de vectores utilizando las características extraídas de cada tripleta **3.** Agregar descriptor identificativo a cada vector, dependiendo del descriptor que tenía la tripleta del cual fueron escogidas sus características para su formación.	

Tabla 8: TI_Comparar las plantillas biométricas.

Tarea de ingeniería	
Número:4	**Número de la HU:**4
Nombre de la Tarea: Comparar las plantillas biométricas.	
Fecha de Inicio: 16 de febrero	**Fecha de Fin:** 28 de febrero
Tipo de Tarea: Desarrollo	**Puntos Estimados:**3
Programador responsable: Luis Yasiel Silva García.	
Descripción: **1.** Realizar una comparación cruzada mediante la fórmula matemática de similitud. $$S = \frac{tp(d(i,j))}{100} + \frac{ts(d(i,j))}{100}.$$	

3.3 Pruebas de rendimiento biométrico.

Para estimar el desempeño del componente teniendo en cuenta las tasas de error se decide optar por la utilización de una de las bases de datos más representativas a nivel internacional la *FVC2004,* las muestras encontradas en esta base de datos se caracterizan por tener una calidad entre baja y media lo cual constituye un factor importante para ser elegidas como objeto de pruebas de rendimiento biométrico.

Para la extracción de las características de las plantillas de minucias de huellas dactilares de la base de datos se utilizó el extractor del SDK propuesto por la empresa Innovatrics y se almacenaron las plantillas de minucias en texto claro. Para las entradas del experimento se utilizaron las plantillas de minucias de huellas dactilares para obtener como salida las tasas de falsos rechazos y falsos aceptados.

En la siguiente figura se exponen los resultados alcanzados por el componente sobre la base de datos DB1_B, para ver los resultados de las demás bases de datos ver en el cuerpo de anexos:

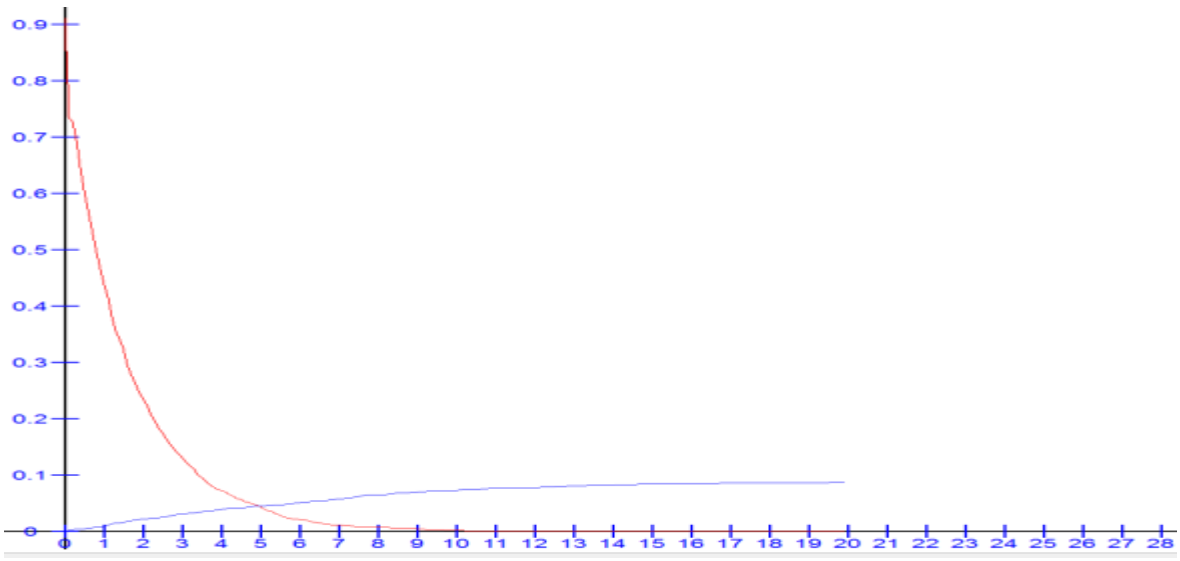

Figura 12: Resultados de las tasas para DB1_B FRA 4.96186 y FRR 0.446792.

Para comparar los datos se realizó teniendo en cuenta los datos obtenidos por la experimentación realizada en la base de datos FVC 2004 en el centro al algoritmo SourceAfis, por la brindad por el autor del motor de reconocimiento y la obtenida en la presente investigación.

Tabla 9: Tabla comparativa.

Tasas	sourceAfis del autor	sourceAfis según pruebas de los especialistas	Método basado en comparación cruzada
FAR	0.01 %	0.07866 %	0.01702118 %
FRR	10.9 %	14.11765 %	0.00087136 %

Como se puede apreciar en la tabla 9 el método implementado en la presente investigación presenta mejores resultados, arrojando tasas 0.01702118 % de falsos aceptados (FAR) y 0.00087136 % de falsos rechazados (FRR), que el método de comparación de plantillas de minucias de huellas dactilares dado por el sourceAfis del autor las cuales son 0.01 % FAR y 10.9 % FRR , y por el sourceAfis según las pruebas de los especialistas en el centro las cuales son 0.07866 % FAR y 14.11765 % FRR; por lo que se puede resumir que el método basado en comparación cruzada presenta mejores resultados desde el punto de vista teórico que los métodos de comparación de plantillas de minucias de huellas dactilares analizados.

En la tabla 10 se muestran los resultados obtenidos durante el proceso de experimentación en diferentes bases de datos los cuales están expresados en cantidad de falsos aceptados y cantidad de falsos rechazos. Estos resultados arrojan tasas de falsos aceptados y falsos rechazos menores a los publicados para el algoritmo de comparación utilizado en el SourceAfis.

Tabla 10: Error de FVC 2004.

Base de datos	Falsos aceptados	Falsos rechazos	Total de comparaciones	Tasa de error
DB1_B	349	102	3160	0.13789 %
DB2_B	956	98	3160	0.496881 %
DB3_B	1314	19	3160	0.83908%
DB4_B	572	103	3160	0.247834%

Para la realización de la tabla 10 se seleccionaron los siguientes parámetros de ajuste:

Grados de libertad de los lados de las tripletas (Gll) = 10%

Grados de libertad de los ángulos de las minucias (Gla): 12%.

Grados de validez de las tripletas (Gv): 130º.

Tasa de exactitud de las Plantillas (PUs): 3.30%

3.4 Pruebas unitarias.

La producción del código es dirigida por las pruebas unitarias. Se establecen antes de escribir el código y se ejecutan constantemente ante las modificaciones del sistema. Los elementos sometidos a las pruebas unitarias deben arrojar resultados que servirán para valorar la correcta implementación de un elemento dentro del componente. Para la realización de estas pruebas el equipo de desarrollo se centró en monitorear el rendimiento de la memoria RAM y el microprocesador, ya que estos recursos son los más demandados por el comente desarrollado, arrojando como resultados:

1. Uso de la CPU 20.9 %.
2. Uso de la memoria RAM 31%

Estos resultados reflejan los recursos necesarios para poder realizar comparaciones en bases de datos de mediano tamaño.

3.5 Pruebas de caja blanca.

Las pruebas de caja blanca requieren del conocimiento de la estructura interna del programa y son derivadas a partir de las especificaciones internas de diseño o el código. Se examinan los caminos lógicos del software exponiendo que los casos de prueba se realizan juntos, definidos de condiciones y/o bucles. Se puede examinar el estado del programa en varios puntos para determinar si el estado real coincide con el esperado o mencionado.

Estas pruebas se basan en el diseño de casos de prueba que usa la estructura de control del diseño procedimental para derivarlos. Mediante la prueba de la caja blanca el ingeniero del software puede obtener casos de prueba que:

1. Garanticen que se ejerciten por lo menos una vez todos los caminos independientes de cada módulo, programa o método.
2. Ejerciten todas las decisiones lógicas en las vertientes verdadera y falsa.
3. Ejecuten todos los bucles en sus límites operacionales.

4. Ejerciten las estructuras internas de datos para asegurar su validez.

3.5.1 Prueba del camino básico.

Permite al diseñador de casos de prueba obtener una medida de la complejidad lógica de un diseño procedimental y usar esa medida como guía para la definición de un conjunto básico de caminos de ejecución. Los pasos que se siguen para aplicar esta técnica son:

1. A partir del diseño o del código fuente, se dibuja el grafo de flujo asociado.
2. Se calcula la complejidad ciclomática del grafo.
3. Se determina un conjunto básico de caminos independientes.
4. Se preparan los casos de prueba que obliguen a la ejecución de cada camino del conjunto básico.

La complejidad ciclomática de un grafo de flujo $V(G)$ establece el número de caminos independientes y se puede calcular de tres formas diferentes, las cuales se definen como:

1. $V(G) = A - N + 2$

Dónde: A es el número de aristas del grafo y N es el número de nodos.

2. $V(G) = P + 1$

Dónde: P es el número de nodos predicado contenido en el grafo G.

3. $V(G) = R$

Dónde: R es el número de regiones.

- Funcionalidad Comparar plantillas biométricas.

```
float Macher::Compare_Templates(Plantilla *p, Plantilla *s, int grades_liberty,int
radius_variation,int side_variation,int valid_grades)
{
 if(!p->IsStructFormed())//1
 p->FormarEstructura();//2
 if(!s->IsStructFormed())//3
 s->FormarEstructura();//4


 if(!p->IsTripletsFormed())//5
 p->FormarTripletas();//6
 if(!s->IsTripletsFormed())//7
 s->FormarTripletas();//8


 QList<ISO_SEXTUPLE> sextuples_p=p->GetSextuples();
 QList<ISO_SEXTUPLE> sextuples_s=s->GetSextuples();
```

```
 QList<ISO_TRIPLET> triplets_p;
 QList<ISO_TRIPLET> triplets_s;

 QList<ISO_TRIPLET> equals;//9

 for(int i=0;i<sextuples_p.length();i++)//10
 {
 QList<ISO_TRIPLET> temp=sextuples_p[i].tripletas;
 for(int j=0;j<temp.length();j++)//11
 {
 if(temp[j].IsValid(valid_grades))//12
 triplets_p.append(temp[j]);//13

 }
 }
 int Total_Tripletas_p=triplets_p.length();//14



 for(int i=0;i<sextuples_s.length();i++)//15
 {
 QList<ISO_TRIPLET> temp=sextuples_s[i].tripletas;
 for(int j=0;j<temp.length();j++)//16
 {
 if(temp[j].IsValid(valid_grades))//17
 triplets_s.append(temp[j]);//18

 }
 }

 for(int i=0;i<triplets_p.length();i++)//19
 {
 for(int j=0;j<triplets_s.length();j++)//20
 {

if(Compare_Triplets(triplets_p[i],triplets_s[j],grades_liberty,radius_variation,side_v
ariation))//21
 {
 equals.append(triplets_p[i]);
 triplets_p.removeAt(i);
 triplets_s.removeAt(j);
 i--;
 break;//22
 }
 }
 }
 int Total_Concidentes=equals.length();
 float porciento_coincidencia=((float)Total_Concidentes/
(float)Total_Tripletas_p)*100.0f;//23
```

```
return porciento_coincidencia;//24
 }
```

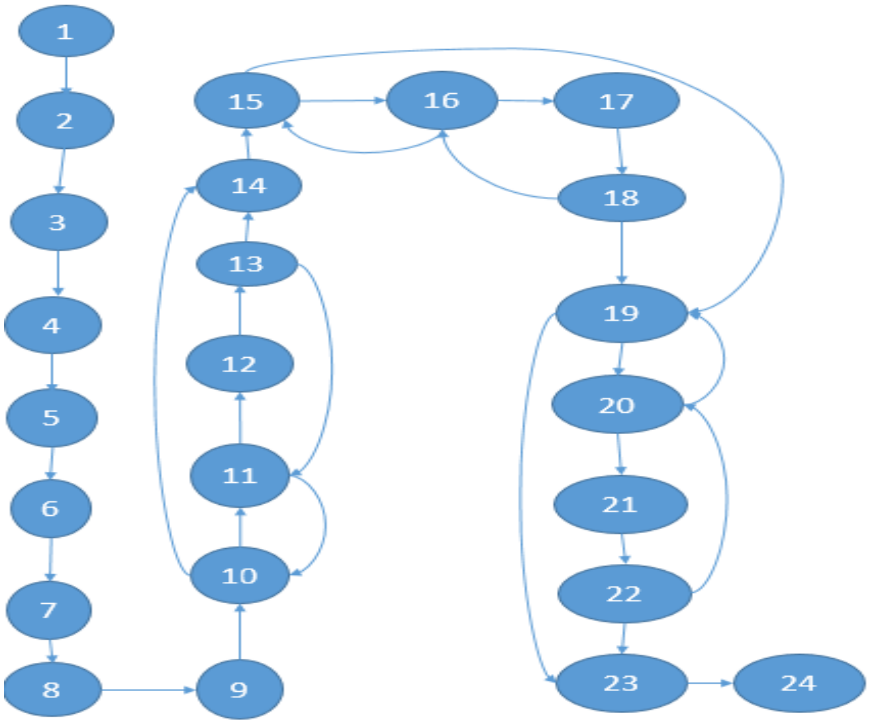

Figura 13: Grafo de Flujo: Funcionalidad comparar plantillas biométricas.

Caminos independientes obtenidos:

1. [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24]
2. [1,2,3,4,5,6,7,8,9,10,11,10,14,15,16,15,19,20,19,23,24]
3. [1,2,3,4,5,6,7,8,9,10,14,15,19,23,24]

Complejidad ciclomática para la funcionalidad comparar plantillas biométricas:

- **Fórmula 1**

V (G) = (A (Aristas) – N (Nodos)) + 2

V (G) = (32 – 24) + 2 = 10

- **Fórmula 2**

V (G) = P (Nodos Predicados) + 1

V (G) = 9+ 1 = 10

- **Fórmula 3**

V (G) = R = 10

3.6 Conclusiones parciales.

1. Las pautas de codificación definidas facilitaron el entendimiento del equipo de desarrollo durante la implementación del componente.
2. Las tareas de ingeniería posibilitaron una mejor organización durante el proceso de desarrollo del componente de comparación de plantillas de minucias de huellas dactilares.
3. La ejecución de pruebas al componente proporcionó datos de funcionamiento, corroborando el correcto funcionamiento del mismo.
4. Las pruebas de rendimiento biométrico realizadas al componente permitieron corroborar la disminución de las tasas de falso aceptado y falso rechazo del componente en comparación con las del componente de comparación presente en el motor de reconocimiento SourceAfis.

Conclusiones Generales.

Como resultado de la investigación realizada se concluye que:

- El análisis de los elementos teóricos asociados a la comparación de plantillas de minucias de huellas dactilares facilitó la definición de una propuesta de solución acorde a las necesidades existentes.
- La selección de AUP-UCI como metodología rectora del desarrollo del componente y el uso de las tecnologías y herramientas seleccionadas permitieron analizar, describir e implementar los procesos y subprocesos que debían ejecutarse.
- La implementación del algoritmo seleccionado facilitó los datos necesarios para realizar comparaciones con los algoritmos que se utilizan en el centro para la comparación de plantillas de minucias de huellas dactilares.
- Las pruebas realizadas al componente de comparación de plantillas de minucias de huellas dactilares implementado evidenciaron mejores resultados que el utilizado en el centro.

Recomendaciones.

1. Mejorar el algoritmo para la comparación cruzada de vectores de forma tal que no se realice una búsqueda intensiva.

Bibliografía

1. **LIC. LEÓN P, SUSAN K.** *AVANCES EN TÉCNICAS BIOMÉTRICAS Y SUS APLICACIONES EN SEGURIDAD.* Venezuela : s.n.

2. **Mata, Federico Bueno de.** *El uso de las TICs como medio de identificar presuntos autores de hechos delictivos.* España : s.n.

3. **Nebot, Ricardo Llopis.** *Sistemas de Autenticació Biometricos, Seguridad Y Protección De La Información.*

4. **Muñoz, Almudena Lindoso.** *Contribución al reconocimiento de huellas dactilares mediante técnicas de el aumento de prestaciones.* Leganés : s.n., Febrero 2009.

5. **Sanz, Gustabo Francisco.** *DESARROLLO DE UN SISTEMA DE RECONOCIMIENTO DE HUELLA DACTILAR PARA APLICACIONES MATCH-ON-CARD.* Madrid : s.n., julio,2009.

6. **Biometria,Reconocimiento de Huellas Dactilares.** ***Biometria,Reconocimiento de Huellas Dactilares.*** **[En línea] [Citado el: 26 de 1 de 2017.] http://www.biometria.gov.ar/metodos-biometricos/dactilar.aspx.**

7. Díaz, Marcos Martínez. ***VULNERABILIDADES EN SISTEMAS DE RECONOCIMIENTO BASADOS EN HUELLA DACTILAR: ATAQUES HILL-CLIMBING.*** **Madrid : s.n., Febrero 2007.**

8. SourceAFIS. ***SourceAFIS.*** **[En línea] 2017. [Citado el: 30 de 5 de 2017.] https://sourceafis.machinezoo.com/.**

9. Abel Ernesto Sanchez Ali, Ramón Santana Fernández. ***Pruebas realizadas a la librería de clases: SourceAfis.*** **Ciudad Habana : s.n.**

10. Abel Ernesto Sánchez Alí, Raúl Angel Ballester Mena. ***Componente para la Comparación de Huellas Dactilares en Tarjetas Inteligentes.*** **Ciudad de La Habana : s.n., Junio 2010.**

11. Davide Maltoni, Dario Maio,Anil k. Jain,Salil Prabhakar. ***Handbook of Fingerprint Recognition.*** **London : s.n., 2009.**

12. Laura Pelegrín Tumbeiro, Pedro Armando Rodríguez Fernández. ***Componente para la reconstrucción de imágenes de huellas dactilares.*** **La habana : s.n., 2012.**

13. ***Revista Cubana de Ciencias Informáticas.*** **Ramón Santana Fernández, Adrián A. Machado Cento,Vivian Estrada Sentí. No. 4, Boyeros,La Habana : s.n., 2015, Vol. Vol. 9.**

14. Miriela Velázquez Arias, Alexei Alayo Rondón. ***Componente de medición de la calidad de imágenes de huellas.*** **La habana : s.n., 2014.**

15. Arakala, Jason Jeffers and Arathi. ***MINUTIAE-BASED STRUCTURES FOR A FUZZY VAULT.*** **2006.**

16. Vigo, Jorge L. Aching Samatelo. David A. Rojas. ***ALGORITMOS PARA EL RECONOCIMIENTO DE IMÁGENES DE HUELLAS DACTILARES.*** **Lima-Peru : s.n., 2003.**

17. Isabel, José Ramón González. ***SISTEMA DE IDENTIFICACIÓN BIOMÉTRICA BASADO EN HUELLA DACTILAR MEDIANTE BINARIZACIÓN SOBRE PLATAFORMAS ANDROID.*** **España,Madrid : s.n., 2013.**

18. Juan López García, Departament d'Enginyeria Electrònica. ***Algoritmo para la identificación de personas basado en huellas dactilares.*** **Barcelona : s.n., 2009.**

19. Arakala, Jason Jeffers and Arathi. ***MINUTIAE-BASED STRUCTURES FOR A FUZZY VAULT.*** **2006 .**

20. Patricio Letelier Torres, Emilio A. Sánchez López. ***Metodologías Ágiles en el Desarrollo de Software.*** **España : s.n., 2003.**

21. Alejandro Martínez, Raúl Martínez. ***Guía a Rational Unified Process.*** **España : Escuela Politécnica Superior de Albacete – Universidad de Castilla la Mancha, 2014.**

22. Beymar Jimenez Ruiz, Sandra Peña Perz, Yamile Valdez Perez, Aracely J. Aramayo, Iver Salazar Zorrilla. ***METODOLOGIA DE DESARROLLO DE SOFTWARE-MSF-.*** **Bolivia : s.n., 2012.**

23. Ken Schwaber, Jeff Sutherland. ***La Guía Definitiva de Scrum.*** **2013.**

24. Informáticas, Universidad de las Ciencias. ***Metodología de desarrollo para la Actividad productiva de la UCI.*** **Carretera a San Antonio Km 2 1⁄2 . Torrens. Boyeros. La Habana. Cuba : s.n.**

25. Ingenieria de Software. Facilitador de Ingeniería de software. ***Ingenieria de Software. Facilitador de Ingeniería de software.*** **[En línea] [Citado el: 3 de 12 de 2016.] https://ingenieriasoftware2011.wordpress.com/2011/07/08/herramientas-case-ejemplos-para-evaluarlas/.**

26. ***Guión Visual Paradigm for UML.Ingeniería del Software. Curso 201 3-2014.*** **2013.**

27. Seco, José Antonio González. ***El lenguaje de programación C#.*** **2000.**

28. Arjona, Jorge L. Ortega. ***Notas de Introducción al Lenguaje de.*** **2004.**

29. Jaime Conde Rojas, Josse Luis Gutierrez Herrera,Liliana Vicenteño Herrera. ***Curso de lenguaje de programacion java.*** **2007.**

30. Rojas, Alan D. Osorio. ***C++ Manual teórico-práctico.*** **2006.**

31. Cabeza, José Luis Comesaña. ***INSTALACIÓN Y USO DE ENTORNOS DE DESARROLLO .*** **2011-2012.**

32. Informatica, De pt. ***Ec lipse (2.1) y Java.*** **Valencia, España : s.n., 2004.**

33. Eclipse IDE. [En línea] [Citado el: 4 de 12 de 2016.] http://www.asociacionaepi.es/eclipse-ide/?print=pdf.

34. Romero, José Rubén Sáez. ***"Diseño y desarrollo de una aplicación para operadora de Telecomunicaciones para monitorizar la conexión a Internet.".*** **Valencia, Campus Gandia : s.n., 2016.**

35. ***Procesamiento de Imágenes Máster NTI,Guión de prácticas.***

36. Reynoso, Carlos Billy. ***Introducción a la Arquitectura de Software.*** **Buenos Aires,Chile : s.n., 2004.**

37. Santiago Domingo Moquillaza Henríquez, Hugo Vega Huerta,Luis Guerra Grados. ***Programación en N capas.*** **España : s.n., 2010.**

38. Tello, Jesús Cáceres. ***Patrones de diseño: ejemplo de aplicación en los Generative Learning Object.*** **Alcalá,España : s.n., 2008.**

39. JUAN, FRANCISCO JAVIER MARTÍNEZ. ***GUÍA DE CONSTRUCCIÓN DE SOFTWARE EN JAVA CON PATRONES DE DISEÑO.*** **España : s.n.**

40. Vidal, Mari Carmen Otero. ***Dpto. de Lenguajes y Sistemas Informáticos.***

41. G. Aguilar-Torres*, G. Sánchez-Pérez, Toscano-Medina, H. Pérez-Meana. ***Fingerprint Recognition Using Local Features and Hu Moments.*** **Mexico : ESIME Culhuacan, Instituto Politécnico Nacional, 2012.**

42. Reséndiz, Lic. Rosendo Ibáñez. ***LA HUELLA DIGITAL Y EL DERECHO MEXICANO.*** **Mexico : s.n.**

43. tecnología(NSTC), Consejo Nacional de Ciencia y. ***Reconocimiento de Huellas Dactilares.*** **Estados Unidos : s.n., 2006.**