Facultad 3

Centro de Diseño y Simulación de Estructuras Mecánicas

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

---

Título: **Herramienta de diseño conceptual basado en bocetos a mano alzada para el *software* GALBA-CAD**

Autor: Anicel Mejías Rodríguez

Tutor: Ing. Gustavo Toranzo Lorca

La Habana, junio del 2014

"No hay más que asomarse a las puertas de la tecnología y la ciencia contemporánea para preguntarnos si es posible vivir y conocer ese mundo futuro sin un enorme caudal de preparación y conocimientos".

Fidel Castro Ruz.

DECLARACIÓN DE AUTORÍA

Declaro que soy el único autor de este trabajo de diploma y autorizo al Centro de Diseño y Simulación de Estructuras Mecánicas (CDSEM) de la Universidad de las Ciencias Informáticas (UCI) a hacer uso del mismo en su beneficio.

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

Anicel Mejías Rodríguez	Ing. Gustavo Toranzo Lorca
_____	_____
Firma del Autor	Firma del Tutor

DATOS DE CONTACTO

Autor

Anicel Mejías Rodríguez.

Universidad de las Ciencias Informáticas, Ciudad de la Habana, Cuba

Correo: amrguez@estudiantes.uci.cu

Tutor

Ing. Gustavo Toranzo Lorca.

Universidad de las Ciencias Informáticas, Ciudad de la Habana, Cuba

Correo: gtoranzo@uci.cu

AGRADECIMIENTOS

Agradecer a mi familia por estar siempre a mi lado, en especial a mis padres, a ellos le debo todo lo que soy. A mi madre por todo el esfuerzo realizado en todos estos años que he estado lejos, por el amor y la confianza que siempre ha tenido en su hijo. A mi papá que siempre ha sido mi ejemplo a seguir y del cual siempre he estado orgulloso.

Agradecer a mi tutor Gustavo Toranzo Lorca que más que tutor ha sido un amigo, por su comprensión, paciencia y por ofrecerme su ayuda, sin él no hubiera sido posible la realización de este trabajo.

A todos los profesores que de una forma u otra ayudaron a mi formación.

A mis compañeros de aula por haberme aguantado cinco años, ustedes saben cuánto jodo.

A todas mis amistades que a lo largo de mi carrera me han brindado su amistad incondicional y han estado ahí para lo que fuera necesario.

DEDICATORIA

Quiero dedicar esta tesis de forma especial a mis padres que gracias a su esfuerzo y sacrificio hoy estoy aquí.

A toda mi familia y amigos que me apoyaron en todo momento.

RESUMEN

El diseño asistido por computadoras abarca un amplio rango de herramientas computacionales que asisten a ingenieros, arquitectos y a otros profesionales del diseño en sus respectivas actividades. Precisamente, en el Centro de Diseño y Simulación de Estructuras Mecánicas (CDSEM) de la Universidad de las Ciencias Informáticas (UCI), se desarrolla un proyecto de *software* CAD para el diseño y modelación de estructuras mecánicas. Este *software*, denominado GALBA-CAD, tiene como objetivo responder a las necesidades de la Industria China-Venezolana de Taladros (ICVT) pertenecientes a PDVSA Industrial de la República Bolivariana de Venezuela.

Los clientes de este sistema plantean la necesidad de contar con dispositivos como *smartphones* (teléfonos inteligentes), *tablets* (tabletas electrónicas), que permitan a sus especialistas realizar bocetos en cualquier momento, y de esta manera contribuir a la gestión de las labores diarias de una manera más fácil e interactiva.

Con el objetivo de solucionar los problemas diarios en el campo del diseño, se decide la creación de una herramienta de diseño conceptual basado en bocetos a mano alzada para móviles con sistema operativo Android que exporte los bocetos creados en un formato compatible con el *software* GALBA-CAD.

En el presente trabajo se hace un análisis sobre las tecnologías CAD/CAE/CAM y herramientas CAD en aras de identificar puntos de reutilización. Se definen un conjunto de artefactos descritos en la metodología seleccionada y se valida la propuesta a fin de demostrar el objetivo de la investigación.

Palabras claves: bocetos; tecnologías CAD/CAE/CAM; herramientas CAD.

ABSTRACT

The computer-aided design represents a wide range of computational tools that assist engineers, architects and other design professionals in their respective activities. At the Center for Design and Simulation of Mechanical Structures (CDSEM) of the University of Informatics Sciences (UCI), is developed a project of CAD software for the design and modeling of mechanical structures. This software, called GALBA-CAD, aims to meet the needs of the Chinese-Venezuelan industry of drills (ICVT) belonging to PDVSA Industrial of the Bolivarian Republic of Venezuela.

Customers of this system raises the need for having devices such as smartphones, tablets, that allow their specialists make sketches at any time, and thus contribute to the management of the daily works of more easy and interactive way.

In order to solve everyday problems in the field of design is decided creating a conceptual design tool based on freehand sketches for Android systems that export the sketches in GALBA-CAD format.

In the present work, an analysis is made of the CAD/CAE/CAM technologies and CAD tools in order to identify points of reuse. A set of artifacts described are defined in current methodology and the proposed is validated in order to demonstrate the purpose of the research.

***Keywords**: sketches; CAD/CAE/CAM technologies; CAD tools.*

ÍNDICE DE CONTENIDOS

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

INTRODUCCIÓN

Durante la última década, los dispositivos móviles (teléfonos inteligentes y tabletas) han adquirido un gran protagonismo al grado de que ya la gran mayoría involucran pantallas táctiles y son cada vez más utilizados en el campo de la tecnología para realizar cualquier tipo de proyecto. Se han convertido en equipos robustos para su utilización en la gestión de situaciones empresariales, como por ejemplo, para la grabación de información estando en cualquier parte. Actualmente son usados para archivar una variedad de tareas y para incrementar la eficiencia, como son: la digitalización de notas, gestión de archivos, capturas de firmas, gestión y escaneo de partes de código de barras. Cada uno presenta ciertas ventajas como el tipo de aplicación que se puede instalar, el tipo de sistema operativo (SO) y el tamaño. En la actualidad, el SO Android es uno de los más utilizados debido a que es de código abierto y tiene licencia libre. Un ejemplo de las tecnologías empleadas por los dispositivos móviles, lo constituyen las tecnologías CAD/CAE/CAM[1], las cuales se pueden interpretar como las "Tecnologías de diseño, ingeniería y fabricación, asistidos por computadoras". El CAD atiende prioritariamente aquellas tareas exclusivas del diseño, tales como el dibujo técnico y la documentación del mismo (1). El término CAE engloba el conjunto de herramientas informáticas que permiten analizar y simular el comportamiento del producto diseñado, y el término CAM se refiere al uso de computadoras para ayudar en todas las fases de la fabricación de un producto, incluyendo la planificación del proceso y la producción, calendarización, administración y control de calidad, con una intervención mínima del operario (2).

Cuba no queda exenta de estos avances tecnológicos y del impacto que representa su introducción en la industria cubana. Por tal motivo en el Centro de Diseño y Simulación de Estructuras Mecánicas (CDSEM) de la Universidad de las Ciencias Informáticas (UCI) se desarrolla un proyecto de herramientas CAD[2] para el diseño y modelación de estructuras mecánicas. Este *software*, denominado GALBA-CAD, tiene como objetivo responder a las necesidades de la Industria China-Venezolana de Taladros (ICVT) pertenecientes a PDVSA[3] Industrial de la República Bolivariana de Venezuela. El sistema consta actualmente con dos módulos principales. El primero es el módulo de creación de los bocetos paramétricos 2D y el segundo es el módulo de creación de partes.

Un boceto es una colección de geometrías (líneas, puntos y arcos) aparejadas con relaciones (parámetros, ecuaciones, dimensiones, restricciones y construcciones geométricas) establecidas en 2D. Estos elementos geométricos son relacionados para reflejar la intensión del diseño. El módulo de creación de bocetos es la entrada para la ejecución del módulo de creación de partes. Este último a

[1] Computer Aided Design/ Computer Aided Engineering/ Computer Aided Manufacturing, por sus siglas en inglés.
[2] Computer Aided Design, por sus siglas en inglés.
[3] Petróleos de Venezuela S.A.

su vez es usado para definir geometrías en 3D y depende para su ejecución de la creación de caras a partir del boceto realizado.

La mayoría de los usuario de *software* de diseño conceptual, opinan que las interfaces CAD no son apropiadas para realizar bocetos de las ideas más básicas. GALBA-CAD cuenta con una interfaz gráfica convencional, que comparte las mismas características fundamentales que otros sistemas de diseño (CAD) líderes en el mundo como son Autodesk Inventor y AutoCAD.

Los actuales sistemas gráficos pueden ser operados por íconos, barra de herramientas y menús. Aunque las dimensiones y la información geométrica no son esenciales en el diseño conceptual, los actuales sistemas CAD requieren que estas sean brindadas por el usuario, lo cual interrumpe el flujo natural de las ideas (1). En la ICVT son utilizadas interfaces CAD convencionales que solo permiten la creación de bocetos en estaciones de trabajo *Desktop*. Los especialistas de la ICVT cuando están en el campo no pueden utilizar las PC de escritorio compuestas por los *software* CAD, por lo tanto no pueden crear bocetos geométricos conceptuales digitales. Sin embargo los especialistas actualmente utilizan lápiz y papel para la creación de los bocetos, esto trae consigo que luego deben crear nuevamente los bocetos en las estaciones de trabajo *Desktop* con lo cual se duplica el trabajo. La creación de bocetos utilizando lápiz y papel tiene como desventaja que resulta engorroso la eliminación de trazos erróneos. Los especialistas regularmente encuentran el inconveniente de que al trabajar en lugares con grasa el papel fácilmente se deteriora. No cuentan con los bocetos almacenados en la PC de trabajo para ser utilizados como punto de referencia en el diseño de nuevos bocetos en el campo.

Por otra parte los clientes plantean la necesidad de contar con dispositivos como *smartphones* (teléfonos inteligentes) o *tablets* (tabletas electrónicas), que permitan a sus especialistas realizar bocetos en cualquier momento, y de esta forma contribuir a la gestión de las labores diarias de un modo más fácil e interactivo.

Dada la situación antes expuesta, se formula el siguiente **problema a resolver**: ¿Cómo permitir a los especialistas de la ICVT desarrollar bocetos geométricos conceptuales a mano alzada en el campo y en las estaciones de trabajo?

En aras de resolver el problema planteado se define como **objeto de estudio**: Las aplicaciones CAD.

Para dar solución al problema enunciado se propone como **objetivo general**: Desarrollar una aplicación Android para la creación de bocetos geométricos conceptuales a mano alzada en el campo y en las estaciones de trabajo que se integre con el *software* GALBA-CAD.

Este objetivo general ha sido desglosado en los siguientes **objetivos específicos**:

- Realizar un análisis del estado del arte de las aplicaciones CAD para la creación de bocetos 2D;
- Realizar el análisis y diseño de la solución propuesta;
- Implementar una aplicación Android para el diseño conceptual a mano alzada;
- Desarrollar pruebas de caja negra para la validación de la solución.

Todo lo anterior precisa como **campo de acción**: Aplicaciones CAD para la creación de bocetos 2D que se integre con GALBA-CAD.

Con el desarrollo de una aplicación para la creación de bocetos a mano alzada para ser integrados al *software* GALBA-CAD, los especialistas de la ICVT podrán elaborar bocetos en el campo y en las estaciones de trabajo.

El documento está estructurado en tres capítulos que se describen a continuación:

El **Capítulo 1, Fundamentación teórica,** presenta las tendencias y tecnologías actuales sobre las cuales se apoya la propuesta del sistema informático. Se describen la metodología de desarrollo de *software*, así como las herramientas y lenguajes a emplear en el desarrollo de la solución.

El **Capítulo 2, Propuesta de solución,** incluye la descripción, diseño y análisis de la solución que se propone para darle respuesta a la problemática planteada. Se especifican los requisitos funcionales y los no funcionales. Se realiza el modelo de CU y describen los diagramas de clases. Se describen los estilos arquitectónicos y patrones de diseño aplicados.

El **Capítulo 3, Implementación y pruebas,** muestra la implementación de la funcionalidad. En él se analizan los estilos de codificación y se brinda una solución a los requisitos especificados. Se describen los diagramas de componentes, se muestra el código fuente de las principales clases y se aplican pruebas al sistema para demostrar la robustez del mismo.

CAPÍTULO 1. FUNDAMENTACIÓN TEÓRICA

En el actual capítulo se presenta el marco teórico de la investigación, en el mismo se definen los principales conceptos relacionados con las tecnologías CAD/CAE/CAM, el diseño de bocetos 2D y el SO Android, que constituyen el eslabón primario para la realización de la solución. Además, se hace alusión a las tendencias actuales del uso de las tecnologías móviles, así como un estudio de las herramientas, lenguajes y tecnologías utilizadas.

1.1 Marco conceptual

1.1.1 Definición de "Boceto"

Boceto. Del italiano bozzetto, es el esquema o proyecto en que se bosqueja cualquier obra. Se trata de un apunte general previo a la ejecución. Por lo general, un boceto (conocido como *layout* en inglés) es un dibujo esquemático que no incluye detalles ni terminaciones (14).

El boceto se encuentra clasificado en tres formas: Burdo, Comprensivo y *Dummy*.

Burdo: Es la primera idea que se visualiza en la mente y se dibuja a mano sobre cualquier papel o dispositivo. Su finalidad es plasmar las ideas que se tienen acerca del trabajo a desarrollar.

Comprensivo: Aquí las ideas se van ajustando para hacerlas más comprensibles y de mayor calidad, se utilizan para la elaboración de instrumentos técnicos para delimitar los espacios.

Dummy: Boceto de arte final que alcanza un alto nivel de calidad y composición mecánica de todos los elementos visuales que se usarán en la reproducción final (14).

1.1.2 Definición de "Diseño"

El diseño es el proceso previo de configuración mental en la búsqueda de una solución. Consiste en una visión representada en forma gráfica de una obra futura. El acto intuitivo de diseñar podría llamarse creatividad como acto de creación o innovación si el objeto no existe, o es una modificación de lo existente, inspiración, abstracción, síntesis, ordenación y transformación. Diseñar requiere principalmente consideraciones funcionales y estéticas (15).

Aunque existe gran variedad de criterios en cuanto a la definición de diseño, el autor se identifica con el citado anteriormente, teniendo en cuenta el enfoque y similitud con la problemática tratada.

1.1.3 Diseño 2D

El diseño 2D es un conjunto de objetos de dibujos que contienen solamente la geometría en dos dimensiones, son objetos planos que permiten la visualización. Se trabaja en los ejes (X) y (Y) de coordenadas (16).

1.1.4 Elementos del diseño gráfico

El diseño gráfico abarca cuatro grupos de elementos (17):

- Elementos conceptuales;
- Elementos visuales;
- Elementos de relación;
- Elementos prácticos.

Los **elementos conceptuales**, aunque no son observables a la vista, están presentes en el diseño y se dividen en cuatro:
Punto: Indica posición, no tiene largo ni ancho. Representa el principio y fin de una línea.
Línea: Es una sucesión de puntos, tiene largo pero no ancho. Contiene una posición y una dirección.
Plano: Tiene largo, ancho, posición y dirección. Está limitado por líneas.
Volumen: Representa el recorrido de un plano en movimiento. Tiene posición en el espacio y está limitado por planos. En un diseño bidimensional el volumen es ilusorio.

Los **elementos visuales** se componen como sigue:
Forma: Aporta para la percepción del ojo humano una identificación del objeto en cuestión.
Medida: Define el tamaño de cada objeto, representado en una unidad de medida.
Color: Define los colores de cada uno de los elementos que componen el diseño.
Textura: Representa el tipo de superficie en cada diseño de acuerdo a la utilización del material.

Los **elementos de relación** definen la ubicación y dependencia de las formas:
Dirección: Define el sentido de una forma. Depende de cómo está relacionada con el observador, con el marco que la contiene y con otras formas cercanas.
Posición: Depende del elemento o estructura que la contenga.
Espacio: Define, como su nombre indica, el espacio que ocupa cada forma. Este espacio puede ser o no visible.
Gravedad: Permite la definición del efecto de gravedad. Atribuye estabilidad o inestabilidad a una forma, a un grupo de ellas.

Los **elementos prácticos** se describen a continuación**:**
Representación: Define la forma de realizar el diseño. Puede ser una representación realista, estilizada o semi-abstracta.
Significado: Descripción de los elementos que componen el diseño.
Función: Responsabilidad u objetivo de cada diseño.
Con el uso del diseño en 2D, se podrá representar o dibujar los planos en dos dimensiones. Posibilitando a los especialistas crear planos y que estos a su vez contengan puntos, líneas, arcos, círculos, textos y dimensiones.

1.2 Tecnologías CAD/CAE/CAM

La aparición de los sistemas CAD/CAE/CAM en los procesos industriales, ha propiciado la disminución y corrección de errores en los diseños de fabricación. Esta tecnología abarca el diseño gráfico, el manejo de bases de datos para el diseño y la fabricación, control numérico de máquinas, herramientas, robótica y visión computarizada. Actualmente estos sistemas están conectados a los sistemas de gestión y producción de tal forma que ya desde la fase de diseño se puede saber el coste del producto final y controlar los *stocks*[4] de componentes y materiales para su fabricación.

El diseño asistido por computadora es el uso de programas informáticos para crear representaciones gráficas de los objetos físicos en dos o tres dimensiones. El *software* CAD puede ser especializado para aplicaciones específicas. El diseño asistido es ampliamente utilizado en la animación por ordenador y en los efectos especiales en películas, publicidad, y otras aplicaciones donde el diseño gráfico en sí es el producto final. También se utiliza para diseñar productos físicos en una amplia gama de industrias, donde el *software* realiza los cálculos para determinar la forma y tamaño óptimos para una variedad de productos y aplicaciones de diseño industrial. En el producto y el diseño industrial, CAD se utiliza principalmente para la creación de modelos 3D detallados de sólidos o superficie, o dibujos en 2D basados en vectores de los componentes físicos. Sin embargo, CAD también se utiliza en todo el proceso de ingeniería desde el diseño conceptual y el diseño de productos, a través de la potencia y el análisis dinámico de los ensamblajes, hasta la definición de los métodos de fabricación. Esto permite que un ingeniero tanto de forma interactiva como automática analizar variantes de diseño, para encontrar el diseño óptimo para la fabricación y reducir al mínimo el uso de prototipos físicos (4).

Ingeniería asistida por computadora es el uso de *software* computacional para simular desempeño y así poder hacer mejoras a los diseños de productos o bien apoyar a la resolución de problemas de ingeniería para una amplia gama de industrias. Esto incluye la simulación, validación y optimización de productos, procesos y herramientas de manufactura. Un proceso típico de CAE incluyen pasos de pre-procesado, solución y post-procesado. En la fase de pre-procesado, los ingenieros modelan la geometría y las propiedades físicas del diseño, así como el ambiente en forma de cargas y restricciones aplicadas. En la fase de post-procesado, los resultados se presentan al ingeniero para su revisión (5).

La tecnología CAM hace referencia a los sistemas informáticos que ayudan a generar los programas de control numérico para la fabricación de piezas en máquinas con control numérico computarizado. A partir de la información de la geometría de la pieza, del tipo de operación deseada, de la herramienta escogida y de las condiciones de corte definidas, el sistema calcula las trayectorias de la herramienta

[4] Componentes que permanecen almacenados para su posterior utilización.

para conseguir el mecanizado correcto, y a través de un post-procesado genera los correspondientes programas de control numérico con la codificación específica del control numérico computarizado. En general, la información geométrica de la pieza proviene de un sistema CAD, que puede estar o no integrado con el sistema CAM (3).

Después del estudio realizado se puede concluir que el uso de tecnologías CAD, contribuyen a la mejora, desarrollo y diseño de los productos con la ayuda de la computadora. El uso de este proceso favorece la reducción de costos de mano de obra, la disminución de errores humanos y posibilita mayor rapidez en el proceso de fabricación. A continuación se explican los tipos de CAD existentes:

1.2.1 Tipos de CAD

CAD Analítico: Usa procedimientos analíticos para definir sus límites o acciones. Los programas del tipo CAD analíticos, surgen para cuantificar y permitir evaluar los resultados de las variables que involucra el diseño estructural. En los CAD analíticos el dibujo o trazado permanece en la memoria de la computadora como una serie de relaciones de puntos-coordenadas, sentido y dirección en programas vectoriales o como un grupo de píxeles, en programas de renderizado y tratamiento de imágenes.

CAD Paramétrico: Usa parámetros para definir sus límites o acciones. En un programa paramétrico la información visual es parte de la información disponible en el banco de datos, representa la información como un objeto en la memoria de la computadora (6).

1.2.2 Ventajas del uso del CAD

Los arquitectos rara vez diseñan en una mesa de dibujo, en la actualidad se explotan las potencialidades de las herramientas CAD. Las principales ventajas que provee el uso de estas tecnologías son:

- Es posible utilizar librerías de elementos comunes;
- Se elimina la distinción entre plano original y copia;
- El almacenamiento de los planos es más reducido, fiable (tomando ciertas medidas de seguridad) y permite realizar búsquedas rápidas y precisas mediante bases de datos;
- Aumenta la uniformidad en los planos;
- La calidad de los planos es mayor(no hay tachones, ni líneas más gruesas que otras);
- El tiempo invertido en las modificaciones se reduce enormemente;
- Reducción del tiempo empleado en operaciones repetitivas;

- Los datos pueden exportarse a otros programas para obtener cálculos, realizar informes, presentaciones;
- Se puede obtener un modelo en 3D para visualizarlo desde cualquier punto de vista;
- Pueden exportarse los datos a programas de CAE;
- Facilitan el trabajo en equipo (7).

1.3 Herramientas CAD

Luego de una revisión bibliográfica sobre las herramientas CAD, se demostró que las mismas están organizadas de acuerdo a los sistemas operativos soportados, en aras de lograr un mejor entendimiento del lector se decidió caracterizar dichas herramientas de acuerdo al SO sobre el cual se ejecuta.

1.3.1 Herramientas CAD para Windows

Autodesk AutoCAD: Es una plataforma de diseño asistido por computadora para dibujo en dos y tres dimensiones. Actualmente es desarrollado y comercializado por la empresa Autodesk. Es un *software* reconocido a nivel internacional por sus amplias capacidades de edición, que hacen posible el dibujo digital de planos de edificios o la recreación de imágenes en 3D. Es uno de los programas más usados, elegido por arquitectos, ingenieros y diseñadores industriales. El programa gestiona una base de datos de entidades geométricas (puntos, líneas, arcos) con la que se puede operar a través de una pantalla gráfica en la que dichas entidades son mostradas, el llamado editor de dibujo o modelador geométrico. Además, procesa imágenes de tipo vectorial, aunque permite incorporar archivos de tipo fotográfico o mapa de bits, donde se dibujan figuras básicas o primitivas (líneas, arcos, rectángulos, textos), y mediante herramientas de edición se crean gráficos más complejos (8).

Autodesk® Inventor®: Ofrece una gama completa y flexible de programas para diseño mecánico en 3D, simulación de productos, creación de herramientas, ingeniería a la carta y comunicación de diseños. Inventor lleva más allá de la tercera dimensión hacia prototipos digitales ya que permite producir un modelo 3D de gran precisión que puede ayudarlo a diseñar, visualizar y simular los productos antes de ser construidos. La creación de prototipos digitales con Inventor contribuye a que las compañías puedan diseñar mejores productos, reducir los costos de desarrollo y llegar al mercado más rápido (9).

Ashampoo 3D CAD Architecture**:** Es una herramienta potente de diseño 2D y 3D, que cuenta con propiedades muy interesantes como son: componentes inteligentes, diseño de planos en 2D/3D, exportación a imágenes de realidad virtual, y cálculo de materiales y costo para una determinada construcción. Posee características como: un asistente de inicio y asistente de proyectos, construcción

libre de objetos en 3D, editor de superficies para el diseño individual de superficies, extensas funciones 2D, ayudas a la construcción y entradas, vistas 2D y 3D (también en paralelo) y extensos catálogos con objetos, materiales, texturas y símbolos (10).

1.3.2 Herramientas CAD para GNU/Linux

LibreCAD: Es una propuesta gratuita de código abierto, que brinda las herramientas básicas necesarias para el trabajo, sin perderse en las complejas y sofisticadas funciones de toda herramienta CAD. Ofrece una herramienta de CAD 2D, derivada de QCad. El proyecto ha modificado algunas partes del código original para resolver problemas de licencias y se distribuye bajo GPL[5]v2. Cuenta con una interfaz de usuario sencilla basada en las bibliotecas de Qt4[6]. Dispone de un panel de herramientas dinámico muy completo, que esconde muchas más funciones de las que pueden apreciarse a simple vista. Tiene soporte para capas, funcionalidad esencial en este tipo de *software*. LibreCAD solo aborda el dibujo 2D, es compatible con ficheros DXF[7], y CXF, pero no soporta ficheros DWG[8]. Está disponible también para *Microsoft Windows*, Mac OS X y algunas de las principales distribuciones de GNU/Linux (Debian, Ubuntu, Fedora, Mandriva, Suse, entre otras). El programa está traducido a 20 idiomas, entre ellos el español, aunque parcialmente (11).

QCad: Aplicación gratuita, enfocada al dibujo en dos dimensiones. Mediante ella se pueden crear vistas de piezas, planos de edificios y esquemas. Soporta capas de vistas, bloques, diversas fuentes de texto, unidades métricas e imperiales, impresión a escala, herramientas de medición, bibliotecas con piezas predefinidas, y herramientas para construcción y modificación de puntos, líneas, círculos, elipses, polilíneas, textos, dimensiones, sombreados, rellenos e imágenes de trama (12). Igualmente es multiplataforma, por lo que también corre bajo los sistemas operativos *Windows* y MAC OS X.

DraftSight: Es una especie de clon de AutoCAD con características interesantes, gratuito, pero tiene la desventaja de que solo trabaja en 2D. Está enfocado a estudiantes y profesionales y permite crear y editar archivos en formato AutoCAD (.dwg). La interfaz del programa es altamente funcional. Las herramientas están bien distribuidas, y se maximiza el área de dibujo mediante el empleo de desplegables, navegación por pestañas en algunas secciones y menús que pueden ocultarse (13). Maneja las entidades básicas 2D como: arcos, círculos y líneas; permite insertar notas, y manipular las

[5]General Public License, por sus siglas en inglés.
[6] Biblioteca multiplataforma para desarrollar aplicaciones con o sin interfaces gráficas, así como herramientas para la línea de comandos y consolas para servidores.
[7]Drawing Exchange Format, por sus siglas en inglés. Es un formato de archivo informático para dibujos de diseño asistido por computadora,
[8] Se originó de la palabra inglesa "*drawing*". es un formato de archivo informático de dibujo computarizado, utilizado principalmente por el programa AutoCAD; producto de la compañía AutoDesk.

dimensiones de las entidades (longitud del arco, radio, diámetro, línea base) Es multiplataforma, por lo que también corre bajo los sistemas operativos *Windows* y MAC OS X.

1.3.3 Resultados

El estudio de las herramientas CAD para *Windows* y Linux contribuyó decisivamente a una aproximación inicial sobre el diseño del sistema. Estas herramientas aportaron diversos conocimientos que se pondrán en práctica en la solución propuesta. Entre ellos, la necesidad de procesar individualmente a cada entidad geométrica teniendo en cuenta su diversidad, las dimensiones a tratar de cada entidad, así como la necesidad de un mecanismo para la construcción de las mismas.

Aunque el estudio de estas herramientas permitió adquirir un mejor dominio de la problemática abordada, se decidió desarrollar una herramienta para la elaboración de bocetos geométricos conceptuales a mano alzada. Teniendo en cuenta, para el desarrollo, la necesidad de ajustarse al uso e integración de las tecnologías de desarrollo empleadas en el proyecto GALBA-CAD, y el uso del SO Android, para de esta forma contribuir a la independencia tecnológica del país.

1.4 Integración con GALBA-CAD

La herramienta FreeHand garantizara la integración con el *software* GALBA-CAD, este *software* implementa la funcionalidad de cargar y guardar los bocetos en un fichero con extensión skt, por lo que FreeHand debe ser capaz de guardar y cargar los ficheros con la extensión antes mencionada.
Los ficheros de intercambio skt son ficheros de texto plano ASCII con codificación ANSI. Cada entidad es descrita por una sola línea. Las entidades se encuentran agrupadas por tipo, antes de ser descritas un grupo de entidades del mismo tipo se almacena un valor que representa la cantidad de entidades de ese tipo que serán descritas en las líneas siguientes. Los grupos de entidades son almacenados en el siguiente orden: puntos, líneas, arcos de círculos, círculos y elipses.
Para un punto se almacenan las coordenadas x, y, z separadas por un espacio en una línea. Para una línea se guardan los índices de los dos puntos extremos, el índice de un punto se corresponde con su posición en la lista de puntos. Para un arco se almacena el índice del punto centro y los índices de los dos puntos extremos. Para un círculo se guarda el índice del punto centro y el valor del radio. Para una elipse se almacena el índice del punto centro, el valor del radio mayor, el valor del radio menor y el valor del ángulo que forma el mayor radio con el eje horizontal en radianes.
La **Figura 1** muestra un boceto con siete vértices, una línea, un arco, cero círculos y una elipse.

Sketch.skt
7 3.077978 1.420541 0 -3.116994 -2.649644 0 -2.676631 1.448154 0

```
-2.527104 -0.6404747 0
-5.476917 1.996998 0
-6.742342 -1.285197 0
0.8502074 -3.32174 0
1
4 5
1
3 1 2
0
1
0 4.33284330368042 1.47521245479584 2.43798780441284
```

Figura 1 Fichero con formato skt

1.5 Sistema operativo Android

Luego de realizar un estudio de sistemas operativos para dispositivos móviles se seleccionó el SO Android, debido a que es una distribución libre y de código abierto. Además dispone de una plataforma adaptable a pantallas grandes, gráficos 2D y 3D.

Android, es un sistema operativo además de una plataforma de *software* basada en el núcleo de Linux. Diseñada en un principio para dispositivos móviles. Android permite controlar dispositivos por medio de bibliotecas desarrolladas o adaptadas por Google mediante el lenguaje de programación Java (18).

1.5.1 Arquitectura de Android

La arquitectura interna de la plataforma Android, está básicamente formada por 4 componentes:

Aplicaciones: Todas las aplicaciones creadas con la plataforma Android, incluyen como base un cliente de correo electrónico, calendario, programa de mensajería instantánea (SMS), mapas, navegador, contactos, y otros servicios mínimos. Todas ellas escritas en el lenguaje de programación Java.

Framework de Aplicaciones: Todos los desarrolladores de aplicaciones Android, tienen acceso total al código fuente usado en las aplicaciones base. Ha sido diseñado de esta forma, para que no se generen cientos de componentes de aplicaciones distintas, que respondan a la misma acción, dando la posibilidad de que los programas sean modificados o reemplazados por cualquier usuario sin tener que empezar a programar sus aplicaciones desde el principio.

Librerías: Android incluye en su base de datos un set de librerías C/C++, que son expuestas a todos los desarrolladores a través del *framework* de las aplicaciones *Android System C library*, librerías de medios, librerías de gráficos, 3D, SQLite.

Runtime de Android: Android incorpora un set de librerías que aportan la mayor parte de las funcionalidades disponibles en las librerías base del lenguaje de programación Java. La Máquina Virtual está basada en registros, y corre clases compiladas por el compilador de Java que anteriormente han sido transformadas al formato .dex (*DalvikExecutable*) por la herramienta dx (19).

1.6 Metodología de desarrollo de *software*

Una metodología de desarrollo es un conjunto de procedimientos, técnicas, herramientas y soporte documental que sirve para guiar el proceso de desarrollo de *software*. Se basan en uno de los modelos de procesos o una combinación de ellos. Puede seguir uno o varios modelos de ciclo de vida, es decir, el ciclo de vida indica qué es lo que hay que obtener a lo largo del desarrollo del proyecto (20).

Teniendo en cuenta que la siguiente investigación, forma parte del conjunto de aplicaciones desarrolladas por el Centro de Diseño y Simulación de Estructuras Mecánicas (CDSEM), fue necesario asumir la metodología, herramientas y tecnologías definidas. Teniendo en cuenta, además, la necesidad de integración con el *software* GALBA-CAD.

El centro, para guiar el ciclo completo de desarrollo de *software*, emplea la metodología Proceso Unificado de Desarrollo (RUP[9]), generando solamente los siguientes artefactos:

- Modelo de dominio, requisitos funcionales, requisitos no funcionales, modelo de casos de uso (CU), descripción de CU, diagrama de clases, diagrama de despliegue y diagrama de componentes.

A continuación se procede a la caracterización de las herramientas y tecnologías usadas en el desarrollo de la aplicación que se presenta.

1.6.1 Proceso Unificado de Desarrollo (RUP)

Proceso Unificado de Desarrollo (RUP) (21): Es una metodología de desarrollo de *software* que está basada en componentes e interfaces bien definidas, y junto con el Lenguaje Unificado de Modelado (UML[10]), constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos. Es un proceso que puede especializarse para una gran variedad de sistemas de *software*, ya sea en diferentes áreas de aplicación, tipos de organizaciones, niveles de aptitud y tamaños de proyecto. RUP no es un sistema con pasos firmemente establecidos, sino un conjunto de metodologías adaptables al contexto y necesidades de cada organización.

El ciclo de desarrollo de *software* describe:
Trabajadores: Define el comportamiento y responsabilidades (rol) de un individuo, grupo de individuos, sistema automatizado o máquina, que trabajan en conjunto como un equipo.

[9] Rational Unified Process, por sus siglas en inglés.
[10] Unified Modeling Language, por sus siglas en inglés.

Actividades: Es una tarea que tiene un propósito claro, es realizada por un trabajador y manipula elementos.
Artefactos: Productos tangibles del proyecto que son producidos, modificados y usados por las actividades. Pueden ser modelos, elementos dentro del modelo, código fuente y ejecutables.
Flujo de actividades: Secuencia de actividades realizadas por trabajadores y que produce un resultado de valor observable.

Esta a su vez, está constituido por cuatro fases:
Inicio: Se describe el negocio y se delimita el proyecto describiendo sus alcances con la identificación de los CU (CU) del sistema.
Elaboración: Se define la arquitectura del sistema y se obtiene una aplicación ejecutable que responde a los CU que la comprometen. A pesar de que se desarrolla a profundidad una parte del sistema, las decisiones sobre la arquitectura se hacen sobre la base de la comprensión del sistema completo y los requerimientos (funcionales y no funcionales) identificados de acuerdo al alcance definido.
Construcción o Implementación: Se obtiene un producto listo para su utilización, que está documentado y tiene un manual de usuario. Se obtiene una o varias versiones del producto que han pasado las pruebas. Se ponen estas versiones a consideración de un subconjunto de usuarios.
Transición o Prueba: La versión ya está lista para su instalación en las condiciones reales. Puede implicar reparación de errores.

El ciclo de vida de RUP se caracteriza por:
Dirigido por CU: Los CU reflejan lo que los usuarios futuros necesitan y desean, lo cual se capta cuando se modela el negocio y se representa a través de los requerimientos. A partir de aquí los CU guían el proceso de desarrollo ya que los modelos que se obtienen, como resultado de los diferentes flujos de trabajo, representan la realización de los CU (cómo se llevan a cabo).
Centrado en la arquitectura: La arquitectura muestra la visión común del sistema completo en la que el equipo de proyecto y los usuarios deben estar de acuerdo, por lo que describe los elementos del modelo que son más importantes para su construcción, los cimientos del sistema que son necesarios como base para comprenderlo, desarrollarlo y producirlo económicamente. RUP se desarrolla mediante iteraciones, comenzando por los CU relevantes desde el punto de vista de la arquitectura.
Iterativo e Incremental: RUP propone que cada fase se desarrolle en iteraciones. Una iteración involucra actividades de todos los flujos de trabajo, aunque desarrolla fundamentalmente algunos más que otros. Por ejemplo, una iteración de elaboración centra su atención en el análisis y diseño, aunque refina los requerimientos y obtiene un producto con un determinado nivel, pero que irá creciendo incrementalmente en cada iteración. Es práctico dividir el trabajo en partes más pequeñas o mini proyectos.

Cuando se estudia RUP, es importante destacar algunos elementos distintivos que la diferencian de otras metodologías. Los CU, no solo sirven para especificar los requisitos, sino que además guían el diseño, la implementación y las pruebas.

1.7 Herramientas y tecnologías de desarrollo

1.7.1 Lenguaje Unificado de Modelado 2.0 (UML)

Es el lenguaje de modelado de sistemas de *software* más conocido y utilizado en la actualidad; está respaldado por el OMG (*Object Management Group*). Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema de *software*. Ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocios y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes de *software* reutilizables (21).

Algunas de las principales ventajas que tiene la utilización de UML en la construcción de sistemas de *software* son (22):

- Puede usarse en las diferentes etapas del ciclo de vida del desarrollo de sistemas
- Es independiente del proceso o metodología de desarrollo y del lenguaje de implementación
- Permite crear y modificar modelos expresivos mediante la combinación de los tipos de diagramas que suministra en su versión 2.0
- Es posible extender la funcionalidad de la notación gráfica mediante estereotipos y proveer una base formal para los diagramas

1.7.2 Visual Paradigm 8.0

Visual Paradigm es una herramienta CASE: Ingeniería de *Software* Asistida por Computación. Propicia un conjunto de ayudas para el desarrollo de programas informáticos, desde la planificación, pasando por el análisis y el diseño, hasta la generación del código fuente de los programas y la documentación (23).

Se caracteriza por:

- Disponibilidad en múltiples plataformas (*Windows*, Linux)
- Diseño centrado en CU y enfocado al negocio que generan un *software* de mayor calidad
- Uso de un lenguaje estándar común a todo el equipo de desarrollo que facilita la comunicación
- Capacidades de ingeniería directa e inversa
- Modelo y código que permanece sincronizado en todo el ciclo de desarrollo
- Disponibilidad de múltiples versiones, con diferentes especificaciones

- Generación de código para Java y exportación como HTML[11]
- Compatibilidad entre ediciones
- Soporte de UML versión 2.1
- Editor de detalles de CU, incluyendo la especificación del modelo general y de las descripciones de los CU
- Generador de informes
- Distribución automática de diagramas
- Reorganización de las figuras y conectores de los diagramas UML
- Importación y exportación de ficheros XML[12]

El uso de esta herramienta permite aumentar la calidad del *software*, a través de la mejora de la productividad en el desarrollo y mantenimiento del *software*. Permite la reutilización del *software*, portabilidad y estandarización de la documentación, además del uso de las distintas metodologías propias de la Ingeniería de *Software*.

1.7.3 Unity3D 4.1.2

Motor gráfico 3D para computadores personales (PC) y Mac que viene empaquetado como una herramienta para crear juegos, aplicaciones interactivas, visualizaciones y animaciones en 3D y tiempo real. *Unity* puede publicar contenido para múltiples plataformas como: PC, Mac, iPhone y Nintendo Wii (25).

El editor de *Unity* es el centro de la línea de producción, ofreciendo un completo editor visual. Se programa usando un lenguaje de *scripts*, lo que significa que el desarrollador no tiene que ser experto en C++, debido a que se compila usando una versión de *JavaScript*, C# y una versión modificada de *Python* llamada Boo. *Unity* es un sistema de desarrollo único. Es enfocado en los *assets* y no en el código, el foco en los *assets* es similar al de una aplicación de modelado 3D.

Assets: Son los bloques constructivos de todo lo que *Unity* posee en sus proyectos. Se guardan en forma de archivos de imagen, modelos del 3D y archivos de sonido.

Scripts: Es una parte esencial de *Unity* ya que define el comportamiento del juego. El *Scripting* es la forma en la que el usuario define el comportamiento o las normas en *Unity*. El lenguaje de programación recomendado para *Unity* es *JavaScript*, aunque C# o Boo pueden ser igualmente usados. En Mac, es llamado como *Unitron*, y en PC, *Uniscite* (25).

[11] HyperText Markup Language, por sus siglas en inglés.
[12] E**X**tensible **M**arkup **L**anguage, por sus siglas en inglés.

Unity3D integra *MonoDevelop* como entorno de desarrollo teniendo en cuenta que es libre y de código abierto. Fue diseñado primordialmente para C# y otros lenguajes como .NET. Es el entorno de desarrollo incluido por defecto en Unity3D.

- La ayuda es muy completa e incluye variedad de ejemplos
- Posee autocompletado de sintaxis
- Posee un navegador incorporado
- Es multiplataforma (26).

Se utilizara Unity3D debido a que posee un motor gráfico potente, posee licencia libre y en el centro se tiene un dominio sobre el uso del mismo, además actualmente en el proyecto se desarrolla un visor de formatos CAD para dispositivos Android con soporte para el formato STL que está llamado a ser integrado con el resultado de este trabajo. También está en desarrollo el sitio web del proyecto CDSEM, el cual deberá ser extendido para soportar la creación de bocetos a mano alzada en la web, apoyándose en las funcionalidades de Unity3D para exportar las aplicaciones para la web, con lo cual la solución obtenida aquí deberá ser exportada. Estos trabajos futuros no forman parte del alcance de la investigación recogida en este documento pero si justifican en gran medida la elección de Unity3D.

1.7.4 JavaScript

Es un lenguaje de programación interpretado. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico. Es un lenguaje del lado del cliente, interpretado por el navegador; no es necesario para su uso tener instalado un *framework* (marco de trabajo, por sus siglas en inglés).

Es dinámico, responde a eventos en tiempo real. Eventos como presionar un botón, pasar el puntero del mouse sobre un determinado texto o el simple hecho de cargar la página o caducar un tiempo. Se utiliza principalmente en su forma del lado del cliente (client-side), implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas (27).

1.7.5 Lenguaje de programación C#

Es un lenguaje de programación orientado a objetos desarrollado y estandarizado por *Microsoft* como parte de su plataforma .NET. Su sintaxis básica deriva de C/C++ y utiliza el modelo de objetos de la plataforma .NET (28).

Presenta las siguientes características:

Programación orientada a objetos: Indica qué partes de código son reutilizables para no volverlas a escribir, con esto se afirma que C# presenta las características necesarias para considerarlo como un lenguaje orientado a objetos, tales son: encapsulación, herencia y polimorfismo; además una de las

mejoras que presenta este lenguaje con respecto a este tipo de programación es que para evitar confusiones no existen variables o funciones globales, sino que se definen dentro de los tipos de datos.
Administración de memoria: Tiene como característica inicializar los datos o variables declaradas en el programa, libera de forma automática la memoria cuando el mismo programa lo cree conveniente. Es decir tiene constructores y destructores, y estos actúan automáticamente a menos que se manipulen desde el código.

Seguridad en el manejo de datos: Tiene la característica de estar comprobando que efectivamente los tipos de datos que se estén manejando, correspondan a los validados para las funciones que han sido creadas. Impide el uso de variables que no han sido inicializadas. Es por ello que permite que no se produzcan errores en el momento de la ejecución. La ventaja de que todos los tipos se deriven de una clase común es que facilita el diseño de colecciones genéricas que puedan almacenar objetos de cualquier tipo.
Compatibilidad: Mantiene una sintaxis muy similar a C, C++ o Java que permite incluir directamente en código escrito en C# fragmentos de código escrito en estos lenguajes. También el *run*-time de lenguaje común ofrece la posibilidad de acceder a código nativo escrito como funciones sueltas no orientadas a objetos.

1.8 Consideraciones parciales del capítulo

Luego de analizar el marco teórico referente al desarrollo y aplicación de tecnologías CAD, se puede llegar a las siguientes conclusiones:

- ✓ El empleo de las tecnologías CAD/CAE/CAM en la empresa conjunta ICVT, favorece la disminución de errores en el diseño y creación de bocetos geométricos conceptuales.
- ✓ Luego de analizar las herramientas representativas de tecnologías CAD, se decide la creación de una aplicación para el sistema operativo Android que se integre con el sistema GALBA-CAD.
- ✓ Aunque existen varias definiciones sobre diseño, ajustado a la problemática planteada se puede definir como: una visión representada en forma gráfica de una obra futura.
- ✓ Teniendo en cuenta que la siguiente investigación forma parte de las soluciones del CDSEM, fue necesario asumir la metodología, herramientas y tecnologías definidas por dicho centro.

CAPÍTULO 2. PROPUESTA DE SOLUCIÓN

En el siguiente capítulo se muestran los resultados obtenidos durante el proceso de desarrollo. Se presentan los artefactos definidos por la dirección del proyecto CDSEM, que responden a la metodología de desarrollo RUP. Se describe el modelo de dominio, los requisitos funcionales y no funcionales, el diagrama de CU, el diagrama de clases, así como los patrones de diseño utilizados.

2.1 Modelo de dominio

Un modelo del dominio captura los tipos más importantes de objetos en el contexto del sistema. Los objetos del dominio representan los elementos que existen o los eventos que suceden en el entorno en el que trabaja el sistema (21).

Se propone en la **Figura 2** un diagrama de clases en el modelo de dominio, que captura los conceptos más importantes del contexto del sistema. Teniendo en cuenta un especialista que diseña bocetos conformados por operaciones de diseño e interactúa con la herramienta *FreeHand* que tiene implícitas estas operaciones de diseño.

El modelo de dominio es una representación gráfica de los principales conceptos en el dominio del problema, muestra las relaciones entre estos conceptos y los atributos de la solución (24).

2.1.1 Diagrama de clases del modelo de dominio

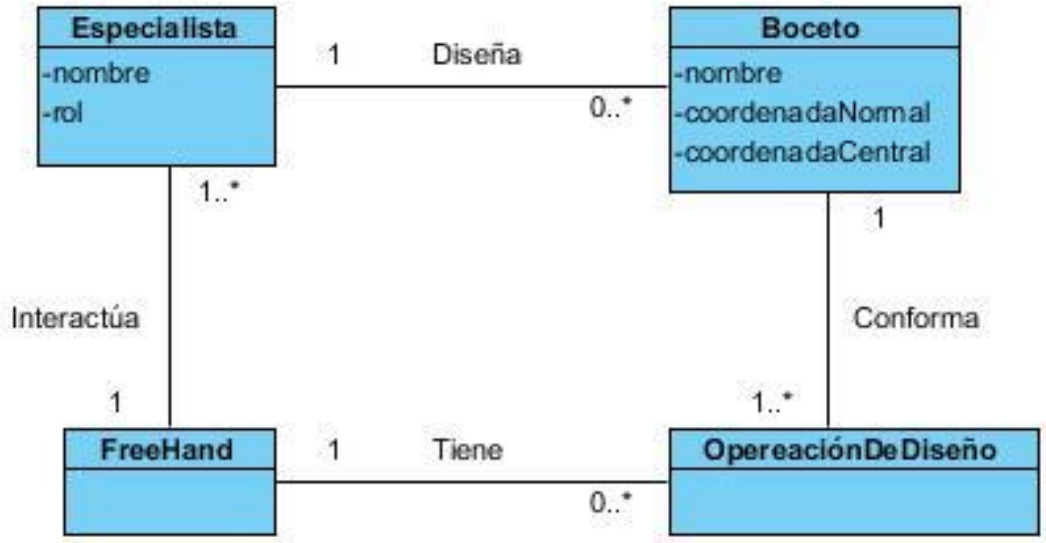


Figura 2 Modelo conceptual

2.1.2 Descripción de las clases del modelo de dominio

Tabla 1 Descripción de los conceptos del modelo de dominio.

Conceptos	Descripción

Especialista	Es la persona encargada de diseñar los bocetos.
FreeHand	Herramienta CAD utilizada para el diseño de bocetos a partir de las operaciones de diseño que trae implementada.
Operaciones de diseño	Son operaciones que permiten crear o modificar bocetos.
Boceto	Es un dibujo esquemático que no incluye detalles ni terminaciones, es la idea más primitiva del diseñador.

2.2 Requisitos de *software*

Los requisitos cumplen un papel primordial en el proceso de producción de *software*, ya que se enfocan en un área fundamental: la definición de lo que se desea producir. Su principal tarea consiste en la generación de especificaciones correctas que describan con claridad, sin ambigüedades, en forma consistente y compacta, el comportamiento del sistema; de esta manera, se pretende minimizar los problemas relacionados al desarrollo de sistemas. Se clasifican en funcionales y no funcionales. Los requisitos funcionales describen qué es lo que el sistema debe hacer para dar soporte a las funciones y objetivos del usuario. Los requisitos no funcionales imponen restricciones de cómo los requisitos funcionales deben ser implementados (29).

Según Ivar Jacobson y James Rumbaugh, algunas de las técnicas de captura de requisitos son:

Entrevistas: La técnica consiste en la realización de preguntas relacionadas con varios aspectos del sistema. La misma cuenta con tres fases: preparación, realización y análisis.

Tormenta de ideas: Es una técnica de reuniones en grupo cuyo objetivo es la generación de ideas en un ambiente libre de críticas o juicios.

Prototipo: Es un modelo del *software* que debe ser construido: se capturan requerimientos, se definen objetivos globales, se diseña rápidamente centrándose en el parte visible al usuario, se construye el prototipo, y los clientes y usuarios evalúan el prototipo refinando los requerimientos.

Para la captura de requisitos se aplicaron dos técnicas de las que se proponen en las bibliografías consultadas, tormenta de ideas y entrevistas, las cuales fueron realizadas en conjunto con profesionales con dominio del negocio y el autor de esta investigación. A continuación se listan los requisitos funcionales que debe cumplir la solución propuesta:

2.2.1 Requisitos funcionales

Los siguientes requisitos fueron agrupados por el criterio de agrupación funcional en 6 requisitos funcionales, en la que se agrupa requisitos que tienen una relación funcional directa, en este caso, por el tipo de datos que va a manejar.

RF1 Gestionar Boceto

- R1.1 Crear boceto
- R1.2 Eliminar boceto
- R1.3 Editar boceto

RF2 Ajustar Curva

- R2.1 Ajustar punto
- R2.2 Ajustar línea
- R2.3 Ajustar círculo
- R2.4 Ajustar arco de círculo
- R2.5 Ajustar elipse

RF3 Guardar boceto

RF4 Cargar boceto

RF5 Configurar aplicación

- R5.1 Editar color de fondo
- R5.2 Editar color de las entidades
- R5.3 Configurar error

RF6 Eliminar entidad

Descripción de los requisitos funcionales

Tabla 2 Requisito funcional <Crear boceto>

Precondiciones			
Flujo de eventos			
Flujo básico: *Crear boceto.*			
1	*Dar clic en el botón **New Sketch***		
2	*Introducir el nombre del boceto*		
3	*Dar clic en el botón **Accept***		
Poscondiciones			
1	*El sistema crea un nuevo boceto con la característica definida*		
Flujos alternativos			
Flujo alternativo 1.a: *Cancelar la acción.*			
1	*Dar clic en el botón **New Sketch***		
2	*Introducir el nombre del boceto*		
3	*Dar clic en el botón **Cancel***		
Poscondiciones			
1	*El sistema no crea ningún boceto*		
Conceptos	*Name*	**Atributos**	*nombre*
Asuntos pendientes	*Posibles mejoras al requisito*		

Tabla 3 Requisito funcional <Editar boceto>

Precondiciones	*Debe existir al menos un boceto creado*		
Flujo de eventos			
Flujo básico: *Editar boceto.*			
1	*Seleccionar el boceto que se desea editar*		
2	*Dar clic en el botón* ***Edit Sketch***		
3	*Diseñar las entidades deseadas*		
Poscondiciones			
1	*El sistema guarda las entidades definidas*		
Conceptos	*Name*	**Atributos**	*nombre*
Asuntos pendientes	*Posibles mejoras al requisito*		

Tabla 4 Requisito funcional <Eliminar boceto>

Precondiciones	*Debe existir al menos un boceto creado*		
Flujo de eventos			
Flujo básico: *Eliminar boceto.*			
1	*Seleccionar el boceto que se desea eliminar*		
2	*Dar clic en el botón* ***Delete Sketch***		
Poscondiciones			
1	*El sistema elimina el boceto seleccionado*		
Conceptos	*Name*	**Atributos**	*nombre*
Asuntos pendientes	*Posibles mejoras al requisito*		

Tabla 5 Requisito funcional <Cargar boceto>

Precondiciones	*Debe existir al menos un boceto creado*
Flujo de eventos	
Flujo básico: *Cargar boceto.*	
1	*Dar clic en el botón* ***Load Sketch***
2	*Seleccionar el boceto que se desea cargar*
3	*Abrir el boceto seleccionado*
Poscondiciones	
1	*El sistema carga el boceto seleccionado*
Flujos alternativos	
Flujo alternativo 1.a: *No existen bocetos*	
1	*Dar clic en el botón* ***Load Sketch***
Poscondiciones	
1	El sistema no encuentra ningún boceto
Asuntos pendientes	*Posibles mejoras al requisito*

Tabla 6 Requisito funcional <Guardar boceto>

Precondiciones	*Debe existir al menos un boceto creado*
Flujo de eventos	
Flujo básico: *Guardar boceto.*	
1	*El usuario selecciona el boceto que se desea guardar*
2	*Dar clic en el botón* ***Save Sketch***
Poscondiciones	
1	*El sistema guarda el boceto seleccionado*
Asuntos pendientes	*Posibles mejoras al requisito*

Tabla 7 Requisito funcional <Ajustar punto>

Precondiciones	
Flujo de eventos	
Flujo básico: *Ajustar punto*	
1	*El usuario diseña la entidad punto*
2	*El sistema ajusta el error geométrico permisible para la entidad punto*
3	*El sistema corrige la entidad diseñada y grafica el punto*
Poscondiciones	
1	*El sistema diseña la entidad punto*
Asuntos pendientes	*Posibles mejoras al requisito*

Tabla 8 Requisito funcional <Ajustar línea>

Precondiciones	
Flujo de eventos	
Flujo básico: *Ajustar línea*	
1	*El usuario diseña la entidad línea*
2	*El sistema ajusta el error geométrico permisible para la entidad línea*
3	*El sistema corrige la entidad diseñada y grafica la línea*
Poscondiciones	
1	*El sistema diseña la entidad línea*
Asuntos pendientes	*Posibles mejoras al requisito*

Tabla 9 Requisito funcional <Ajustar círculo>

Precondiciones	
Flujo de eventos	
Flujo básico: *Ajustar círculo*	
1	*El usuario diseña la entidad círculo*
2	*El sistema ajusta el error geométrico permisible para la entidad círculo*

3	*El sistema corrige la entidad diseñada y grafica el círculo*
Poscondiciones	
1	*El sistema diseña la entidad círculo*
Asuntos pendientes	*Posibles mejoras al requisito*

Tabla 10 Requisito funcional <Ajustar elipse>

Precondiciones	
Flujo de eventos	
Flujo básico: *Ajustar elipse*	
1	*El usuario diseña la entidad elipse*
2	*El sistema ajusta el error geométrico permisible para la entidad elipse*
3	*El sistema corrige la entidad diseñada y grafica el elipse*
Poscondiciones	
1	*El sistema diseña la entidad elipse*
Asuntos pendientes	*Posibles mejoras al requisito*

Tabla 11 Requisito funcional <Editar color de fondo>

Precondiciones	
Flujo de eventos	
Flujo básico: *Editar color de fondo*	
1	*Dar clic en el botón **Setting***
2	*El usuario selecciona el color de fondo que desee*
3	*Dar clic en el botón **Accept***
Poscondiciones	
1	*El sistema cambia el color de fondo de los bocetos*
Flujos alternativos 1.a: *Cancelar*	
1	*Dar clic en el botón **Setting***
2	*Seleccionar el color de fondo*
3	*Dar clic en el botón **Cancel***
Poscondiciones	
1	*No se cambia el color de fondo del boceto seleccionado*
Asuntos pendientes	*Posibles mejoras al requisito*

Tabla 12 Requisito funcional <Editar color de entidad>

Precondiciones	
Flujo de eventos	
Flujo básico: *Editar color de entidad*	
1	*Dar clic en el botón **Setting***
2	*El usuario selecciona el color de las entidades que desee*

3	*Dar clic en el botón* ***Accept***
Poscondiciones	
1	*El sistema cambia el color de las entidades*
Flujos alternativos 1.a: *Cancelar*	
1	*El usuario da clic en el botón* ***Setting***
2	*El usuario selecciona el color de las entidades*
3	*Dar clic en el botón* ***Cancel***
Poscondiciones	
1	*No se cambia el color de las entidades*
Asuntos pendientes	*Posibles mejoras al requisito*

Tabla 13 Requisito funcional <Configurar error>

Precondiciones	
Flujo de eventos	
Flujo básico: *Configurar error*	
1	*Dar clic en el botón* ***Setting***
2	*El sistema muestra una interfaz con los valores predeterminados*
3	*El usuario modifica el error predeterminado para cada entidad que desee diseñar*
4	*Dar clic en el botón* ***Accept***
5	*El sistema guarda la nueva configuración*
Poscondiciones	
1	El *sistema guarda la nueva configuración*
Flujos alternativos 1.a: *Configuración predeterminada*	
1	*Dar clic en el botón* ***Default Setting***
Poscondiciones	
1	*El sistema carga la configuración predeterminada*
Asuntos pendientes	*Posibles mejoras al requisito*

Tabla 14 Requisito funcional <Eliminar entidad>

Precondiciones	
Flujo de eventos	
Flujo básico: *Eliminar entidad*	
1	*El usuario selecciona la opción* ***box Remove Entities Mode***
2	*El usuario selecciona las entidades a eliminar*
3	*El* sistema elimina la entidad seleccionada
Poscondiciones	
1	*El sistema elimina la entidad seleccionada*
Asuntos pendientes	*Posibles mejoras al requisito*

2.2.2 Requisitos no funcionales (RNF)

La solución propuesta fue iniciada en el CDSEM. De esta manera los RNF a los que se debe acoger el componente a desarrollar son los establecidos al inicio del proceso de desarrollo. Los requisitos que se muestran a continuación son agrupados en las siguientes categorías: *Software*, *Hardware*, Restricciones en el diseño y la implementación, Requisitos de licencia y Requisitos legales.

Software

Sistema operativo:

- Android
- *Firmware* 2.2 o superior
- Kernel Versión: 2.6.32.9 o superior

Hardware

El sistema debe funcionar con las siguientes especificaciones:

- Procesador ARM con frecuencia mínima de 1 GHz.
- Memoria RAM: 512 MB Mínimo.
- Memoria interna: 100 MB

Restricción de diseño e implementación

- Lenguajes de programación: C#
- Motor Gráfico: Unity3D
- Entorno de Desarrollo Integrado (IDE): Unity3D y *MonoDevelop*
- *Framework* de desarrollo: Unity3D

Requisitos de licencias

- Android *Open Source License*
- *Unity Free License*

Requisitos legales, de derecho de autor y otros

Se declara a Anicel Mejias Rodríguez ser el único autor de este trabajo y se reconoce a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

2.3 Modelo de casos de uso

El modelo de CU captura todos los requisitos funcionales del sistema, el cual ayuda al cliente, a los usuarios y a los desarrolladores a llegar a un acuerdo sobre cómo utilizar el sistema. Un caso de uso especifica una secuencia de acciones, incluyendo variantes, que el sistema puede llevar a cabo, y que producen un resultado observable de valor para un actor concreto (21).

Un actor representa un conjunto coherente de roles que los usuarios de los CU juegan al interactuar con estos. Normalmente, un actor representa un rol que es jugado por una persona, un dispositivo *hardware* o incluso otro sistema (24).

Según Ivar Jacobson y James Rumbaugh, el modelo de CU es una técnica de captura de requisitos que fuerza a pensar en términos de importancia para el usuario y no solo en términos de funciones que sería bueno contemplar. Se define un caso de uso como un fragmento de funcionalidad del sistema que proporciona al usuario un valor añadido. Los CU representan los requisitos funcionales del sistema (24).

En RUP los CU no son solo una herramienta para especificar los requisitos del sistema. También guían su diseño, implementación y prueba. Los CU constituyen un elemento integrador y una guía del trabajo como se muestra en la **Figura 3**.

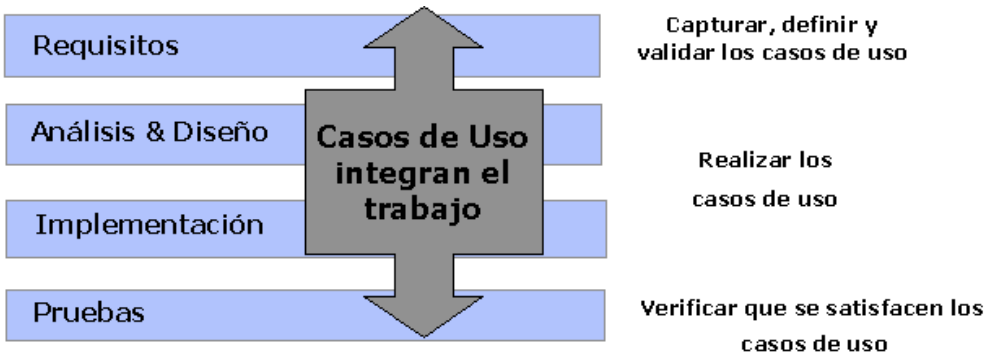


Figura 3 Los CU integran el trabajo

2.3.1 Diagrama de casos de uso del sistema

Un diagrama de CU muestra un conjunto de CU, los actores y sus relaciones; los diagramas de CU muestran los CU de un sistema desde un punto de vista estático (21). En la **Figura 4** se muestra el diagrama de CU del sistema de la solución propuesta, donde se identifica un actor Especialista que es el encargado de inicializar los CU Gestionar boceto, Ajustar curva, Guardar boceto, Cargar boceto, Configurar la aplicación y Eliminar entidad. Se utiliza en el caso de uso Gestionar boceto el patrón de caso de uso CRUD (*Creating, Reading, Updating, Deleting* por sus siglas en inglés) completo y en los CU Ajustar curvas y Configurar aplicación un CRUD parcial, ya que el patrón permite modelar las operaciones que pueden ser realizadas sobre una parte de la información de un tipo específico.

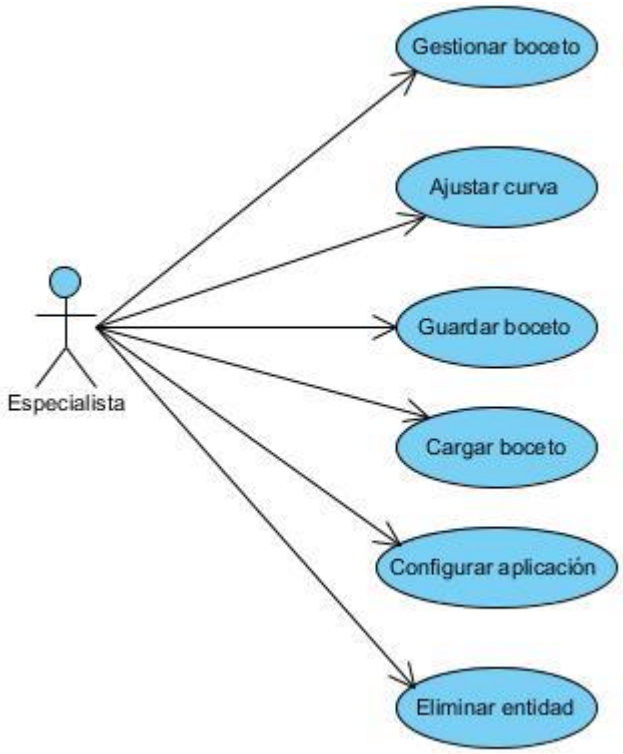


Figura 4 Diagrama de caso de uso del sistema

2.3.2 Descripción de los casos de uso del sistema

Tabla 15 Descripción del caso de uso <Gestionar boceto>

Nombre	Gestionar boceto	
Objetivo	Gestionar el diseño de los bocetos	
Actor	Especialista	
Resumen	Se inicia cuando el actor selecciona alguna de las siguientes opciones: Crear boceto, Editar boceto o Eliminar boceto. Finaliza cuando el sistema realiza la operación seleccionada.	
Prioridad	Crítica	
Precondiciones	N/A	
Poscondiciones	Se obtendrá una transformación de acuerdo a la operación seleccionada.	
Flujo de eventos		
Flujo básico: Crear boceto		
	Actor	**Sistema**
1.	Dar clic en el botón ***New Sketch***	El sistema muestra la interfaz
2.	Introducir el nombre del boceto	El sistema valida que el campo este correcto
3.	Dar clic en el botón ***Accept***	El sistema crea el boceto definido
Flujo alterno: Cancelar		
4.	Dar clic en el botón ***Cancel***	El sistema cancela la creación del boceto
Interfaz		
Flujo de eventos		

Flujo básico: Editar boceto		
	Actor	**Sistema**
1.	El actor selecciona el boceto a Editar	
2.	Dar clic en el botón ***Edit Sketch***	El sistema abre el boceto a Editar
3.	Diseñar las entidades deseadas (punto, línea, círculo, arco de círculo, elipse)	El sistema crea las entidades deseadas
4.	Dar clic en el botón ***Accept***	El sistema edita el boceto seleccionado.
Flujo de eventos		
Flujo básico: Eliminar boceto		
	Actor	**Sistema**
1	Seleccionar el boceto que se desea eliminar	
2.	Dar clic en el botón ***Delete Sketch***	El sistema elimina el boceto seleccionado

Tabla 16 Descripción del caso de uso <Ajustar curva >

Nombre	Ajustar curva	
Objetivo	Ajustar la figura diseñada de acuerdo a las coordenadas permisibles matemáticamente.	
Actor	Especialista	
Resumen	Se inicia cuando el actor diseña cualquiera de las entidades que componen un boceto (punto, línea, círculo, arco, elipse, arco de círculo). El sistema debe ser capaz de ajustar y corregir cada entidad al rango de errores permitidos. Finaliza cuando el sistema muestra la entidad diseñada.	
Prioridad	Crítica	
Precondiciones	Debe existir al menos un boceto creado.	
Poscondiciones	Se crea la entidad deseada.	
Flujo de eventos		
Flujo básico: Ajustar curva		
	Actor	**Sistema**
1.	El actor diseña la entidad deseada	El sistema ajusta el error geométrico permisible e identifica la entidad
2.		El sistema corrige la entidad diseñada y muestra en pantalla el resultado

Tabla 17 Descripción del caso de uso <Guardar boceto>

Nombre	Guardar boceto
Objetivo	Guardar el boceto diseñado
Actor	Especialista

Resumen	Se inicia cuando el actor selecciona un boceto para guardarlo en cualquier dirección física establecida en el sistema.	
Prioridad	Crítica	
Precondiciones	Debe existir al menos un boceto creado.	
Poscondiciones	Se guarda el boceto seleccionado.	
Flujo de eventos		
Flujo básico: Guardar boceto		
	Actor	**Sistema**
1.	Seleccionar el boceto que se desea guardar	
2.	Dar clic en el botón ***Save Sketch***	El sistema permite seleccionar la ruta para guardar el boceto seleccionado
3.	Dar clic en el botón ***Accept***	
Flujo alterno: Cancelar		
	Actor	**Sistema**
3.	Dar clic en el botón **Cancel**	El sistema no guarda el boceto seleccionado
Flujo alterno: No existen bocetos		
	Actor	**Sistema**
1.	Seleccionar el boceto que se desea guardar	No existen bocetos para guardar

Tabla 18 Descripción del caso de uso <Cargar boceto>

Nombre	Cargar boceto	
Objetivo	Cargar el boceto seleccionado	
Actor	Especialista	
Resumen	Se inicia cuando se selecciona un boceto para cargarlo del sistema.	
Prioridad	Crítica	
Precondiciones	Debe existir al menos un boceto creado.	
Poscondiciones	Se carga el boceto seleccionado.	
Flujo de eventos		
Flujo básico: Cargar boceto		
	Actor	**Sistema**
1.	Dar clic en el botón ***Load Sketch***	
2.	Seleccionar el boceto que se desea cargar	
3.	Dar clic en el botón ***Accept***	El sistema carga el boceto seleccionado
Flujo alterno: Cancelar		
	Actor	**Sistema**
3.	Dar clic en el botón ***Cancel***	El sistema no carga el boceto seleccionado

Flujo alterno: No existen bocetos		
	Actor	**Sistema**
2.	Seleccionar el boceto que se desea cargar	El sistema no encuentra ningún boceto

Tabla 19 Descripción del caso de uso <Eliminar entidad>

Nombre	Eliminar entidad	
Objetivo	Eliminar la entidad seleccionada por el especialista.	
Actor	Especialista	
Resumen	Se inicia cuando el usuario da clic en la opción ***box Delete Entity***. Finaliza cuando el usuario desmarca la opción ***box Delete Entity***.	
Prioridad	Crítica	
Precondiciones	Debe existir al menos una entidad creada.	
Poscondiciones	Se elimina la entidad seleccionada.	
Flujo de eventos		
Flujo básico: Eliminar entidad		
	Actor	**Sistema**
1.	Dar clic en la opción ***box Delete Entity***	
2.	El actor selecciona la entidad que desea eliminar	El sistema elimina la entidad seleccionada

Tabla 20 Descripción del caso de uso <Configurar aplicación>

Nombre	Configurar aplicación	
Objetivo	Configurar la aplicación.	
Actor	Especialista	
Resumen	El caso de uso es inicializado por el especialista, el sistema debe permitir editar el color de fondo de los bocetos, editar el color de las entidades, cargar las configuraciones del mínimo de error permisible para cada entidad.	
Prioridad	Crítica	
Precondiciones	N/A	
Poscondiciones	Se realizan las configuraciones establecidas.	
Flujo de eventos		
Flujo básico: Editar color de fondo		
	Actor	**Sistema**
1.	Dar clic en el botón ***Setting***	
2.	El actor selecciona el color de fondo que desee	
3.	Dar clic en el botón ***Accept***	El sistema guarda la configuración realizada

Flujo de eventos		
Flujo básico: Editar color de las entidades		
	Actor	**Sistema**
1.	El actor da clic en el botón ***Setting***	
2.	El actor selecciona el color de entidad que desee	El sistema muestra la configuración realizada
3.	Dar clic en el botón ***Accept***	El sistema guarda la configuración realizada
Flujo de eventos		
Flujo básico: Configurar error		
	Actor	**Sistema**
1.	Dar clic en el botón ***Setting***	
2.		El sistema muestra una interfaz con los valores del error para las entidades
3.	El actor modifica el error predeterminado para cada entidad que desee diseñar	
4.	Dar clic en el botón ***Accept***	El sistema guarda la nueva configuración
Flujo alterno: Configuración predeterminada		
	Actor	**Sistema**
1.	Dar clic en el botón ***Setting***	
2.		El sistema muestra una interfaz con los valores del error para las entidades
3.	El actor da clic en el botón ***Default Setting***	El sistema carga la configuración predeterminada
4.	Dar clic en el botón ***Accept***	El sistema guarda la nueva configuración

2.4 Diagrama de clases

El diagrama de clase describe la estructura del sistema, mostrando las clases, relaciones entre clases, atributos, métodos y asociaciones (30).

La **Figura 5** ¡Error! No se encuentra el origen de la referencia.muestra el diagrama de clases diseñado para la solución, donde se representan las principales clases que componen el sistema y las relaciones entre ellas.

***Main*:** Es la clase encargada de controlar la creación de los bocetos y entidades definidas.

***PresentationEntity*:** Esta clase incluye las clases de presentación, entre ellas *PointPresentation, LinePresentation, CirclePresentation, ArcPresentation* y *EllipsePresentation*.

***Entity*:** Esta clase agrupa todas las entidades que se diseñan en la aplicación, existe una clase para cada entidad que se desee diseñar, en estas clases se implementan los métodos específicos para cada entidad.

***Sketcher*:** Es la clase encargada de definir las coordenadas y nombre del boceto que se desee crear, así como del error geométrico permisible para cada entidad.

Setting: Es la clase encargada de configurar en la aplicación el color de las entidades, el color de fondo del boceto y los errores permitidos para cada entidad.

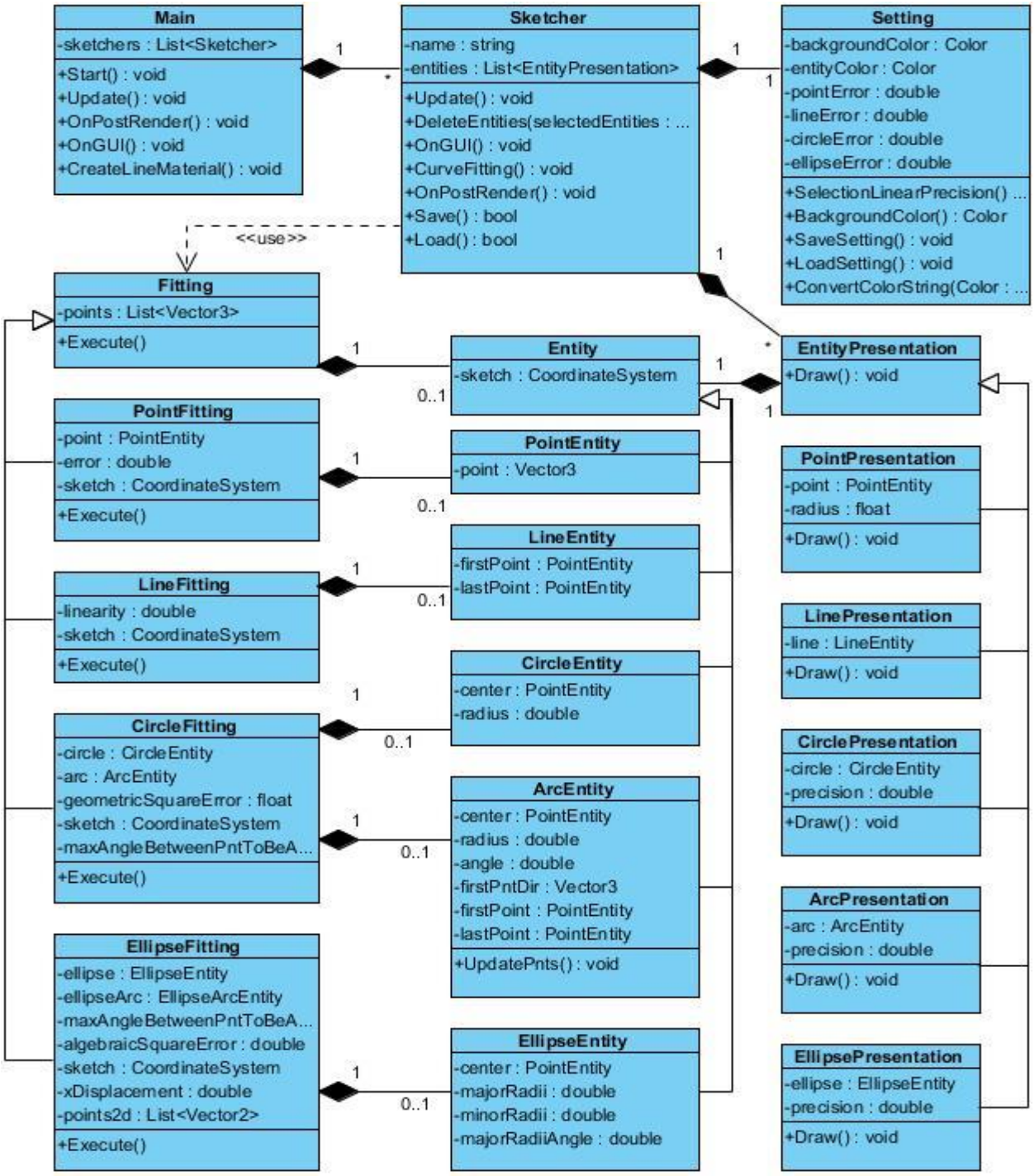


Figura 5 Diagrama de clases

2.5 Patrones de diseño utilizados

Los patrones de diseño son una solución estándar para un problema común de diseño dentro de un contexto dado y constituyen una manera más práctica de describir ciertos aspectos de la organización de un programa.

Después de analizar los patrones de diseño propuestos en *"Introduction to Design Patterns"* (31), se considera necesario para el desarrollo de la herramienta el uso de los siguientes patrones:

2.5.1 Patrones GRASP utilizados

Son los patrones generales de *software* para asignar responsabilidades, GRASP[13], describen en su totalidad los principios fundamentales sobre la asignación de responsabilidades a objetos, todo en forma de patrones (32).

- **Bajo acoplamiento:** El acoplamiento es una medida de la fuerza con que una clase está conectada a otras clases, con que las conoce y con que recurre a ellas. Una clase con bajo (o débil) acoplamiento no depende de muchas otras. Este patrón se evidencia en las clases *Setting* y *Sketcher*.
- **Alta cohesión:** El uso de éste patrón indica que la información almacenada en las clases debe ser coherente y relacionada a lo que se maneja en dicha clase, para esta solución se ve su uso igualmente en la clase controladora.

2.5.2 Patrones GOF utilizados

- **Singleton:** El patrón *singleton* consiste en crear una instancia de un objeto y solo una, para toda nuestra aplicación. Sería como una especie de variable global que almacena nuestro objeto. El patrón *singleton* se implementa mediante una clase con un constructor privado. Existe un método que crea una instancia del objeto llamando al constructor solo si todavía no existe ninguna (33). Ver **Anexo 1**

2.6 Consideraciones parciales del capítulo

Luego de realizar la propuesta de solución mediante el diseño, se puede apreciar que:

- ✓ Fueron expuestos los artefactos generados durante el análisis y diseño de la solución propuesta.
- ✓ Mediante el empleo de las técnicas de captura de requisitos se identificaron y describieron los requisitos funcionales de la Herramienta de diseño conceptual basado en bocetos a mano alzada para el *software* GALBA-CAD y los RNF para garantizar la correcta ejecución de la aplicación.
- ✓ Se mostró el diseño de clases para lograr una mayor comprensión de cómo están estructuradas las clases que conforman la herramienta.
- ✓ Una vez concluido el diseño de la solución puede darse paso a los flujos de implementación y prueba de la misma.

[13] Por sus siglas en inglés General Responsability Asignment *Software* Paterns.

CAPÍTULO 3. IMPLEMENTACIÓN Y PRUEBAS

En el capítulo se muestran los artefactos generados en la fase de implementación. Se presentan los componentes de *software* y el código fuente de las funcionalidades más importantes. Se exponen además el conjunto de pruebas que evalúan el cumplimiento de los requisitos planteados y la calidad de la herramienta desarrollada.

3.1 Diagrama de componentes

Los diagramas de componentes muestran la organización y dependencias lógicas entre componentes de *software*. Un diagrama de componentes permite visualizar con facilidad la estructura general del sistema y el comportamiento de los servicios que estos componentes proporcionan y utilizan a través de las interfaces (36). Los diagramas de componentes se utilizan para modelar la vista de implementación estática de un sistema. Esto implica modelar las cosas físicas que residen en un nodo tales como ejecutables, bibliotecas y archivos. Para la vista de implementación de la aplicación desarrollada que se muestra en la **Figura 6** se definen los componentes de tipo archivo de código fuente y archivo de intercambio en formato skt, ambos con estereotipo *file*.

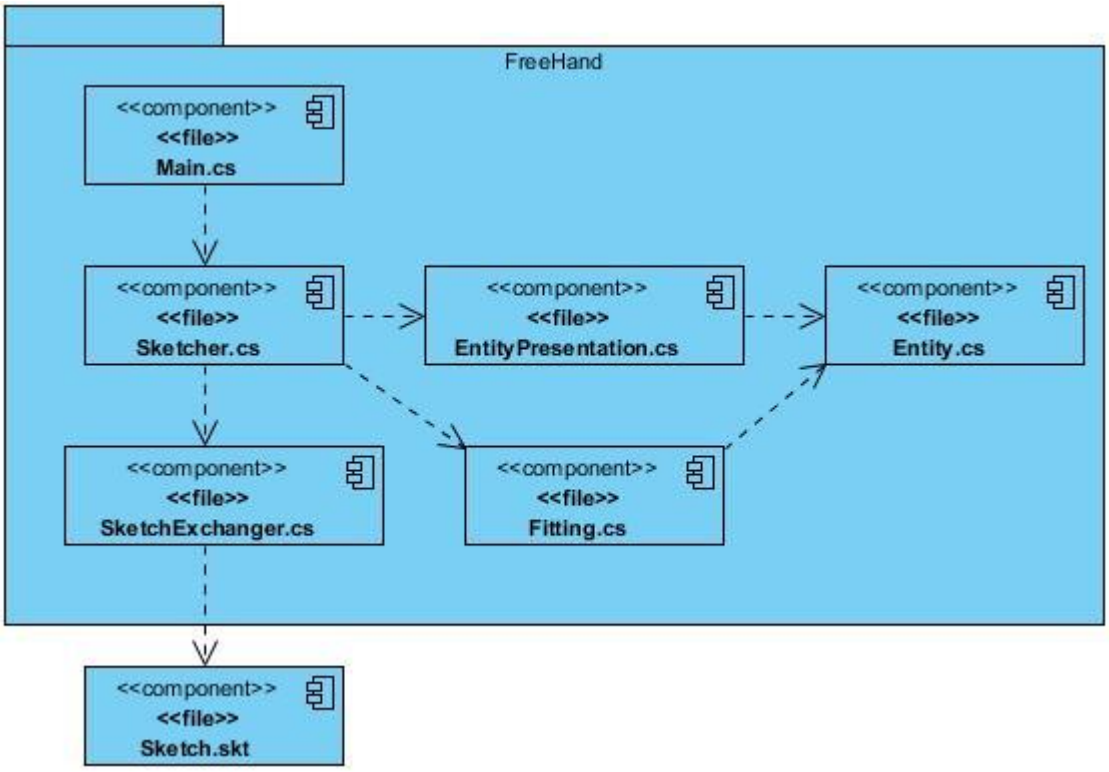


Figura 6 Diagrama de componentes

El paquete FreeHand es el resultado del presente trabajo, tiene como objetivo la creación de bocetos geométricos conceptuales a mano alzada permitiendo guardar y cargar los bocetos creados en archivos con extensión skt.

3.2 Estándares de codificación

3.2.1 Espacios de nombres y clases

- ✓ Los identificadores para los espacios de nombres y las clases siempre serán sustantivos y comenzarán además con letra inicial mayúscula. El resto del nombre será en minúscula. Véase el caso para una clase.

Ejemplo:
class Vector **{** **};**

- ✓ Ningún nombre puede contener underscores.

Ejemplo:
class CVector_3D // Incorrecto **{** **};**

- ✓ En caso de que el identificador de un espacio de nombre o de una clase sea compuesto, se escribirá la letra inicial de cada palabra en mayúscula.

Ejemplo:
class CommandFactory **{** **};**

NOTA: De manera general se intentará mantener los nombres de las clases simples y descriptivos. Se deben utilizar palabras completas, evitar acrónimos o abreviaturas (a no ser que la abreviatura sea mucho más conocida que el nombre completo, como URL o HTML).

3.2.2 Atributos de clases, parámetros, variables locales

- ✓ Todos los caracteres en minúscula.

Ejemplo:

```
class Person
{
private:
int age;
char name;
};
```

- ✓ En caso de que existan nombre compuestos se utiliza la siguiente regla. El primer nombre se escribe con minúscula y a partir de ese momento cada nuevo nombre con letra inicial mayúscula.

Ejemplo:

```
int do(int code, int age, char sex)
{
int theCode = code;
int theAge = age;
return (theCode * theAge);
}
class Person
{
private char fullName;
};
```

- ✓ Ningún nombre podrá contener underscores.

Ejemplo:

```
class Person
{
```

```
private char full_Name; // Incorrecto
};
```

NOTA: Los nombres de variables de un solo carácter se deben evitar, excepto para variables de índices temporales. Algunos nombres comunes para variables temporales son i, j, k, m, y n para enteros; c, d, y e para caracteres. Así mismo se deberá evitar, para la identificación de variables, la utilización de las letras “o” y “l” (ele). Su evidente parecido a los números 0 y 1 puede ocasionar problemas difíciles de depurar.

3.2.3 Métodos

- ✓ Se escribirá la letra inicial con mayúscula, si es compuesto se escribirá la letra inicial de cada palabra en mayúscula y no puede contener underscores.

Ejemplo:

```
class Vector
{
public float Length();
public void Normalize();
public bool IsUnary();
};
```

3.3 Código fuente

A continuación se muestra fragmentos de código referente a los métodos *Execute* de las clases *PointFitting*, *LineFitting*, *CircleFitting* y *EllipseFitting*, respectivamente. Este método es el responsable de realizar el ajuste de puntos en cada una de las clases antes mencionadas.

PointFitting

```
public override void Execute(){
            Vector3 pnt = new Vector3();
            for(int i=0; i<points.Count; i++){
                  pnt += points[i];
            }
            pnt /= points.Count;
            error = 0;

            for(int i=0; i<points.Count; i++){
```

```
                double distance = Vector3.Distance(pnt, points[i]);
                if(error < distance){
                        error = distance;
                }
        }

        if(error <= Setting.Instance.PointError){
                point = new PointEntity(pnt, sketch);
        }
        else {
                point = null;
        }
    }
}
```

LineFitting

```
public override void Execute(){
        if(points.Count == 0){
                line = null;
                return;
        }
        linearity = double.MaxValue;
        Vector3 firstPoint = points[0];
        Vector3 lastPoint = points[points.Count - 1];

        if(firstPoint.Equals(lastPoint)){
                line = null;
        }
        else {
                double length = Vector3.Distance(firstPoint, lastPoint);

                double chordLength = 0;

                for(int i=1; i<points.Count; i++){
                        chordLength += Vector3.Distance(points[i-1], points[i]);
                }
                linearity = length/chordLength;
                if(Linearity >= Setting.Instance.LineError){
```

```
                        line = new LineEntity(firstPoint, lastPoint, sketch);
                  }
                  else{
                        line = null;
                  }
            }
      }
}
```

CircleFitting

```
public override void Execute (){
            List<Vector2> points2d = points3dTo2d();
       //This is an implementation of the linear least-squares algorithm by
Coope (1993)
            //for circle fitting
            // Having the points, we can fit a circle
            // through the point set, consisting of n points.
            // The (n times 3) matrix consists of
            //   x_1, y_1, 1
            //   x_2, y_2, 1
            //                ...
            //   x_n, y_n, 1
            // where x_i, y_i is the position of point p_i
            // The vector y of length n consists of
            //              x_i*x_i+y_i*y_i
            // for i=1...n.

            if(points2d.Count<3){
                  circle = null;
                  geometricSquareError = int.MaxValue;
                  return;
            }
            RectangularMatrix M = new RectangularMatrix(points2d.Count, 3);
            ColumnVector y = new ColumnVector(points2d.Count);

            for(int i=0; i<points2d.Count; i++){
                  M[i, 0] = points2d[i].x;
                  M[i, 1] = points2d[i].y;
```

```
            M[i, 2] = 1;

        y[i] = Mathf.Pow(points2d[i].x, 2) + Mathf.Pow(points2d[i].y, 2);
        }

        // Now, the general linear least-square fitting problem
        //    min_z || M*z - y||_2^2
        // is solved by solving the system of linear equations
        //    (M^T*M) * z = (M^T*y)
        RectangularMatrix MT = M.Transpose();
        ColumnVector z = (MT*M).QRDecomposition().Solve(MT * y);

        // Sistema de ecuaciones: A x = b
        // Donde A es MT * M
        // Donde x es z
        // Donde b es MT * y
        // Finally, we can read from the solution vector z the coordinates
        [xm, ym] of the center
        // and the radius r.
        float xc = (float)(z[0]*0.5);
        float yc = (float)(z[1]*0.5);

        Vector3 center = sketch.point2dTo3d(new Vector2(xc, yc));

        float radius = Mathf.Sqrt((float)z[2] + Mathf.Pow(xc, 2) +
        Mathf.Pow(yc, 2));
        circle = new CircleEntity(center, radius, sketch);
        geometricSquareError = 0;

        for(int i=0; i<points2d.Count; i++) {
              float tx = (float)points2d[i].x;
              float ty = (float)points2d[i].y;
              geometricSquareError += Mathf.Pow(radius -
              Mathf.Sqrt(Mathf.Pow(tx-xc, 2) + Mathf.Pow(ty-yc, 2)), 2);
        }
        arcFitting(points2d);
    }
```

EllipseFitting

```
public override void Execute (){
            if(points.Count < 3){
                  return;
            }
            points2d = points3dTo2d();
            xDisplacement = findXDisplacement();
            double x4 = summation(4, 0);
            double x3y = summation(3, 1);
            double x2y2 = summation(2, 2);
            double x3 = summation(3, 0);
            double x2y = summation(2, 1);
            double x2 = summation(2, 0);
            double xy3 = summation(1, 3);
            double xy2 = summation(1, 2);
            double xy = summation(1, 1);
            double y4 = summation(0, 4);
            double y3 = summation(0, 3);
            double y2 = summation(0, 2);
            double x = summation(1, 0);
            double y = summation(0, 1);

            SquareMatrix M = new SquareMatrix(5);

            M[0, 0] = x4;     M[0, 1] = x3y;    M[0, 2] = x2y2;   M[0, 3] = x3;
                  M[0, 4] = x2y;
            M[1, 0] = x3y;    M[1, 1] = x2y2;   M[1, 2] = xy3;    M[1, 3] = x2y;
                  M[1, 4] = xy2;
            M[2, 0] = x2y2;   M[2, 1] = xy3;    M[2, 2] = y4;     M[2, 3] = xy2;
                  M[2, 4] = y3;
            M[3, 0] = x3;     M[3, 1] = x2y;    M[3, 2] = xy2;    M[3, 3] = x2;
                  M[3, 4] = xy;
            M[4, 0] = x2y;    M[4, 1] = xy2;    M[4, 2] = y3;     M[4, 3] = xy;
                  M[4, 4] = y2;
            ColumnVector independent = new ColumnVector(5);
            independent[0] = -x2;
            independent[1] = -xy;
            independent[2] = -y2;
            independent[3] = -x;
```

```
independent[4] = -y;
ColumnVector solution = M.QRDecomposition().Solve(independent);
double A = solution[0];
double B = solution[1];
double C = solution[2];
double D = solution[3];
double E = solution[4];
double F = 1;
double xc = (B*E-2*C*D)/(4.0f*A*C-B*B);
double yc = (D*B-2*A*E)/(4.0f*A*C-B*B);
// Value in radians
double majorRadiusAngle = Math.Atan((B/(A-C)) )/2.0;
if(majorRadiusAngle < 0){
      majorRadiusAngle = Math.Pi()/2.0 + majorRadiusAngle;
}
if(B >= 0){
      majorRadiusAngle = Math.Pi()/2.0 + majorRadiusAngle;
}
double lambda = A*C - Math.Pow(B/2.0, 2);
SquareMatrix triMatrix = new SquareMatrix(3);
triMatrix[0, 0] = A;triMatrix[0, 1] = B/2.0;triMatrix[0, 2] = D/2.0;
triMatrix[1, 0] = B/2.0;triMatrix[1, 1] = C;triMatrix[1, 2] = E/2.0;
triMatrix[2, 0] = D/2.0;triMatrix[2, 1] = E/2.0;triMatrix[2, 2] = F;
double tri = triMatrix.QRDecomposition().Determinant();
double coef = Math.Abs(tri)/Math.Abs(lambda);
double R1 = Math.Sqrt( coef * (2.0/((A+C) + Math.Sqrt(Math.Pow(A-C,
2)   + B*B))) );
double R2 = Math.Sqrt( coef * (2.0/((A+C) - Math.Sqrt(Math.Pow(A-C,
2) + B*B))) );
double majorRadius = (R1 > R2) ? R1 : R2;
double minorRadius = (R1 < R2) ? R1 : R2;
algebraicSquareError = 0;
for(int p=0; p<points.Count; p++){
      double tx = points2d[p].x + xDisplacement;
      double ty = points2d[p].y;
       algebraicSquareError += Math.Pow(A*tx*tx + B*tx*ty + C*ty*ty +
        D*tx + E*ty + F, 2);
}
double center2dX = xc - xDisplacement;
```

```
            double center2dY = yc;
            Vector3 center = sketch.point2dTo3d(new Vector2((float)(center2dX),
            (float)(center2dY)));
            Ellipse = new EllipseEntity(center, majorRadius, minorRadius,
            majorRadiusAngle, sketch);
            arcFitting(points2d);
        }
```

3.4 Pruebas realizadas

Las pruebas son una actividad, en la cual un sistema o componente es ejecutado bajo condiciones o requerimientos específicos, los resultados son observados y registrados para una posterior evaluación. Constituyen una revisión final de las especificaciones del diseño y de la codificación (37).

Para realizar pruebas al sistema desarrollado se utilizó un Samsung Galaxy Tab 7.0 Plus con sistema operativo Android 4.1.2 y también un Samsung Galaxy S3 con Android 4.3. A estos dispositivos se les instaló la aplicación *FreeHand* implementada, brindando la posibilidad de realizar varias acciones que dan cumplimiento a cada uno de los requisitos funcionales planteados, como se muestra en la **Figura 7**.

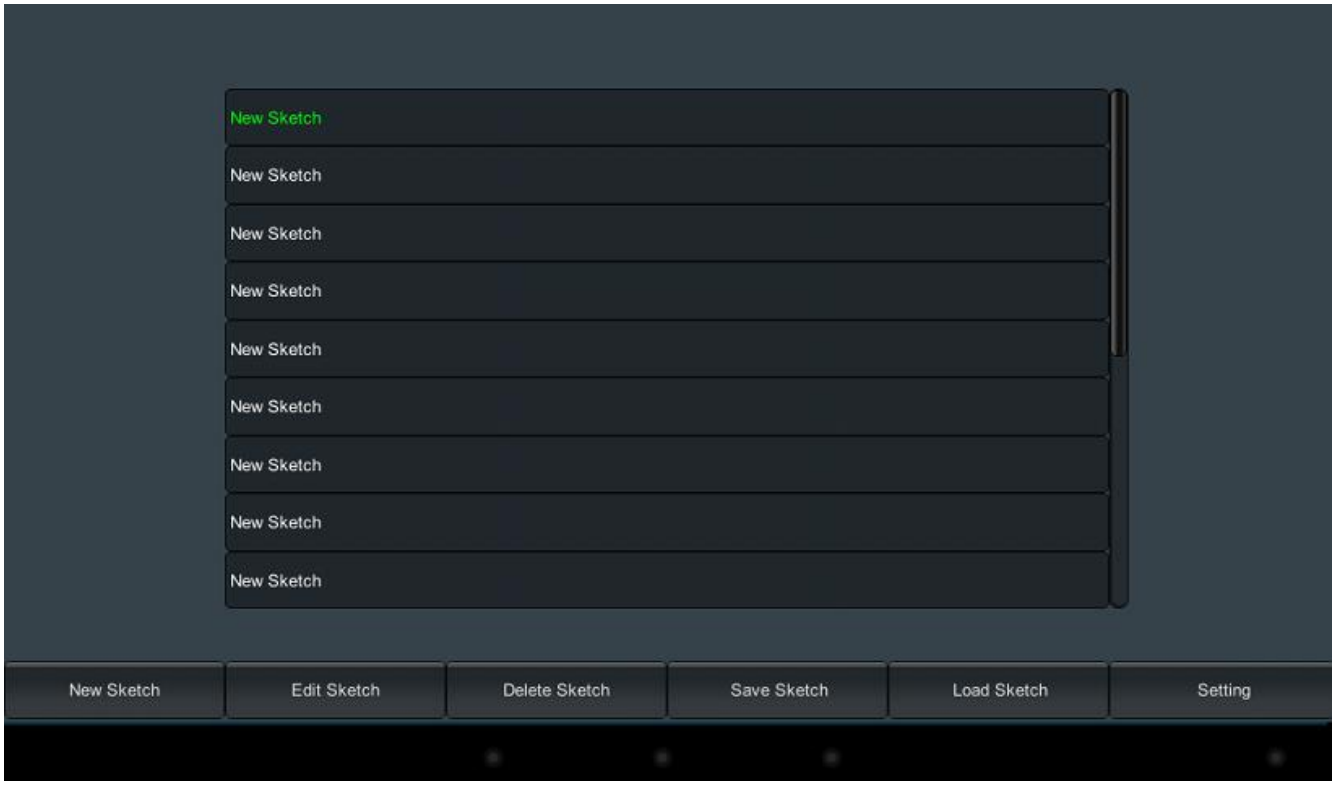


Figura 7 Interfaz principal

Para cada valor de entrada o acción realizada, se recibió el valor esperado. A continuación se muestra para cada acción realizada a mano, la curva ajustada. Todas las pruebas realizadas de ajustes de curva arrojaron resultados satisfactorios.

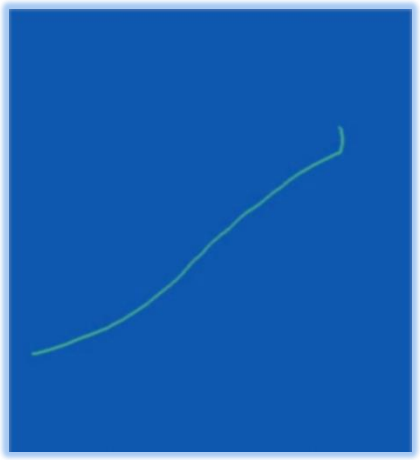


Figura 8 Trazo de línea a mano

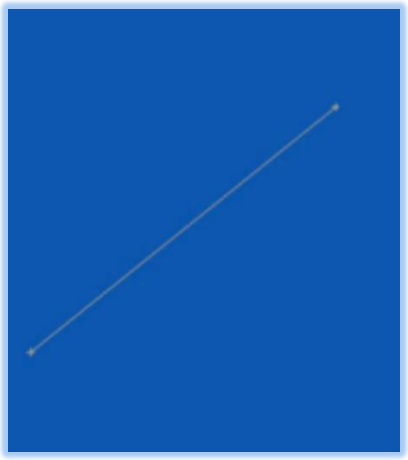


Figura 9 Ajuste para la línea

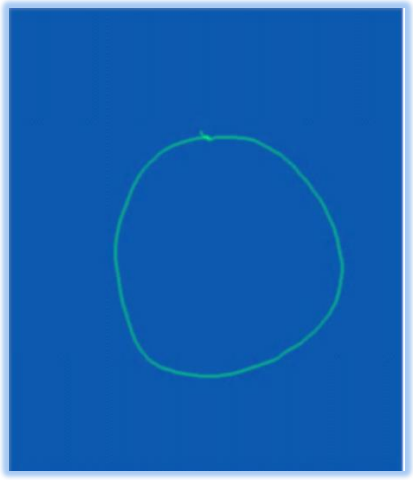


Figura 10 Trazo de un círculo a mano

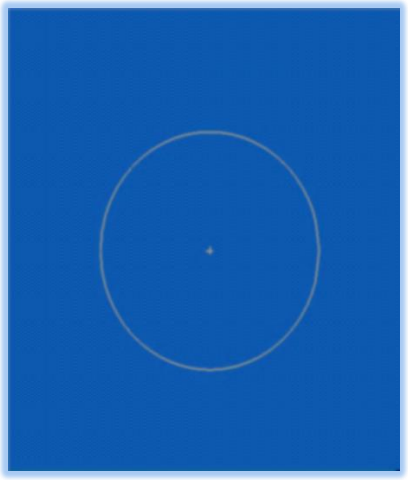


Figura 11 Ajuste de un círculo

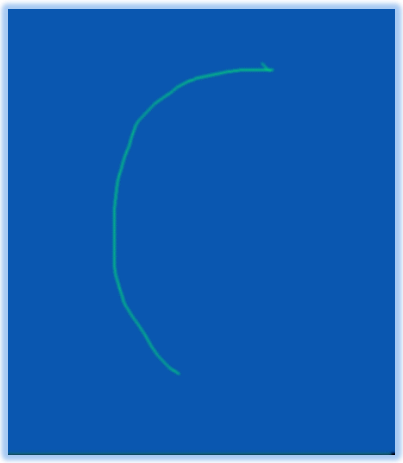


Figura 12 Trazo de un arco a mano

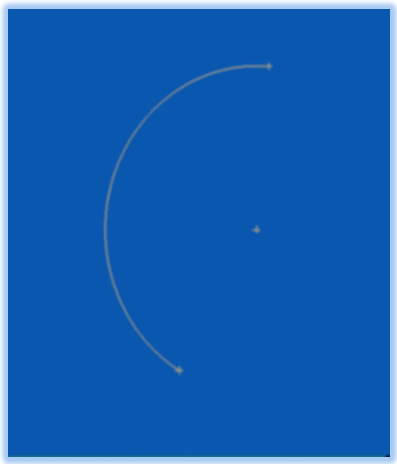


Figura 13 Ajuste de un arco

Figura 14 Trazo de una elipse a mano

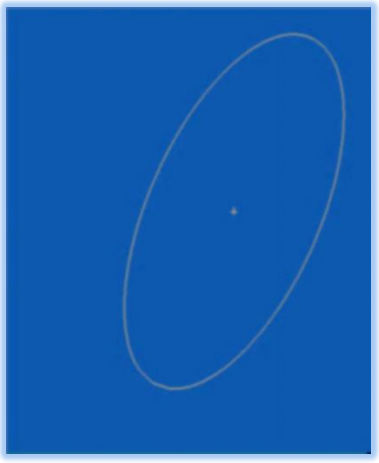


Figura 15 Ajuste de una elipse

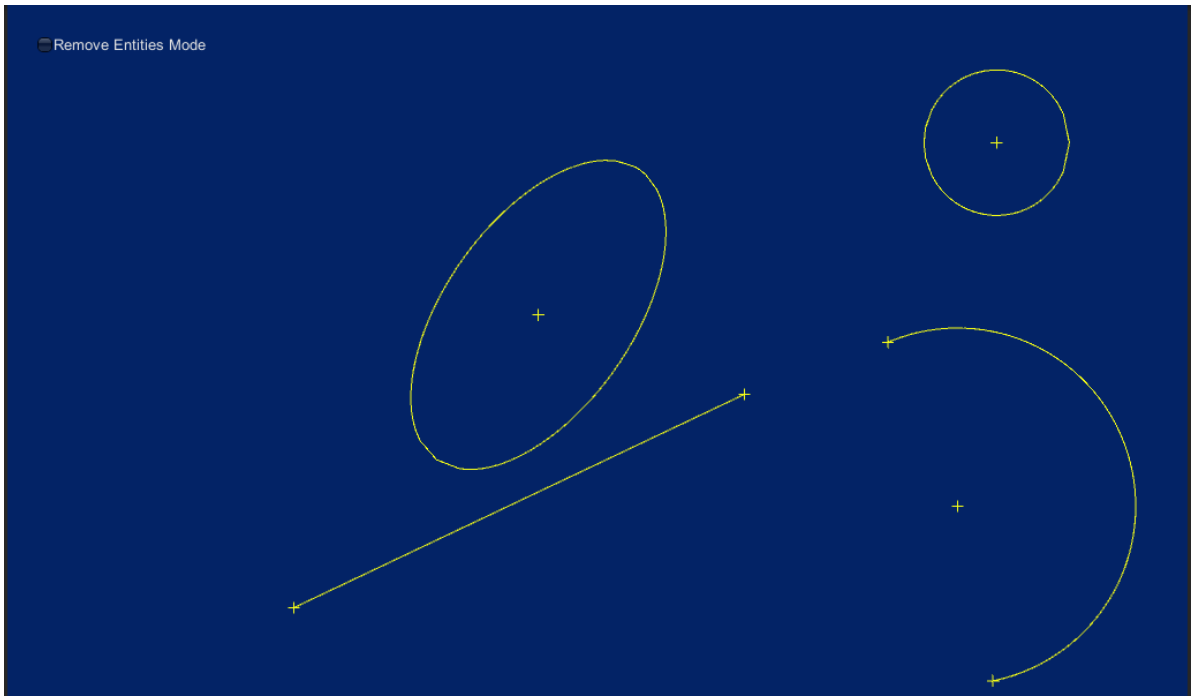


Figura 16 Boceto

El sistema brinda además la posibilidad de realizar configuraciones para establecer los colores (azul, verde, amarillo y gris) de las curvas, del fondo de la aplicación, en donde se realizarán los bocetos. También se podrán configurar los errores para cada una de las entidades tratadas. Ver **Anexo 2**

3.4.1 Resultados obtenidos

- El sistema mostró ser completamente funcional, dando solución a todos los requisitos planteados.
- Todas las pruebas dieron resultados satisfactorios, de esta manera se demuestra que el sistema cumple con cada uno de los CU que lo conforman.

3.5 Consideraciones parciales del capítulo

Culminada la implementación de la solución propuesta y luego de realizar pruebas al sistema, se puede llegar a las siguientes consideraciones:

- ✓ La modelación de diagramas relacionados con el análisis y diseño crearon las bases para la implementación del sistema.
- ✓ Los diagramas de componentes y de despliegue de la solución propuesta permitieron la organización de los componentes y del nodo físico sobre los cuales funcionará el sistema.
- ✓ La realización de las pruebas constituyen la certificación de que el sistema soporta las funcionalidades requeridas.
- ✓ Los resultados de las pruebas demuestran la solidez de la implementación del sistema.

CONCLUSIONES GENERALES

1. El estudio de las herramientas basadas en bocetos a mano alzada permitió conocer su importancia, así como, la necesidad de creación de una nueva aplicación para el sistema operativo Android que propicie la disminución y corrección de errores en los diseños de fabricación.
2. Se generaron los diagramas necesarios para el diseño de la nueva aplicación.
3. El desarrollo de la aplicación Android *FreeHand*, permite a los especialistas de la ICVT desarrollar bocetos geométricos conceptuales a mano alzada en el campo y en las estaciones de trabajo.
4. La aplicación *FreeHand* desarrollada garantiza su integración con el *software* GALBA-CAD mediante las funcionalidades de importar y exportar bocetos en el formato skt.

RECOMENDACIONES

- ✓ Adicionar al sistema la posibilidad de realizar *snap* entre entidades.
- ✓ Incorporar al sistema el ajuste de la entidad arco de elipse.
- ✓ Compilar la herramienta desarrollada para otros sistemas operativos móviles.
- ✓ Incorporar a la herramienta soporte para varios lenguajes.

REFERENCIAS BIBLIOGRÁFICAS

1. Scribd. *Scribd.* [Online] [Cited: enero 8, 2014.] http://es.scribd.com/doc/17754860/Sistema-CAD.

2. **Costa, José Antonio Velásquez.** *ComputerIntegratedManufacturing CIM. .* Lima, Perú: Universidad Ricardo Palma: s.n., 2009.

3. **SIEMENS.** SIEMENS. [Cited: enero 12, 2014.] http://www.plm.automation.siemens.com/es_sa/plm/cam.shtml.

4. **SIEMENS.** SIEMENS. [Cited: enero 12, 2014.] http://www.plm.automation.siemens.com/es_sa/plm/cad.shtml.

5. **SIEMENS.** SIEMENS. [Cited: enero 12, 2014.] http://www.plm.automation.siemens.com/es_sa/plm/cae.shtml.

6. CAD. *CAD.* [Online] [Cited: enero 12, 2014.] http://www.mitecnologico.com/Main/DisenoAsistidoPorComputadora.

7. Revista Informática. *Revista Informática.* [Online] [Cited: enero 14, 2014.] http://www.larevistainformatica.com/DISENO-ASISTIDO-COMPUTADORA.HTML.

8. AutoCAD.PlataformaArquitectura. *AutoCAD.PlataformaArquitectura.* [Online] [Cited: enero 16, 2014.] http://www.plataformaarquitectura.cl/2012/02/04/autocad.

9. Inventor - *Software* CAD 3D y de Diseño Mecánico 3D. Autodesk. *Inventor - Software CAD 3D y de Diseño Mecánico 3D. Autodesk.* [Online] [Cited: enero 21, 2014.] http://latinoamerica.autodesk.com/adsk/servlet/pc/index?id=14601337&siteID=7411870.

10. Ashampoo. Ashampoo.com. *Ashampoo. Ashampoo.com.* [Online] [Cited: enero 22, 2014.] http://www.ashampoo.com/es/eur/pin/0460/CAD_Construccion/Ashampoo-3D-CAD-Architecture-3.

11. GenBeta. *GenBeta.* [Online] [Cited: enero 24, 2014.] http://www.genbeta.com/herramientas/librecad-un-programa-sencillo-para-iniciarse-en-el-mundodel-cad.

12. Qcad. *Qcad.* [Online] [Cited: enero 23, 2014.] http://www.qcad.org/.

13. Arquitectura CAD. *Arquitectura CAD.* [Online] [Cited: enero 26, 2014.] http://www.arquitectura.com/cad/artic/elcad.asp.

14. **Madero, Nelson.** [Online] [Cited: enero 20, 2014.] http://www.nelsonmadero.com/pages/studiotip005.php?lang=es/.

15. **Frascara, Jorge.** *Diseño Gráfico y Comunicación.* s.l. : s.l.: Séptima edición, 2000. . ISBN/9879637054.

16. Blog Informático. *Blog Informático.* [Online] [Cited: enero 25, 2014.] http://www.scribd.com/doc/54444984/conceptos-del-diseno-bidimensional.

17. **en:, [En línea] [Consultado el: 7 de Febrero de 2014]Disponible.**CristaLab. *CristaLab.* [Online] [Cited: enero 25, 2014.] http://www.cristalab.com/tutoriales/fundamentos-del-diseno-grafico-c126l/.

18. **M. Grajales y P. Susano**, «Repositorio Institucional Universidad Veracruzana,» 2013. [En línea]. http://cdigital.uv.mx/handle/123456789/34448.

19. **Vilchez, Angel.** Qué es Android: Características y Aplicaciones. *Qué es Android: Características y Aplicaciones.* [Online] [Cited: febrero 5, 2014.] http://www.configurarequipos.com/doc1107.html.

20. **Pressman, Roger S.***"Ingeniería del Software. Un enfoque práctico". 6ta Edición. Parte IV.* 2007.

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

22. **UML, Sitio oficial de.** Introducción a UML 2.0. *Introducción a UML 2.0.* [Online] [Cited: febrero 6, 2014.] http://www.uml.org/.

23. Modelado de Sistemas con UML. *Modelado de Sistemas con UML.* [Online] [Cited: febrero 5, 2014.] http://es.tldp.org/Tutoriales/doc-modelado-sistemas-UML/doc-modelado-sistemas-uml.pdf.

24. **Jacobson, Ivar, Booch, Grady y Rumbaugh, James.**El lenguaje Unificado de Modelado*.* Madrid: Addison-Wesley Professional, 2005.

25. Technologies, Unity. MotoresGráficos. [Online] 2011. [Cited: febrero 7, 2014.] http://www.mat.ub.edu/futurs_ub/activitats/Matefest/2011/triptics/motoresgraficos.pdf..

26. Unity3D. *Unity3D.* [Online] [Cited: enero 28, 2014.] http://www.unity3d.com/.

27. **Pathfinding, Navmesh and.** Unity Technologies. *Unity Technologies.* [Online] 2010. [Cited: enero 28, 2014.] http://docs.unity3d.com/Documentation/Manual/NavmeshandPathfinding.html..

28. **JavaScript., Sitio oficial de.** JavaScript. *JavaScript.* [Online] [Cited: febrero 3, 2014.] http://www.maestrosdelweb.com/editorial/¿que-es-javascript/.

29. **Ivar Jacobson, G.B., James Rumbaugh,** *El proceso unificado de desarrollo de software. Edición en español ed.* 2000.

30. **Pressman, R.,** *Ingeniería de software, Un enfoque práctico.* 2007-2008.

31. **Cooper, J.W.** *Introduction to Design Patterns in C#.* 2002.

32. **LARMAN, C,** *UML Y PATRONES INTRODUCCION AL ANALISIS Y DISEÑO ORIENTADO A OBJETOS.* s.l. : PRIMERA EDICION, 1999

33. **Prieto, Félix. 2009.** *Patrones de diseño.* España: Departamento de Informática. Universidad de Valladolid, 2009.

34. **Model-View-Controller Architecture Overview**. s.l. : CyberFicient Technologies, 2002.

35. **GARCÍA LIRA, K. G. D., TEJERA HERNANDEZ, AILEC**. *Conferencia #1: Continuación de la Disciplina Análisis y Diseño*. En Ingenieria de *software* II. Departamento de ISW, Universidad de las Ciencias Informáticas, Curso: 2010 - 2011. p. 56.

36. **Tema II.***Fase de Construcción. En Conferencia 6. Disciplina de Implementación*. Cuba, Universidad de las Ciencias Informáticas*.2011*. p. 45.

37. **PRESSMAN, R. S**. *Estrategia de prueba del software. En Ingenieria del Software, Un Enfoque Práctico*. 6 ed. España: Editorial McGraw-Hil, 2002, p. 640.

GLOSARIO DE TÉRMINOS

A

Aplicación: En informática es un tipo de *software* diseñado para facilitar al usuario la realización de una determinada tarea o trabajo.

F

Framework: es una estructura conceptual y tecnológica de soporte definido, generalmente, con artefactos o módulos de *software* concretos, con base a la cual otro proyecto de *software* puede ser más fácilmente organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas y un lenguaje interpretado, entre otras herramientas, para así ayudar a desarrollar y unir los diferentes componentes de un proyecto. Los *frameworks* son herramientas de desarrollo que facilitan infraestructuras que ahorran horas de trabajo.

H

HTML: Lenguaje de Marcas de Hipertexto (*HyperText Markup Language*). Es usado para describir la estructura y el contenido en forma de texto, así como para complementar el texto con objetos tales como imágenes. HTML se escribe en forma de "etiquetas", rodeadas por corchetes angulares (<,>). HTML también puede describir, hasta un cierto punto, la apariencia de un documento, y puede incluir un script (por ejemplo *Javascript*), el cual puede afectar el comportamiento de navegadores web y otros procesadores de HTML.

I

IDE: Es un entorno de programación que ha sido empaquetado como un programa de constructor de interfaz gráfica de usuario (GUI). Los IDEs pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes.

IPhone: De la compañía *Apple Inc.*, es un teléfono inteligente multimedia que permite la conexión a Internet por medio de una red inalámbrica, posee pantalla táctil y una interfaz de hardware despojada de elementos sobrantes.

N

Navegador: Un navegador web o cliente HTTP, es un programa que permite interpretar la información y el código que contiene una página *web* (esté alojada en un servidor dentro de la *World Wide Web* o en uno local) y presentarla de manera legible a los usuarios.

T

Telefonía celular: También llamada telefonía móvil, básicamente está formada por dos grandes partes, una red de comunicaciones y los terminales que permiten el acceso a dicha red.

S

Script: Es un guion o conjunto de instrucciones. Permiten la automatización de tareas creando pequeñas utilidades. Es un trozo de código que puede recibir argumentos y devolver un valor. Los scripts se utilizan para generalizar código repetido y así ahorrar espacio y ganar velocidad.

Smartphone: Es un dispositivo electrónico que funciona como teléfono celular con características similares a las de un computador personal. Los teléfonos inteligentes permiten la instalación de programas para incrementar el procesamiento de datos y la conectividad.

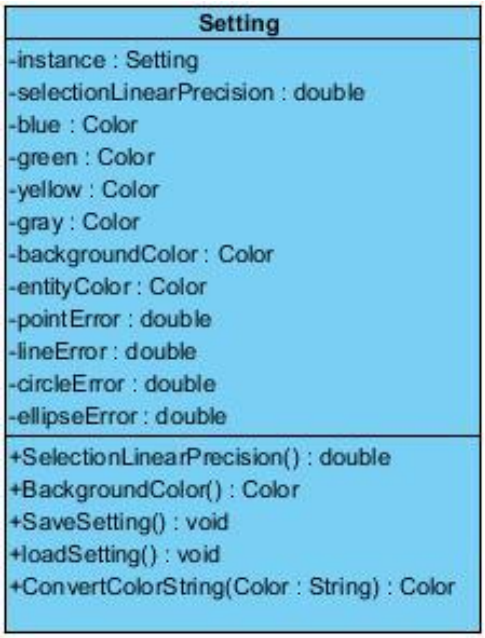
SMS (*Short Message Service*): Es un servicio de mensajería por teléfonos celulares. Con este sistema se puede enviar o recibir mensajes entre celulares y, luego, a través de internet.

X

XML: Lenguaje extensible de marcas (*Extensible Markup Language*), es un metalenguaje extensible de etiquetas desarrollado por el *World Wide Web Consortium*. Su objetivo es conseguir páginas *Web* más semánticas. XML separa la estructura del contenido y permite el desarrollo de vocabularios modulares. Se trata de un formato abierto.

ANEXOS

Anexo 1 Clase Setting para realizar configuraciones.



Anexo 2 Configurar aplicación