Universidad de las Ciencias Informáticas

Facultad 4

Funcionalidades básicas para el modelado en tres dimensiones de componentes genéricos, para su empleo en sistema de diseño asistido por computadoras.

Autor:

Raciel Perdomo Gómez

Tutores:

Dr.C Augusto Cesar Rodríguez Medina

Ing. Sandy García Santos

La Habana, julio de 2017

“Año 59 de la Revolución”

Declaración de autoría

Declaro ser único autor del presente trabajo de diploma y autorizo a la Universidad de las Ciencias Informáticas a ser uso del mismo en su beneficio.
Para que así conste firmo la presente a los ____ días del mes de _____ del año 2016.

_____
Raciel Perdomo Gómez
Firma del Autor

_____ _____
Dr.C Augusto Cesar Rodríguez Medina Ing. Sandy García Santos
Firma del Tutor Firma del Tutor

El logro más impresionante de la industria del software es su continua anulación de los constantes y asombrosos logros de la industria del hardware.

Henry Petroski

Dedicatoria

A Linita, mi madre de crianza, que, aunque no esté físicamente conmigo, si lo está espiritualmente cada segundo que respiro.

Agradecimientos:

A mis padres por todo su esfuerzo y dedicación, por servirme de faro y guía a lo largo de mi vida. A mis compañeros en general (especial para Down, Frank, Pepin, Anselmo, Domador), a mis tutores por la dedicación mostrada. A los profesores que tuve a lo largo de mis 5 años de estudio. A Manolo, Nesty y Pututi, los verdaderos sobrevivientes (no enseñen nunca el video). Al ACM por ser el club de mis amores y servirme como fuente de inspiración. A Silvio por todo lo logrado y a Li Yonghong por dar el paso adelante. A SES92. A los compañeros de la ACM, en especial a Eduin por ser el que me inició en el lindo mundo de la programación.

Resumen

Los módulos de piezas son aplicaciones informáticas que facilitan el diseño de piezas y componentes genéricos en tres dimensiones. Son ampliamente utilizados en diversas ramas de la industria y posibilita crear las piezas y componentes de una forma rápida y con alta precisión matemática. La investigación contiene los resultados del proceso de desarrollo de un conjunto de funcionalidades básicas para la creación de piezas genéricas, en el cual se utiliza la tecnología “Open CASCADE”, el framework Qt y el lenguaje de programación C++; con el requerimiento de una arquitectura flexible que facilite el empleo del módulo, para poder utilizarlo en una aplicación para el Diseño Asistido por Computadora (CAD por las siglas de “Computer Aided Design”). Para el desarrollo de la investigación fue necesario analizar soluciones similares de código abierto, y la bibliografía disponible de herramientas propietarias. Se analizaron diferentes herramientas y tecnologías de desarrollo. Finalmente, se obtuvo una solución que responde a los objetivos planteados en la investigación.

Palabras clave: framework, funcionalidades, modelado, paramétrico, pieza, sólido.

Índice

Introducción

La presente investigación sienta sus bases un proyecto del Grupo de Investigación “Soluciones Informáticas para la Ingeniería y la Industria” (SIPII) perteneciente al Departamento de Ciencias Básicas de la Facultad 4, en la Universidad de las Ciencias Informáticas; la estructura del documento contiene los resultados del proceso de investigación y desarrollo, asociado a integrar un conjunto de funcionalidades que permitan el modelado paramétrico en tres dimensiones de piezas genéricas en una aplicación para el Diseño Asistido por Computadora.

La idea del mencionado desarrollo parte de una exigencia de proyecto de agrupar un conjunto de funcionalidades para el modelado de piezas en tres dimensiones, que permita resolver las dificultades que enfrentan los ingenieros que diseñan piezas genéricas en la industria nacional, asociados a las restricciones del bloqueo económico y comercial impuesto por el gobierno de los Estados Unidos de América contra Cuba.

Las funcionalidades para el modelado en tres dimensiones se desarrollaron mediante el uso de un estilo arquitectónico que garantiza una alta cohesión, bajo acoplamiento informático, robustez y eficacia.

Se utilizaron tecnologías de código abierto como la versión comunitaria de “Open CASCADE”, el marco de trabajo Qt y el código fuente de la aplicación “FreeCAD”, con el objetivo de garantizar una autonomía e independencia en el producto desarrollado. Se ejecutó el proceso transitando desde los aspectos más simples a los más complejos haciendo un empleo sistemático, en lo fundamental, del método teórico de análisis y síntesis y el empírico de observación.

En el proceso de delimitación y recolección de información se consultaron numerosas fuentes electrónicas e impresas, particular importancia tuvo el estudio de las Interfaces de Programación de Aplicaciones (“Application Programming Interface” o API), así como los tutoriales orientados a la creación de piezas genéricas, disponibles en los foros de la comunidad de “FreeCAD”.

El documento está estructurado en tres capítulos; en el primero se exponen los aspectos generales de la fundamentación teórica del trabajo, iniciando con la conformación del perfil de la investigación; se incluyen además los requerimientos generales de un módulo para el modelado de piezas genéricas y las tecnologías utilizadas para el desarrollo de las funcionalidades de modelado. En el segundo capítulo se confecciona la propuesta de

solución, para lo que se parte del modelo del dominio, se exponen los requisitos capturados, se realiza la descripción de las funcionalidades, se define la arquitectura a utilizar y se establecen los patrones de diseño. En el tercero se muestran los resultados obtenidos de las etapas de implementación y de pruebas. Se concluye el trabajo relacionando las conclusiones y recomendaciones que se fueron captando durante la investigación.

Capítulo 1: Fundamentación teórica de la investigación

La investigación se corresponde con un grupo de investigación destinado a obtener un sistema informático para el modelado de componentes y piezas mecánicas en dos y tres dimensiones, este tipo de aplicación CAD es utilizado por los ingenieros de diversos perfiles. Para hacer estas tareas actualmente se utilizan sistemas propietarios, los más difundidos en Cuba son “SolidEdge”, “AutoCAD” e “Inventor” (1).

Como se verá en el apartado 1.1 cuando se profundiza en la situación problemática que da origen a la investigación, existen dificultades que conllevan a la necesidad de remplazar los mencionados sistemas para evitar conflictos por el uso indebido de éstos.

Como se trata de sistemas para el modelado de componentes y piezas genéricas, se hace necesario conocer varios aspectos de esa actividad para comprender su importancia en el ámbito de la ingeniería.

1.1 Situación problemática.

Las tecnologías para el diseño asistido por computadoras, en las últimas cinco décadas, se han generalizado como herramienta necesaria para el trabajo de ingenieros, arquitectos y diseñadores. Estas herramientas se dividen básicamente en programas de dibujo en dos dimensiones (2D) y de modelado en tres dimensiones (3D). Las primeras se basan en el trazado de entidades geométricas vectoriales como puntos, líneas, arcos y polígonos, con las que se puede operar a través de una interfaz gráfica; mientras que las segundas se sustentan en entidades geométricas como planos, superficies y sólidos en tres dimensiones.

En el grupo de investigación “Soluciones Informáticas para la Ingeniería y la Industria” (SIPII), se desarrollan actualmente componentes para aplicaciones de Diseño Asistido por Computadoras y **como resultado de las necesidades de proyecto se requiere disponer de funcionalidades para el modelado de piezas genéricas**; como regla, las aplicaciones comerciales existentes cuentan con un módulo que agrupa funcionalidades para ejecutar este tipo de tarea.

Algunas de las herramientas informáticas comerciales que incluyen funcionalidades para el modelado de piezas son “Autodesk Inventor”, “Solid Edge” y “Solidworks”, como son propietarios, los usuarios no son sus dueños, sino que reciben autorización para su uso por un tiempo cuando adquieren la licencia correspondiente; estos sistemas también se de denominan MCAD (“Mechanical Computer Aided Design”) y se distingue en ellos como

característica, la existencia de módulos dedicados al modelado de piezas, conformación de ensambles y de dibujo, los que concentran las funcionalidades requeridas por cada uno de esos escenarios.

Las restricciones impuestas por el bloqueo de los EE. UU, impiden que Cuba acceda a este tipo de producto, lo que dificulta la introducción de tecnologías de avanzada destinadas al diseño para la ingeniería. En la práctica los ingenieros de las industrias nacionales emplean sistemas CAD, pero no de forma institucional, son de dudosa procedencia y constituyen un serio problema de seguridad nacional; los diseños resultantes del trabajo con sistemas "crackeados" no pueden ser comercializados por que violan las leyes del "Copyright".

En software libre existen desarrollos alternativos como "LibreCAD" y "FreeCAD", el primero para el dibujo en 2D y el segundo para el modelado 3D con funcionalidades para la creación de piezas y componentes genéricos. Una característica de "FreeCAD" es que su arquitectura está concebida de manera que las funcionalidades para el modelado de piezas, se encuentran dispersas en tres módulos denominados "Part", "Part Design" y "Draft", lo que afecta la productividad del diseñador durante el proceso, pues tiene que buscar en varios escenarios diferentes lo que necesita en la sesión de trabajo; las operaciones para el modelado en los módulos referidos se basan en el "kernel" gráfico **"**Open CASCADE**"**.

El módulo "Part Design" provee funcionalidades para modelar sólidos a partir de un boceto paramétrico en el módulo "Sketcher" (2); el "Part" contiene funcionalidades para la creación de primitivas y operaciones booleanas; y con el módulo "Draft" es posible dibujar rápidamente objetos simples en dos dimensiones y ofrece herramientas para modificarlos posteriormente, también proporciona un sistema de pinzamiento o "snapping" y utilidades para administrar objetos y configuraciones (3).

En "FreeCAD" se emplean además, dos de los métodos para el modelado que se encuentran implementados en los sistemas comerciales: el "History Based Feature" y la parametrización, paradigmas que pueden ser reutilizados en desarrollos propios pues lo permite la licencia LGPL con la que es distribuido (2).

Otra aplicación que permite el modelado de piezas es AsiXMec, desarrollada en la Universidad de Ciencias Informáticas. Su enfoque de modelación paramétrica es basado en figuras e historial de operaciones (4), pero por decisiones de proyecto se decide estudiar otras herramientas para lograr el desarrollo de la solución.

Aparejada a la exigencia de proyecto mencionada en el segundo párrafo de este apartado, está asociada la realidad de que en la aplicación “FreeCAD”, las funcionalidades están dispersas; las que inicialmente se requieren son las básicas, entiéndase, las de modelado en tres dimensiones a partir de un “Sketcher” 2D con funcionalidades para la parametrización.

En base a lo anterior, se plantea el siguiente **problema de investigación**:
¿Cómo agrupar las funcionalidades básicas para el modelado de piezas genéricas, mediante la reutilización de las funcionalidades contenidas en los módulos “Part”, “Part Design” y “Draft” de la aplicación “FreeCAD”?

Para solucionar el problema planteado se propone como **objetivo general**: **Desarrollar un conjunto de funcionalidades, para el modelado de piezas genéricas y componentes mecánicos en tres dimensiones**.

El **objeto de estudio** de la investigación se centra en el modelado de entidades, teniendo como **campo de acción**, las funcionalidades básicas para el modelado de piezas y componentes genéricos en tres dimensiones.

A partir del objetivo general definido, se derivan los siguientes **objetivos específicos**:

- Definir las funcionalidades básicas requeridas para el modelado de piezas genéricas en tres dimensiones.
- Diseñar las funcionalidades básicas requeridas para el modelado de piezas genéricas en tres dimensiones.
- Implementar funcionalidades básicas para realizar el modelado paramétrico de piezas y componentes mecánicos en tres dimensiones.
- Realizar pruebas al conjunto de funcionalidades implementadas.

Para dar cumplimiento a los objetivos específicos se proponen como **tareas de investigación**:

- Asimilación de los conceptos y tecnologías requeridos para el desarrollo de las funcionalidades.
- Definición del perfil y diseño de la investigación.
- Búsqueda y revisión bibliográfica sobre el objeto de investigación.
- Fundamentación de las necesidades y requerimientos de proyecto que motivan la ejecución del trabajo (Conjunto de funcionalidades para un módulo de pieza).

- Análisis de los códigos fuentes de las aplicaciones “FreeCAD” (módulos “Part”, “Part Design” y “Draft”) y “Salome-Meca” (Módulo “Geometry”);
- Comparación de las funcionalidades de módulos similares existentes en aplicaciones comerciales para el diseño y la ingeniería asistidos por computadoras (sistemas CAD) con los existentes en aplicaciones de código abierto como “FreeCAD” y “Salome-Meca”.
- Estudio de los aspectos de la metodología de desarrollo de software (AUP-UCI) que se aplicarán en la investigación.
- Obtención de requisitos a partir de los módulos de piezas existentes en sistemas propietarios y de código abierto, así como consideraciones de los potenciales usuarios.
- Definición de la arquitectura del módulo (definición de las funcionalidades para el modelado en tres dimensiones) lo que implica además definir los patrones de diseño con los que cumplirá.
- Determinación de la estructura de clases del módulo que contendrá las funcionalidades.
- Implementación de las funcionalidades.
- Integración del módulo “Sketcher” a la estructura del conjunto de funcionalidades.
- Integración de las funcionalidades en la rama principal de la aplicación en desarrollo.
- Comprobación del módulo implementado (pruebas de aceptación y unitarias).

Las funcionalidades a desarrollar son:

- Extrusión.
- Vaciado.
- Espesor.
- Revolución.
- Redondeado.
- Chaflanado.
- Superficie reglada.
- Reflejar característica.
- Crear “Sketch” con soporte en la cara plana de un sólido.
- Operación booleana (Unión, Diferencia, Intersección, Sección).

Con el fin de resolver y dar cumplimiento a los objetivos y las tareas propuestas se empleó como **métodos de investigación**:

Métodos teóricos:

Análisis y síntesis: El análisis se aplicó durante el proceso de búsqueda de información, la captura de requisitos, en el estudio y reutilización de los códigos de las aplicaciones "FreeCAD" y "SALOME-Meca"; la síntesis se empleó durante el proceso de arribo a conclusiones parciales y finales, así como durante la toma de decisiones en relación a las tareas de desarrollo, por ejemplo, la definición de las funcionalidades a implementar.

El análisis se realizó en diversas variantes, documental, análisis de código fuente, e histórico lógico.

Empíricos:

Experimentación: En las pruebas realizadas a las funcionalidades.

1.2 Características y requerimientos generales de las funcionalidades para el modelado de piezas genéricas a integrar en una aplicación CAD.

Las funcionalidades para el modelado en tres dimensiones de piezas genéricas, debe garantizar la construcción de modelos geométricos y topológicos de piezas en tres dimensiones, con facilidades para la generación y modificación de rasgos característicos ("History Based Feature") y la parametrización; estos son métodos o paradigmas que se implementan en los sistemas CAD desde que en 1987 apareció el "Pro Engineer", primer sistema con estas características desarrollado por la empresa "Parametric Technology Corporation" (PTC) (5). Estos paradigmas se encuentran en todos los sistemas CAD modernos, los que se emplean en los procesos de diseño en diversas ramas de la ingeniería como la mecánica, industrial, civil, aeronáutica, naval, de estructuras y otras líneas afines tanto en el ámbito civil como el militar.

En un modelo paramétrico, cada entidad posee parámetros asociados, que controlan la configuración geométrica de estos, por ejemplo, el largo, ancho y alto de un prisma rectangular, o el radio de un "fillet" son parámetros; estos pueden ser cambiados por el usuario para crear la pieza deseada. Los modelos paramétricos funcionan conectados además, al paradigma "History Based Feature", mediante este mecanismo es posible almacenar la información de cómo fue construido el modelo y visualizar tales operaciones en forma de árbol, usuario puede cambiar los nodos del árbol para alterar sus rasgos geométricos que también pueden estar parametrizados (6). Los usuarios emplean estos

métodos para variar configuraciones de diseños existentes sin tener que modelarlos desde cero. La importancia que tienen los paradigmas referidos en las aplicaciones CAD es la incidencia directa en el aumento de la productividad en el proceso de diseño de piezas. Otros beneficios de las tecnologías CAD es que permiten manejar con elevada precisión los modelos, así como realizar cálculos y numerosas operaciones de forma automática. En el siguiente apartado se profundiza en la descripción de los módulos que contienen funcionalidades para el modelado en tres dimensiones, de piezas genéricas.

1.2.1 Descripción general de los módulos de pieza genéricos en los sistemas para el Diseño Asistido por Computadoras.

Los módulos de pieza genéricos por lo general parten de un boceto en dos dimensiones o “sketch” en el cual se conforma un perfil característico de la pieza paramétricamente, y una vez aplicadas las restricciones necesarias se le aplican operaciones para convertirlo en un modelo en tres dimensiones; así se van esculpiendo las diferentes regiones de una pieza.

La extrusión es una de las operaciones que se le aplica al perfil parametrizado, la cual extiende una forma en una distancia y dirección especificada. A continuación, se muestra el tipo de forma geométrica de salida esperada de un tipo de forma geométrica de entrada dado, en los escenarios más comunes:

- Extruir un vértice (punto), produce un borde lineal (línea).
- Extruir un borde abierto (línea, arco), produce una cara abierta (plano).
- Extruir un borde cerrado como un círculo produce opcionalmente una cara cerrada (por ejemplo, un cilindro de extremo abierto), o un cilindro sólido cerrado.
- Extruir una cara (por ejemplo, un plano), produce un sólido (por ejemplo, Ortoedro).

“FreeCAD” en su versión 16, obtiene los parámetros de extrusión mediante una interfaz con forma de diálogo, a diferencia de herramientas comerciales como “Solid Edge”, donde se utiliza un manejador a través del ratón; y la distancia de extrusión se construye de forma aproximada por el usuario, para después ajustarla manualmente mediante una vista (7).

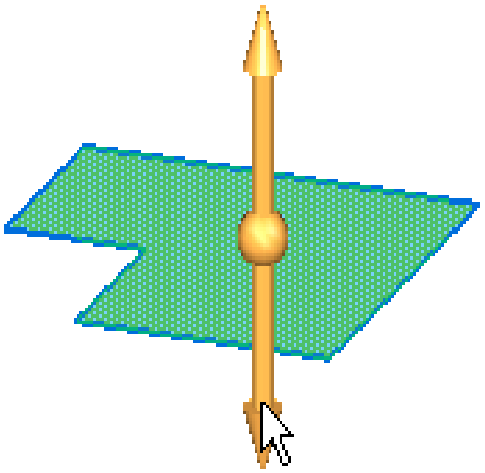


Figura 1: Manejador de Extrusión en Solid Edge.

Otra operación que se le aplica al perfil parametrizado para obtener un sólido es la revolución. Se denomina sólido de revolución o volumen de revolución, al sólido obtenido al rotar una región del plano alrededor de una recta ubicada en el mismo, las cuales pueden o no cruzarse. Dicha recta se denomina eje de revolución (8). En principio, cualquier cuerpo con simetría axial o cilíndrica es un sólido de revolución (8).

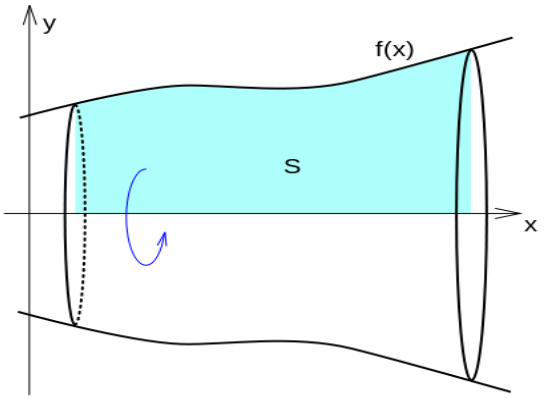


Figura 2: Operación de revolución de una forma sobre el eje x.

A continuación, se muestra el tipo de forma de salida esperada, al aplicar la operación de revolución sobre un tipo de forma dado:

Tabla 1: Forma de salida esperada luego de aplicar la operación de revolución.

Forma de entrada	**Forma de salida**
“Vertex”	“Edge”

“Edge”	“Face”
“Wire”	“Shell”
“Face”	“Solid”
“Shell”	“Compound solid”

En “FreeCAD” la operación de revolución puede arrojar resultados inválidos si se desea obtener un sólido y el eje utilizado intersecta la cara a rotar.

Reflejar característica, otra de las operaciones sobre formas parametrizadas, produce un nuevo objeto el cual es una reflexión del original. El objeto imagen se crea detrás de un plano de espejo. En “FreeCAD” dicho plano tiene que ser el estándar (XY, XZ, o YZ) o cualquier plano paralelo a un plano estándar. Esta operación posee como limitante que no se puede seleccionar un plano arbitrario (plano no paralelo a un plano estándar) (3).

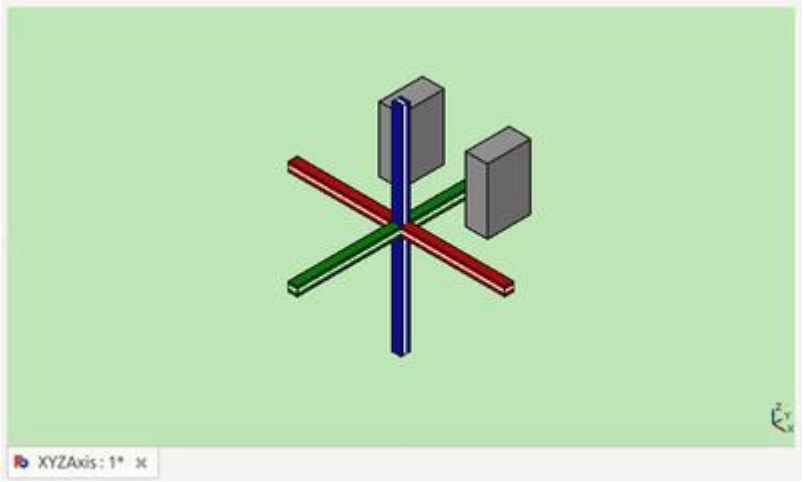


Figura 3: Reflejar sólido a través del plano YZ.

Los módulos de pieza genéricos, como regla, contienen geometrías de construcción, las que facilitan el proceso de modelado en tres dimensiones durante la construcción de rasgos complejos de la pieza, como pueden ser un orificio con un ángulo de inclinación determinado; estas geometrías son planos, líneas y puntos, no forman parte de la pieza, solamente se emplean como instrumentos constructivos y en general se implementan diversas formas para realizarlos, por ejemplo, un plano de construcción se realiza con un punto y una línea, con tres puntos, con dos líneas; de igual manera ocurre con las líneas y puntos de construcción.

Algunos módulos de piezas como en el caso de “SolidWorks” contienen cajas de herramientas que se habilitan solamente cuando se necesitan sus funcionalidades; en los sistemas propietarios los módulos de piezas pueden activarse desde un ensamble, de modo que basado en una pieza concreta se pueden editar las características de la misma sin salir del módulo de ensamble para entrar nuevamente en el de pieza; gracias a la parametrización y al historial de rasgos, las piezas modificadas se pueden actualizar a su último estado.

Existen funcionalidades en los sistemas CAD basadas en una técnica denominada Geometría Constructiva de Sólidos que emplea operaciones booleanas para conformar sólidos, las tres funcionalidades básicas son:

- **Fusión:** unión de dos objetos.
- **Intersección:** Extrae la parte común (intersección) de dos objetos.
- **Diferencia:** Corta (resta) un objeto de otro.

En “FreeCAD”, para realizar las operaciones de fusión, intersección y diferencia de dos sólidos, hay que tener en cuenta que dichos sólidos deben ser obtenidos con “Open CASCADE”. La tecnología “Open CASCADE” es una librería de clases de C++ designada para la producción rápida de sofisticadas aplicaciones para el CAD, la Fabricación Asistida por Computadora (CAM por las siglas de “Computer Aided Manufacturing”), y la Ingeniería Asistida por Computadora (CAE por las siglas de “Computer Aided Engineering”).

La fusión se puede realizar teniendo como objetos de entrada:

- “Solid” + “Solid”: el objeto de salida es un sólido que ocupa todo el volumen cubierto por los objetos de entrada.
- “Shell” + “Shell”, “Shell” + “Face”, “Face” + “Face”: el resultado es un objeto “Shell”; donde las caras se cortan, se dividen, y estos objetos pueden ser no múltiples.
- “Wire” + “Wire”, “Edge” + “Wire”, “Edge” + “Edge”: el resultado es un objeto “Wire”. Las aristas son divididas donde se intersectan.

También, suele implementar primitivas tanto en uno, dos como en tres dimensiones para facilitar algunas acciones del modelado; las cuales se encuentran presentes en varios sistemas propietarios y también libres (3), algunos de ellas son:

- **Caja:** Dibuja una caja especificando sus dimensiones.
- **Cono:** Dibuja un cono especificando sus dimensiones.
- **Cilindro:** Dibuja un cilindro especificando sus dimensiones.

- **Esfera:** Dibuja una esfera especificando sus dimensiones.
- **Toro:** Dibuja un toro especificando sus dimensiones.

Algunas de las operaciones matemáticas que se pueden aplicar sobre las mencionadas primitivas son:

- **Operaciones booleanas:** Realiza operaciones booleanas sobre los objetos (Unión, Intersección, Diferencia, Sección).
- **Redondear:** Redondea las aristas de un objeto.
- **Revolución:** Crea un objeto haciendo que gire alrededor de un eje.
- **Sección:** Crea una sección por la intersección de un objeto con un plano de sección.
- **Chaflán:** Crea un chaflán en las aristas de un objeto.
- **Reflejar característica:** Crea una simetría de los objetos seleccionados alrededor de un plano de simetría dado.

“FreeCAD”, al igual que muchas aplicaciones de diseño moderno como “Revit” o “CATIA”, se basa en el concepto de Banco de Trabajo. Un Banco de Trabajo puede ser considerado como un conjunto de herramientas especialmente agrupadas para una determinada tarea. En un taller de muebles tradicional, se tiene una mesa de trabajo para la persona que trabaja con madera, otra para el que trabaja con piezas de metal y quizás una tercera para la persona que monta todas las piezas juntas (3).

En “FreeCAD”, se aplica el mismo concepto. Las herramientas se agrupan en Bancos de Trabajo de acuerdo con las tareas a las que están relacionados. Cuando se cambia de un Banco de Trabajo a otro, cambian las herramientas disponibles en la interfaz. Barras de herramientas, barras de comandos y posiblemente otras partes de la interfaz cambian al nuevo banco de trabajo, pero el contenido de su escena no cambia. Para diseñar una pieza se puede empezar a dibujar formas 2D en el Banco de Trabajo del módulo “Draft”, y a continuación, trabajar con ellos en el Banco de Trabajo del módulo Part (3).

1.2.2 Herramientas Comerciales.

Con el fin de obtener conclusiones que ayuden en el desarrollo de la investigación, se analizarán varios sistemas CAD con licencia privada para determinar las principales funcionalidades con que cuenta un módulo de piezas genérico. Los sistemas a analizar son los siguientes:

- **“Catia” (“computer-aided three dimensional interactive application”).**

  “Catia” es un programa informático de diseño, fabricación e ingeniería asistida por computadora comercial realizado por “Dassault Systèmes”. El programa está desarrollado para proporcionar apoyo desde la concepción del diseño hasta la producción y el análisis de productos. Está disponible para “Microsoft Windows”, “Solaris”, “IRIX” y “HP-UX”. Provee una arquitectura abierta para el desarrollo de aplicaciones o para personalizar el programa; las interfaces de programación de aplicaciones, se pueden programar en “Visual Basic” y C++. Fue inicialmente desarrollado para servir en la industria aeronáutica, se ha hecho un gran hincapié en el manejo de superficies complejas (9).

  Para modelar piezas genéricas se utiliza la aplicación “Catia Version 5 Part Design” (10). Esta posibilita diseñar piezas mecánicas en tres dimensiones con una interfaz de usuario flexible e intuitiva. Además, utiliza metodologías de diseño como post-diseño y parametrización local 3D. Para modelar la pieza cuenta con un “Sketcher” paramétrico el cual posee un conjunto de restricciones. Una vez creado el perfil de la pieza, se activa un banco de trabajo el cual posee varias funcionalidades agrupadas como redondeado, reflejar característica, vaciado, extrusión, crear “Sketch” en la cara de un sólido, entre otras.

- **“Autodesk Inventor”.**

  “Autodesk Inventor” es un paquete que se basa en técnicas de modelado paramétrico de sólidos en tres dimensiones producido por la empresa de software “Autodesk”. Compite con otros programas de Diseño Asistido por Computadoras como “SolidWorks”, “Catia” y “Solid Edge”. Entró en el mercado en 1999, muchos años después que los antes mencionados y se agregó a las Series de Diseño Mecánico de “Autodesk” como una respuesta de la empresa a la creciente migración de su base de clientes de diseño mecánico en dos dimensiones hacia la competencia, permitiendo que las computadoras personales ordinarias puedan construir y probar montajes de modelos extensos y complejos (11).

  Los usuarios comienzan diseñando piezas que se pueden combinar en ensamblajes, y corrigiendo las mismas, pueden obtenerse diversas variantes. Los bloques de construcción cruciales de Inventor son las piezas, se crean definiendo las características, que a su vez se basan en bocetos (dibujos en dos dimensiones). Este sistema de modelado es mucho más intuitivo que en ambientes antiguos de

modelado, en los que para cambiar dimensiones básicas era necesario generalmente suprimir el archivo entero y comenzar de cero.

Como parte final del proceso, las partes se conectan para hacer ensamblajes, los ensamblajes pueden consistir en piezas u otros ensamblajes. Las piezas son ensambladas agregando restricciones entre las superficies, bordes, planos, puntos y ejes (11).

- **“SolidWorks”.**

  “SolidWorks” es un software CAD para modelado mecánico en tres dimensiones, desarrollado en la actualidad por “SolidWorks Corp.”, una filial de “Dassault Systèmes, S.A.” (Suresnes, Francia), para el sistema operativo “Microsoft Windows”. Su primera versión fue lanzada al mercado en 1995 con el propósito de hacer la tecnología CAD más accesible (12).

  El programa permite modelar piezas y conjuntos y extraer de ellos tanto planos técnicos como otro tipo de información necesaria para la producción. Es un programa que funciona con base en las nuevas técnicas de modelado con sistemas CAD. El proceso consiste en trasvasar la idea mental del diseñador al sistema CAD, "construyendo virtualmente" la pieza o conjunto. Posteriormente todas las extracciones (planos y ficheros de intercambio) se realizan de manera bastante automatizada (13).

1.2.3 Aplicaciones en código abierto.

En el proceso de investigación que se lleva a cabo a nivel del grupo de investigación, se logró identificar como una línea estratégica importante durante el desarrollo, la reutilización del código fuente disponible, con funcionalidades probadas en otros sistemas de código abierto, con lo que será posible reducir los tiempos de desarrollo para la obtención de las metas planteadas. Por ejemplo, se puede reutilizar código disponible en varios softwares como “FreeCAD”, “LibreCAD” y “Salome Meca” el cual es un sistema para la Ingeniería Asistida por Computadora (“Computer Aided Engineering” o CAE). Los sistemas a analizar son los siguientes:

- **“FreeCAD”.**

  “FreeCAD” es una aplicación libre de Diseño Asistido por Computadoras en tres dimensiones e Ingeniería Asistida por Computadoras, para la asistencia en

ingeniería mecánica y el diseño de elementos mecánicos. Está basado en “Open CASCADE” y programado en los lenguajes C++ y “Python” (13).

A diferencia de los CAD analíticos tradicionales, como pueden ser “AutoCAD” o “Microstation”, “FreeCAD” es un CAD paramétrico que utiliza parámetros para definir sus límites o acciones. En el diseño paramétrico cada elemento del dibujo (muros, puertas, ventanas, etc.) es tratado como un objeto, el cual no es definido únicamente por sus coordenadas espaciales (x, y, z), sino también por sus parámetros, ya sean estos gráficos o funcionales.

“FreeCAD” posee varias funcionalidades para el diseño de piezas genéricas dispersas en tres módulos, “Part”, “Part Design” y “Draft”.

Para el diseño del perfil de la pieza “FreeCAD” cuenta con un “Sketcher” paramétrico el cual permite crear un conjunto de geometrías (3), algunas de ellas se listan a continuación:

- **Punto:** dibuja un punto.
- **Línea dados dos puntos:** Dibuja un segmento de línea entre los dos puntos.
- **Arco:** Dibuja un segmento de arco dado un centro, un radio, un ángulo de inicio y un ángulo de fin.
- **Círculo:** Dibuja un círculo dado un centro y un radio.
- **Polilínea:** Dibuja una línea formada por múltiples segmentos de línea.
- **Rectángulo:** Dibuja un rectángulo dados dos puntos opuestos.
- **Redondeado:** Realiza un redondeado entre dos líneas unidas en un punto.

El “Sketcher” cuenta también con un conjunto de restricciones matemáticas para lograr una mayor precisión en el modelado del perfil de la pieza. Estas restricciones se dividen en dos tipos: no asociadas a valores numéricos y asociadas a valores numéricos (3).

Entre las no asociadas a valores numéricos se encuentran las siguientes:

- **Vertical:** Restringe las líneas seleccionadas a una verdadera orientación vertical. Se puede seleccionar más de un objeto antes de aplicar esta restricción.
- **Horizontal:** Restringe las líneas seleccionadas a una verdadera orientación horizontal. Se puede seleccionar más de un objeto antes de aplicar esta restricción.

- **Paralela:** Restringe dos o más líneas a que sean paralelas con respecto a las demás.
- **Perpendicular:** Restringe dos líneas a que sean perpendiculares una a la otra.
- **Igual longitud:** Restringe dos entidades seleccionadas a que tengan la misma longitud. Si esta restricción es usada en círculos o arcos de círculos sus radios serán iguales.

Para las restricciones asociadas a valores numéricos se pueden usar expresiones matemáticas. Algunas de estas restricciones se describen a continuación:

- **Distancia horizontal:** Fija la distancia horizontal entre dos puntos o dos puntos finales de una línea.
- **Distancia vertical:** Fija la distancia vertical entre dos puntos, o dos puntos finales de una línea.
- **Longitud:** Define la distancia de una línea seleccionada restringiendo su longitud, o define la distancia entre dos puntos restringiendo la distancia entre ellos.
- **Radio:** Define el radio de un arco o un círculo restringiéndolo a un valor.
- **Ángulo interno:** Define el ángulo interno entre dos líneas seleccionadas.

El “Sketcher” es incluido por el módulo “Part Design”, para ejecutar sus funcionalidades, como por ejemplo el “sketch support”. Un “sketch” puede ser creado en un plano estándar, en un plano paralelo a uno estándar o en la cara plana de un sólido; en este último caso, el sólido existente se convierte en el soporte del “sketch”. Una de las herramientas del módulo “Part Design” que funciona con un sketch con soporte, es el “Pocket”; definido como una extrusión en el cual se le retira material a un sólido.

- **“SALOME-MECA”:**

  “SALOME-MECA” es un software de código abierto que proporciona una plataforma genérica para pre y post-procesamiento para la simulación numérica. Se basa en una arquitectura abierta y flexible hecha de componentes reutilizables. “SALOME-MECA” es una solución multiplataforma. Se distribuye como código abierto bajo los términos de la licencia LGPL. Se puede utilizar como aplicación independiente para la generación de modelo CAD, en la preparación para cálculos numéricos y post-procesamiento de los resultados de cálculo. También puede utilizarse como una

plataforma para la integración de los códigos numéricos de terceros externos para producir una nueva aplicación para la gestión completa del ciclo de vida de los modelos CAD (14).

La aplicación “SALOME-MECA” posee varias funcionalidades que permiten calcular información topológica y dimensiones de una figura como son el volumen de un sólido, el área de una superficie, la longitud de una entidad, el centro de gravedad de un sólido, entre otros. Además, permite realizar operaciones de transformación sobre objetos, entre las que se encuentran la traslación, rotación, modificación de la ubicación de una entidad, reflexión de característica y escalado. Al igual que otras herramientas CAD, posee funcionalidades básicas para el modelado en tres dimensiones como son la extrusión y revolución. También posibilita la creación de figuras geométricas básicas y primitivas 3D.

1.2.4 Funcionalidades para el modelado de piezas genéricas.

Los softwares de modelado de piezas crean una representación virtual en tres dimensiones de componentes para el análisis y diseño de piezas genéricas. Un modelo de piezas genéricas consiste en un grupo de figuras, adicionadas una a la vez, hasta que el modelo esté completo. Una de las principales funcionalidades para el modelado de piezas son los sketchers. Estos son herramientas para generar objetos 2D en el diseño de piezas.

Otro tipo de funcionalidad para el modelado es el “surfacing” (modelado de superficie de forma libre). Aquí las superficies se definen, se recortan, se fusionan y se llenan para hacer sólido. Las superficies suelen definirse con curvas de referencia en el espacio y una variedad de comandos complejos. El modelado “surfacing” es más difícil, pero mejor aplicable a algunas técnicas de fabricación, como el moldeo por inyección. Los modelos sólidos para piezas moldeadas por inyección usualmente tienen características de superficie y dibujo (15).

El modelado paramétrico es otra herramienta que facilita la creación de componentes y piezas. Este usa parámetros para definir un modelo. Ejemplo de parámetros son las dimensiones usadas para crear una figura geométrica, la densidad del material, las fórmulas para describir características de barridas, datos importados, entre otros. Los parámetros pueden ser modificados posteriormente a la creación de la figura, y el modelo se actualizará para reflejar dichas modificaciones.

Algunos modelos paramétricos permiten el uso de restricciones. Estas se usan para construir relaciones entre los parámetros; cuando un elemento cambia debido a la regeneración de una pieza, lo hace manteniendo las restricciones asociadas al modelo.

1.2.5 Herramientas de ayuda al modelado.

En el diseño de piezas genéricas se hace necesario disponer de una forma de hacer el proceso de modelado más simple, y los objetos geométricos como las líneas, puntos y planos brindan un conjunto de características y propiedades para realizarlo.

Las líneas de construcción (también conocidas como líneas x), son entidades temporales que son usadas como referencias a la hora de crear y posicionar objetos en un plano. Las líneas de construcción pueden ser círculos o líneas rectas que se extiendan hasta el infinito en ambas direcciones (11).

Las líneas de construcción poseen ciertas propiedades como son:
Perpendicularidad: Una línea es perpendicular a otra cuando la corta originando un ángulo de 90 grados. Las líneas perpendiculares dividen al plano en cuatro zonas o ángulos rectos.

Paralelismo: Dos líneas se consideran paralelas si siguen la misma dirección, y, por tanto, aunque se prolonguen no se cortan.

Tangente: es una recta que toca un punto o una línea o superficie curva. El punto de contacto se denomina punto de tangencia o de contacto.

Otras herramientas de ayuda al modelado son los patrones de repetición de figuras, entre los cuales se encuentran el patrón polar y el patrón lineal. El patrón polar toma una figura y crea un conjunto de copias de la misma figura, repetidas circularmente alrededor de un eje definido, mientras que el patrón lineal selecciona una o más figuras y crea una repetición de estas trasladadas en una dirección específica. Este conjunto de funcionalidades, aunque importante para asistir la construcción de los modelos, no forman parte del presente trabajo, pero se recomienda realizar su desarrollo para completar los requerimientos de los módulos para el modelado de piezas.

1.2.6 Asignación y determinación de características geométricas y físicas a una pieza genérica.

Existen formas para definir una pieza y sus características, algunas de ellas son el punto, la línea, el plano, el círculo, el cilindro, el cono y la esfera. También existen ciertas características geométricas y físicas que determinan la condición de piezas como son:

Volumen: es una magnitud métrica de tipo escalar definida como la extensión en tres dimensiones de una región del espacio (16). Es una magnitud derivada de la longitud, ya que se halla multiplicando la longitud, el ancho y la altura.

Masa: es una magnitud que expresa la cantidad de materia de un cuerpo, medida por la inercia de este, que determina la aceleración producida por una fuerza que actúa sobre él (16).

Peso: es una medida de la fuerza gravitatoria que actúa sobre un objeto. El peso equivale a la fuerza que ejerce un cuerpo sobre un punto de apoyo, originada por la acción del campo gravitatorio local sobre la masa del cuerpo (16).

Rectitud: Una condición en la que todos los puntos forman una línea recta (17).

Plano: Todos los puntos de una superficie están en un plano.

Redondez: Todos los puntos de una superficie forman un círculo.

1.3 Fundamentos matemáticos y algoritmos para el modelado de piezas tipo genéricas.

Para el diseño de piezas genéricas, hay que tener en cuenta el significado de Modelado Sólido. Este es una rama relativamente reciente del Modelado Geométrico, que hace hincapié en crear solamente modelos “completos” de los sólidos, es decir, modelos que son adecuados para responder algorítmicamente (sin la ayuda externa del usuario) a cualquier pregunta geométrica que se formule.

Los principales esquemas de Modelado Sólido desarrollados son el de Representación de Fronteras (“Boundary Representation” o B-Rep) y el de la Geometría Constructiva de Sólidos (“Constructive solid Geometry” o CSG), aunque existen muchos otros, como el modelado de barrido translacional y rotacional, o los esquemas de modelado híbridos (18).

Los modelos B-Rep están compuestos por dos partes: topología y geometría. Los principales elementos topológicos son caras (porción limitada de una superficie), aristas (pieza limitada de una curva) y vértices (definido por un punto), mientras que entre los geométricos se encuentran superficies, curvas y puntos (19).

Por otra parte, CSG es una técnica que permite crear objetos sólidos mediante el uso de operaciones booleanas para combinar primitivas geométricas como el cono y el cilindro, por citar algunas. A estas primitivas se le aplican operaciones de traslación y rotación, para después realizar una operación de corte, unión o diferencia.

Para el modelado de piezas hay que tener en cuenta dos conceptos importantes, que son la **interpolación** y la **extrusión**. La **interpolación** se denomina a la obtención de nuevos puntos partiendo del conocimiento de un conjunto discreto de puntos. Uno de los métodos

más simples es la interpolación lineal, la cual es rápida y sencilla, pero en ciertos casos no muy precisa, también existe la interpolación polinómica y la interpolación mediante splines. Un spline es una curva diferenciable definida en porciones mediante polinomios. En los problemas de interpolación, se utiliza a menudo la interpolación mediante splines porque da lugar a resultados similares requiriendo solamente el uso de polinomios de bajo grado, evitando así las oscilaciones, indeseables en la mayoría de las aplicaciones, encontradas al interpolar mediante polinomios de grado elevado (20). En el modelado de sólidos, la **extrusión** es la operación de convertir cualquier superficie plana en una superficie tridimensional, estirándola sobre un eje definido (3)**.**

1.4 Acerca de las metodologías para el desarrollo.

Con el incremento de la complejidad de los sistemas informáticos, se hizo necesario elaborar metodologías para organizar los procesos de desarrollo, en el presente apartado, se revisan algunos aspectos sobre las mismas.

Se entiende por metodología una colección de documentación formal referente a los procesos, las políticas y los procedimientos que intervienen en el desarrollo del software y, a su vez, los estándares de desarrollo por una estructura aplicada al desarrollo de un producto de software que contempla varios modelos a seguir para el establecimiento de un proceso para el desarrollo de software, cada uno de los cuales describe un enfoque diferente para diferentes actividades que tienen lugar durante el proceso. Algunos autores consideran un modelo de ciclo de vida, un término más general que un determinado proceso para el desarrollo de software.

1.4.1 Metodología de desarrollo AUP:

El Proceso Unificado Ágil de Scott Ambler (AUP por las siglas de “Agil Unified Process”) es una versión simplificada del Proceso Racional Unificado (RUP por las siglas de “Rational Unified Process”). Este describe de una manera simple y fácil de entender la forma de desarrollar aplicaciones de software de negocio mediante técnicas ágiles y conceptos que aún se mantienen válidos en RUP. El AUP aplica técnicas ágiles que incluyen Desarrollo Dirigido por Pruebas, Modelado Ágil, Gestión de Cambios Ágil, y Refactorización de Base de Datos para mejorar la productividad (21).

El proceso unificado (UP por las siglas de “Unified Process”) es un marco de desarrollo de software iterativo e incremental. A menudo es considerado como un proceso altamente ceremonioso porque especifica muchas actividades y artefactos involucrados en el

desarrollo de un proyecto software. Dado que es un marco de procesos, puede ser adaptado y la más conocida es RUP de IBM.

AUP se preocupa especialmente de la gestión de riesgos. Propone que aquellos elementos con alto riesgo obtengan prioridad en el proceso de desarrollo y sean abordados en etapas tempranas del mismo. Para ello, se crean y mantienen listas identificando los riesgos desde etapas iníciales del proyecto. Especialmente relevante en este sentido es el desarrollo de prototipos ejecutables durante la base de elaboración del producto, donde se demuestre la validez de la arquitectura para los requisitos clave del producto y que determinan los riesgos técnicos (21).

El proceso AUP establece un Modelo más simple que el que aparece en RUP por lo que reúne en una única disciplina las disciplinas de Modelado de Negocio, Requisitos y Análisis y Diseño. El resto de disciplinas (Implementación, Pruebas, Despliegue, Gestión de Configuración, Gestión y Entorno) coinciden con las restantes de RUP (21).

Al igual que en RUP, en AUP se establecen cuatro fases que transcurren de manera consecutiva y que acaban con hitos claros alcanzados:

Inicio: El objetivo de esta fase es obtener una comprensión común cliente-equipo de desarrollo del alcance del nuevo sistema y definir una o varias arquitecturas candidatas para el mismo.

Elaboración: El objetivo es que el equipo de desarrollo profundice en la comprensión de los requisitos del sistema y en validar la arquitectura.
Construcción: Durante la fase de construcción el sistema es desarrollado y probado al completo en el ambiente de desarrollo.
Transición: El sistema se lleva a los entornos de preproducción donde se somete a pruebas de validación y aceptación y finalmente se despliega en los sistemas de producción.

Las disciplinas se llevan a cabo de manera sistemática, a la definición de las actividades que realizan los miembros del equipo de desarrollo a fin de desarrollar, validar, y entregar el software de trabajo que responda a las necesidades de sus interlocutores. Las disciplinas son:
Modelo: El objetivo de esta disciplina es entender el negocio de la organización, el problema de dominio que se abordan en el proyecto, y determinar una solución viable para resolver el problema de dominio.

Aplicación: El objetivo de esta disciplina es transformar su modelo (s) en código ejecutable y realizar un nivel básico de las pruebas, en particular, la unidad de pruebas.

Prueba: El objetivo de esta disciplina consiste en realizar una evaluación objetiva para garantizar la calidad. Esto incluye la búsqueda de defectos, validar que el sistema funciona tal como está establecido, y verificar que se cumplan los requisitos.

Despliegue: El objetivo de esta disciplina es la prestación y ejecución del sistema y que el mismo este a disposición de los usuarios finales.

Gestión de configuración: El objetivo de esta disciplina es la gestión de acceso a herramientas de su proyecto. Esto incluye no sólo el seguimiento de las versiones con el tiempo, sino también el control y gestión del cambio para ellos.

Gestión de proyectos: El objetivo de esta disciplina es dirigir las actividades que se lleva a cabo en el proyecto. Esto incluye la gestión de riesgos, la dirección de personas (la asignación de tareas, el seguimiento de los progresos, entre otros), coordinación con el personal y los sistemas fuera del alcance del proyecto para asegurarse de que es entregado a tiempo y dentro del presupuesto.

Entorno: El objetivo de esta disciplina es apoyar el resto de los esfuerzos por garantizar que el proceso sea el adecuado, la orientación (normas y directrices), y herramientas (hardware, software, entre otros) estén disponibles para el equipo según sea necesario.

Variación de AUP para la UCI en el escenario 4.

Al no existir una metodología de software universal, ya que toda metodología debe ser adaptada a las características de cada proyecto (equipo de desarrollo, recursos, etc.) exigiéndose así que el proceso sea configurable. Se decide hacer una variación de la metodología AUP, de forma tal que se adapte al ciclo de vida definido para la actividad productiva de la UCI. Una metodología de desarrollo de software tiene entre sus objetivos aumentar la calidad del software que se produce, de ahí la importancia de aplicar buenas prácticas, para ello nos apoyaremos en el Modelo CMMI-DEV v1.3, el cual constituye una guía para aplicar las mejores prácticas en una entidad desarrolladora. Estas prácticas se centran en el desarrollo de productos y servicios de calidad (21).

De las 4 fases que propone AUP (Inicio, Elaboración, Construcción, Transición) se mantuvo la fase de Inicio, pero modificando el objetivo de la misma, se unificaron las restantes 3 fases de AUP en una sola, llamada Ejecución y se agregó la fase de Cierre.

AUP propone 7 disciplinas (Modelo, Implementación, Prueba, Despliegue, Gestión de configuración, Gestión de proyecto y Entorno), se decidió para el ciclo de vida de los

proyectos de la UCI tener 7 disciplinas también, pero a un nivel más atómico que el definido en AUP. Los flujos de trabajos: Modelado de negocio, Requisitos y Análisis y diseño en AUP están unidos en la disciplina Modelo, en la variación para la UCI se consideran a cada uno de ellos disciplinas. Se mantiene la disciplina Implementación, en el caso de Prueba se desagrega en 3 disciplinas: Pruebas Internas, de Liberación y Aceptación. Las restantes 3 disciplinas de AUP asociadas a la parte de gestión para la variación UCI se cubren con las áreas de procesos que define CMMI-DEV v1.3 para el nivel 2, serían CM (Gestión de la configuración), PP (Planeación de proyecto) y PMC (Monitoreo y control de proyecto) (21).

AUP propone 9 roles (Administrador de proyecto, Ingeniero de procesos, Desarrollador, Administrador de Bases de Datos, Modelador ágil, Administrador de la configuración, Stakeholder, Administrador de pruebas, Probador), se decide para el ciclo de vida de los proyectos de la UCI tener 11 roles, manteniendo algunos de los propuestos por AUP y unificando o agregando otros. Los roles son los siguientes (21):

- Jefe de proyecto.
- Planificador.
- Analista.
- Arquitecto de información.
- Desarrollador.
- Administrador de la configuración.
- “Stakeholder” (Cliente/Proveedor de requisitos).
- Administrador de calidad.
- Probador.
- Arquitecto de software.
- Administrador de Base de Datos.

De los cuatro escenarios con que cuenta la metodología AUP-UCI se seleccionó el escenario 4, ya que posee un negocio muy bien definido, además el cliente estará siempre acompañando al equipo de desarrollo para convenir los detalles de los requisitos y se cuenta con poco tiempo para su realización.

1.4.2 Lenguaje de modelado unificado

Lenguaje de Modelado Unificado (UML por las siglas de “Unified Modeling Language”) es un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software (22).

El UML proporciona a los desarrolladores un vocabulario que incluye tres categorías: elementos, relaciones y diagramas (22).

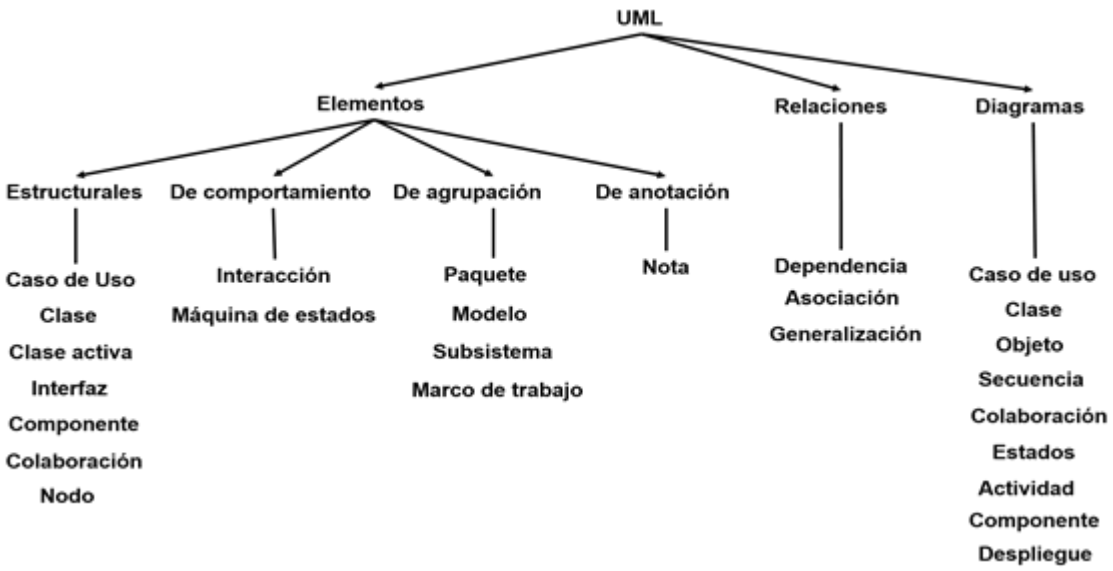


Figura 4: Vocabulario de UML (14).

El UML está pensado para ser empleado en herramientas interactivas de modelado visual que tengan generadores de código y/o generadores de informes. La especificación de UML no define un proceso estándar, pero está ideado para ser útil en un proceso de desarrollo iterativo. Pretende dar apoyo a la mayoría de los procesos de desarrollo orientados a objetos (22).

La capacidad de UML para modelar cualquier sistema de software y el hecho de que haya sido adoptado por el Grupo de Gestión de Objetos como un estándar desde noviembre de 1997, permitieron determinarlo como el lenguaje de modelado para desarrollar la propuesta de solución (22).

Se escogió como lenguaje de modelado UML porque es posible establecer una serie de requerimientos y estructuras necesarias para modelar un sistema de software previo al proceso intensivo de escribir código. UML es un lenguaje que posee más características visuales que programáticas, las que facilitan a integrantes de un equipo participar e intercomunicarse fácilmente, siendo estos integrantes los analistas, diseñadores, especialistas de área y desde luego los programadores. Una de las características más importantes que hacen que se escoja UML como lenguaje de modelado es que está

diseñado para uso con software orientado a objetos, y tiene un uso limitado en otro tipo de cuestiones de programación.

1.4.3 Herramienta para el modelado. “Visual Paradigm” v8.0

Las herramientas para el modelado son un conjunto de programas y ayudas que brindan asistencia a los analistas, ingenieros de software y desarrolladores, durante todos los pasos del ciclo de vida de desarrollo de un software (Investigación preliminar, Análisis, Diseño, Implementación e Instalación).

Existen varias herramientas para el modelado entre las que se encuentran “Rational Rose”, “ArgoUML” y “Visual Paradigm”. En el desarrollo de esta investigación será utilizada “Visual Paradigm” para crear los diagramas que serán elaborados en cada flujo de trabajo propuesto por la metodología y los prototipos de interfaces que se utilizarán.

“Visual Paradigm” ha sido concebida para soportar el ciclo de vida completo del proceso de desarrollo del software a través de la representación de todo tipo de diagramas. Puede ser utilizado en diferentes plataformas lo que es una ventaja ya que el componente será desarrollado en Ubuntu. Permite realizar ingeniería directa e inversa, y de una manera muy simple, se pueden crear todo tipo de diagramas UML.

1.5 Tecnologías para el desarrollo, frameworks y lenguajes.

Las tecnologías están presentes en todos los procesos de la sociedad y de la economía del ser humano. Su principal propósito es transformar el entorno, para adaptarlo mejor a las necesidades y deseos del hombre. Las TIC son aquellas herramientas computacionales e informáticas que procesan, almacenan, sintetizan, recuperan y presentan información en la más variada forma; el desarrollo alcanzado por estas tecnologías, trae consigo un continuo perfeccionamiento de las herramientas informáticas, por tanto, al implementar cualquier tipo de software, se hace necesario realizar un estudio detallado en relación a cualquier recurso a utilizar. Seguidamente se describen las tecnologías empleadas en la solución y se realiza un análisis de las principales características, ventajas y desventajas de cada una de ellas, con el fin de realizar una mejor selección que pueda sostener el desarrollo de la solución propuesta.

1.5.1 Lenguaje C++

C++ es un lenguaje de programación, diseñado a mediados de los años 1980, por Bjarne Stroustrup, como extensión del lenguaje de programación C, abarca tres paradigmas de la

programación: la programación estructurada, la programación genérica y la programación orientada a objetos por lo que se considera un lenguaje híbrido multiparadigma (Un Lenguaje de programación multiparadigma es el cual soporta más de un paradigma de programación. Permiten crear programas usando más de un estilo de programación.). C++ es un lenguaje de programación maduro y de gran velocidad de compilación y presenta las siguientes características (23):

- Lenguaje versátil, potente y general.
- Lenguaje rico en operadores y expresiones.
- Flexible, conciso y eficiente.
- Presenta programación modular.
- Introduce nuevas palabras claves y operadores para manejo de clases.

C++ está considerado por muchos como el lenguaje más potente, debido a que permite trabajar tanto a alto como a bajo nivel. Además, se trata de un lenguaje de programación estandarizado (ISO/IEC 14882:1998), ampliamente difundido, y con una biblioteca estándar C++ que lo ha convertido en un lenguaje universal, de propósito general, y ampliamente utilizado tanto en el ámbito profesional como en el educativo. Posee una serie de propiedades difíciles de encontrar en otros lenguajes de alto nivel (23):

- Posibilidad de redefinir los operadores (sobrecarga de operadores).
- Identificación de tipos en tiempo de ejecución (RTTI).

C++ fue el lenguaje seleccionado porque además de que lo utiliza el framework seleccionado (QT), posee características que lo realzan de los demás lenguajes, y las funcionalidades de “Open CASCADE” están desarrolladas en dicho lenguaje. También se seleccionó por las ventajas que brinda la programación orientada a objetos y por el conocimiento y la experiencia adquiridos con este lenguaje.

1.5.2 Contador de líneas de código CCCC

En el marco de la aplicación se pretende seguir con especial atención la cantidad de líneas de código procesadas, por lo que se hace necesaria una aplicación capaz de automatizar este trabajo. C and C++ Code Counter (CCCC por sus siglas en inglés) fue desarrollado como una base de prueba para varias ideas sobre métricas, siguiendo la metodología de cantidad de líneas de código escritas, la cual es similar a la que se pretende utilizar en el proyecto para llevar un seguimiento del trabajo realizado por los desarrolladores.

1.5.3 Framework Qt

Qt es un framework de desarrollo para aplicaciones multiplataforma que simplifica mucho el desarrollo de aplicaciones en C++ de forma nativa, también puede ser utilizado en otros lenguajes, funciona en las principales plataformas y tiene un amplio apoyo. Es una biblioteca para desarrollar interfaces gráficas de usuario y también para el desarrollo de programas sin interfaz gráfica como herramientas de consola y servidores.

Este *framework* está compuesto por bibliotecas Qt (clases en C++), además se pueden crear formularios visualmente con "*Qt Designer"* y presenta un acceso rápido a la documentación a través de "*Qt Assistant"*. Permite una traducción rápida con su "*Qt Linguist"* y simplifica el proceso de construcción de proyectos en las diferentes plataformas soportadas. La API de la biblioteca cuenta con métodos para acceder a bases de datos mediante SQL, así como uso de XML; cuenta con una API multiplataforma unificada para la manipulación de archivos y otras para el manejo de ficheros, además de estructuras de datos tradicionales.

Otras Características:

- QT constituye además una biblioteca para la creación de interfaces gráficas. Se distribuye bajo una licencia libre GPL (o QPL), además de la LGPL, que permite incorporar QT en una aplicación open-source.
- Se encuentra disponible para un gran número de plataformas: Linux, MacOs X, Solaris, HP-UX, UNIX con X11, Windows.
- Es orientado a objetos y el lenguaje para el que se encuentra disponible es C++.
- Es una biblioteca que se basa en los conceptos de widgets (objetos), Señales-Slots y Eventos (Ej.: clic del ratón).
- Las señales y los slots es el mecanismo para que unos widgets se comuniquen con otros.
- Los widgets pueden contener cualquier número de hijos. El "widget" superior (también conocido como "top-level"), puede ser cualquiera, sea ventana, botón, etc.
- Compatibilidad multiplataforma con un sólo código fuente.
- Fácil de internacionalizar.
- Arquitectura lista para "plugins".

Con todo lo anterior expuesto se puede decir que Qt dispone de tres grandes ventajas ante otras librerías rivales (24):

- Qt es completamente gratuito para aplicaciones de código abierto, y una de las licencias mencionadas anteriormente, la LGPL, permite que se utilice gratuitamente con fines comerciales.
- Las herramientas, librerías y clases están disponibles para casi todas las plataformas Unix y sus derivados (MacOs X y Solaris) como también para la familia Windows, por lo que una aplicación puede ser compilada y utilizada en cualquier plataforma sin necesidad de cambiar el código y la aplicación se verá y actuará mejor que una aplicación nativa.
- Qt tiene una extensa librería con clases y herramientas para la creación de aplicaciones con interfaz de usuario enriquecidas. Estas librerías y clases están bien documentadas, son muy fáciles de usar y tienen una gran herencia de programación orientada a objetos lo cual hace de la programación de interfaces gráficas un trabajo más ameno.

A continuación, otras de las características más importantes de este framework que ayudan a la selección del mismo (24):

- Compatibilidad multiplataforma con un sólo código fuente.
- Rendimiento en C++.
- Disponibilidad del código fuente.
- Excelente documentación.

Con este framework se pueden crear de forma rápida y sencilla interfaces gráficas de usuario; tiene disponible como lenguaje C++, siendo este el lenguaje seleccionado para la implementación del componente. También el uso de esta biblioteca ahorra una enorme cantidad de tiempo y esfuerzo, pues de lo contrario se tendrían que desarrollar algunos componentes desde cero. Estas son las principales características que debe tener la aplicación que se desarrolla, lo cual dio paso a la selección del mismo para la realización de la aplicación de este trabajo.

1.5.4 Open CASCADE Community Edition

Dentro de las variantes que han surgido para la implementación y trabajo con las tecnologías CAD/CAE/CAM(“Computer Aided Manufacturing”) en código libre, está la tecnología “Open Cascade” (1).

“Open CASCADE” (abreviatura de las siglas en inglés “Computer Aided Software for Computer Aided Design and Engineering”), desarrollada inicialmente a principios de los años 90 por Matra Datavision, es una aplicación orientada a usuarios que se desenvuelven dentro del área del diseño, como ingenieros, arquitectos y diseñadores. Gracias a sus bondades es posible diseñar y modificar cualquier modelo bidimensional o tridimensional deseado (1).

Cuenta con una licencia de funcionamiento GPL, lo que permite utilizarla de forma ilimitada en cualquier ordenador. Se comporta como un entorno de desarrollo integrado, orientado a la edición y el diseño de superficies y objetos bidimensionales y tridimensionales, y utiliza para su diseño una gran cantidad de opciones de personalización que facilitan enormemente el modelado de estas figuras u objetos (1).

Cuenta con la capacidad de realizar análisis gráficos de cualquier superficie, lo que se traduce en estimar y simular los datos generados por las curvas de los gráficos de forma numérica y rápida, para de esta manera facilitar los estudios de ingeniería. Por si fuera poco, ofrece la posibilidad de importar y exportar figuras. Además, viene con un paquete completo de efectos especiales y texturas que se pueden añadir a los diseños de modo que se puedan personalizar por completo.

La manera más sencilla de trabajar con “Open CASCADE” es a través de su framework OCAF (“Open CASCADE Application Framework”), por sus siglas en inglés. OCAF es una plataforma fácil de usar y que permite el desarrollo rápido de aplicaciones sofisticadas de diseño. Una aplicación típica desarrollada con OCAF puede tener un modelador geométrico en dos o tres dimensiones, herramientas de fabricación o de análisis, y aplicaciones de simulación o herramientas de ilustración. Cuenta con un sinnúmero de características que la convierten en una opción atractiva a la hora de hacer una selección de tecnología para el trabajo con CAD/CAE/CAM, y que justifican su selección en este caso, como son (1):

- Facilidad de diseño de la arquitectura de la aplicación.
- Definición de los componentes y la forma en que cooperan.
- Definición del modelo de datos capaz de soportar las funcionalidades requeridas.
- Estructuración del software en relación a:
    - Sincronizar la visualización con los datos. Los comandos de visualización de objetos tienen que actualizar la vista una vez que estos son llamados.

- Soportar comandos para las operaciones “deshacer” y “rehacer” (Ctrl+Z). Esta función debe ser tomada en cuenta muy tempranamente en el proceso de diseño, y hecho así, funciona muy eficazmente.
- Permite la implementación de las funciones para salvar los datos. Si la aplicación tiene un ciclo de vida muy largo, la compatibilidad de los datos entre las versiones tiene que ser tratada.
- Facilidad para crear una interfaz de usuario.

- Gracias a tener implementadas todas estas funciones, permite desarrollar aplicaciones en buen tiempo, para que así, a la hora de desarrollar una aplicación, solo sea necesario concentrarse en las funcionalidades específicas, y no en la esencia del código que se maneja, factor decisivo para su selección final.

1.5.5 Git

Software de control de versiones diseñado por Linus Torvalds, pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando éstas tienen un gran número de archivos de código fuente. Al principio, Git se pensó como un motor de bajo nivel sobre el cual otros pudieran escribir la interfaz de usuario o front end como Cogito o StGIT. Sin embargo, Git se ha convertido desde entonces en un sistema de control de versiones con funcionalidad plena. Hay algunos proyectos de mucha relevancia que ya usan Git, en particular, el grupo de programación del núcleo Linux.

Cada desarrollador o equipo de desarrollo puede hacer uso de Git de la forma que le parezca conveniente. Sin embargo, una buena práctica es la siguiente:
Se deben utilizar 4 tipos de ramas: “Master”, “Development”, “Features”, y “Hotfix”.

“Master”:

Es la rama principal. Contiene el repositorio que se encuentra publicado en producción, por lo que debe estar siempre estable.

“Development”:

Es una rama sacada de master. Es la rama de integración, todas las nuevas funcionalidades se deben integrar en esta rama. Luego que se realice la integración y se corrijan los errores (en caso de haber alguno), es decir que la rama se encuentre estable, se puede hacer un merge de development sobre la rama master.

“Features”:

Cada nueva funcionalidad se debe realizar en una rama nueva, específica para esa funcionalidad. Estas se deben sacar de development. Una vez que la funcionalidad esté

desarrollada, se hace un merge de la rama sobre development, donde se integrará con las demás funcionalidades.

“Hotfix”:

Son “bugs” que surgen en producción, por lo que se deben arreglar y publicar de forma urgente. Es por ello, que son ramas sacadas de master. Una vez corregido el error, se debe hacer un merge de la rama sobre “master”. Al final, para que no quede desactualizada, se debe realizar el “merge” de “master” sobre “development”.

1.6 Consideraciones del capítulo.

Se presentó el diseño teórico de la investigación en cuyo marco se analizaron las dificultades que enfrenta la ingeniería nacional en relación con el uso de los sistemas propietarios para el diseño asistido por computadora. En el estudio de la arquitectura de la aplicación “FreeCAD”, se determinó como uno de los principales problemas que las funcionalidades para el modelado de piezas se encuentran dispersas en tres módulos denominados “Part”, “Part Design” y “Draft”, lo que afecta la productividad del diseñador durante el proceso de diseño.

Se abordaron temas relacionados con los módulos de piezas genéricas, se mostró la metodología de modelado de piezas genéricas y se expusieron las decisiones, que en el contexto del proyecto se tomaron, sobre las metodologías para el desarrollo de software Variación AUP para la UCI en el escenario 4, que se usarán en el desarrollo del módulo para el modelado paramétrico de piezas y componentes mecánicos en tres dimensiones. Se presentó la selección del patrón arquitectónico adecuado junto con su justificación y se analizaron las tendencias y tecnologías de desarrollo de software actuales ofreciendo características, ventajas y desventajas de las mismas.

Capítulo 2: Propuesta de solución.

A continuación, se aborda la propuesta de solución, se identifican y describen los requisitos funcionales y no funcionales, se presenta el estilo y patrón arquitectónico, los patrones de diseño empleados, el diagrama de clases y las historias de usuario.

2.1 Modelo del dominio:

Un Modelo de Dominio es un artefacto de la disciplina de análisis, construido con las reglas de UML durante la fase de concepción, en la tarea construcción del modelo de dominio, presentado como uno o más diagramas de clases y que contiene, no conceptos propios de un sistema de software sino de la propia realidad física, (25).

Los modelos de dominio pueden utilizarse para capturar y expresar el entendimiento ganado en un área bajo análisis como paso previo al diseño de un sistema, ya sea de software o de otro tipo. Similares a los mapas mentales utilizados en el aprendizaje, el modelo de dominio es utilizado por el analista como un medio para comprender el sector industrial o de negocios al cual el sistema va a servir. A continuación se presenta el modelo del dominio definido en la solución:

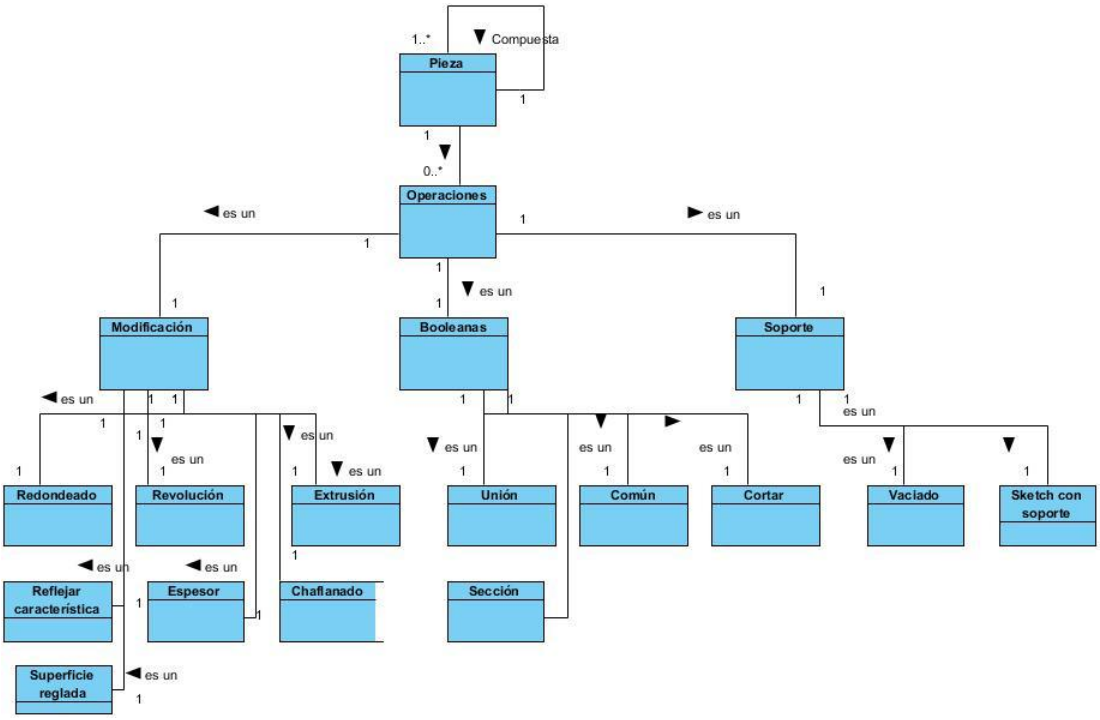


Figura 5: Modelo de dominio.

2.2 Definición de las clases del Modelo de dominio

Pieza: es un objeto sólido que tiene forma y volumen. Está fabricado con material duradero ya que forma parte de máquinas que realizan trabajos y debido a eso deber ser muy resistente (26).

Extrusión: extiende una forma a una distancia especificada, en una dirección especificada.

Revolución: Crea un objeto haciendo que gire alrededor de un eje.

Reflejar característica: Crea una simetría de los objetos seleccionados alrededor de un plano de simetría dado.

Redondeado: Redondea las aristas seleccionadas de un objeto.

Chaflanado: Crea un chaflán en las aristas seleccionadas de un objeto.

Superficie reglada: Crea una superficie reglada entre dos aristas.

Espesor: Transforma un sólido en un objeto hueco, y define un espesor a cada una de las caras del objeto.

Unión: Fusión (unión) de dos objetos.

Común: Extrae la parte común (intersección) de dos objetos.

Cortar: Corta (resta) un objeto de otro.

Sección: Extrae una sección de la intersección de dos formas seleccionadas, la segunda forma se utiliza como un plano de sección.

Sketch con soporte: Crea un sketch con soporte en la cara plana de un sólido.

Vaciado: Crea un vaciado en la cara de un sólido la cual sirve de soporte a un sketch.

2.3 Descripción del componente

Las funcionalidades desarrolladas tienen base en el código de la aplicación FreeCAD, y permiten el modelado paramétrico de piezas y componentes genéricos en tres dimensiones. La parametrización viene dada debido a que para diseñar el perfil de la pieza se parte de un sketch paramétrico utilizando el módulo Sketcher, el cual define los parámetros con que cuenta un objeto haciendo uso de la clase Property y sus derivadas del núcleo de la aplicación. Cuando uno de estos parámetros cambia en un objeto, el núcleo manda una señal indicando que una propiedad cambió, para posteriormente recalcular el objeto.

Para lograr la integración del módulo Sketcher con las funcionalidades desarrolladas está la clase Part2DObject, la cual tiene una propiedad llamada Support, encargada de almacenar el objeto al cual está vinculado un sketch, en caso de que el mismo esté creado con soporte en la cara plana de un sólido.

Como se muestra en la figura 6, las funcionalidades están agrupadas dentro de la pestaña Part, en la sección Pieza.

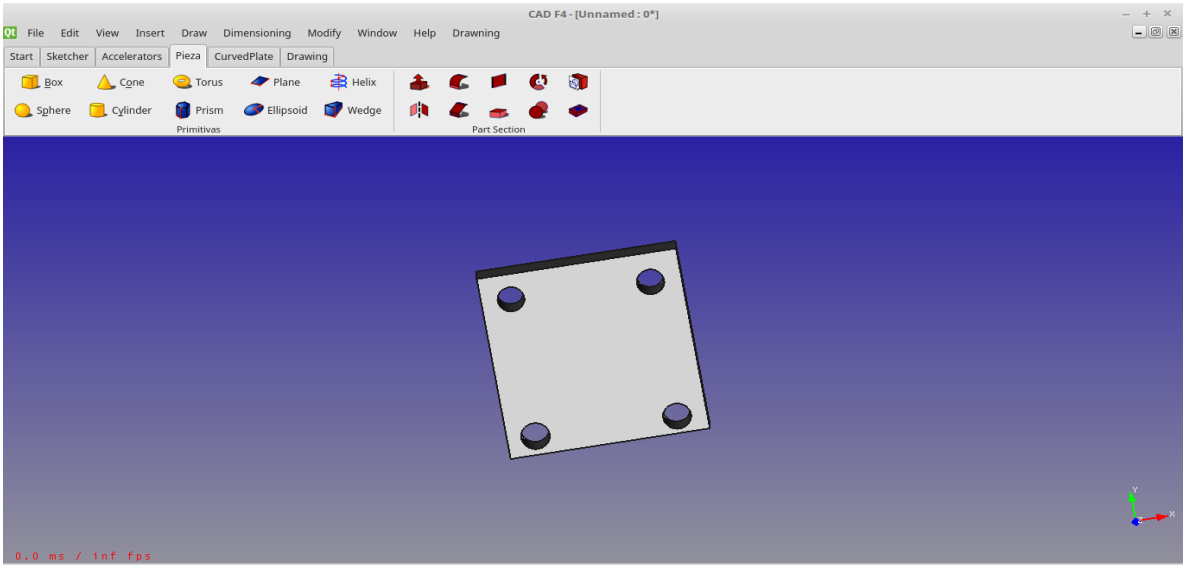


Figura 6: Operación “Pocket” en la sección Pieza.

Cada funcionalidad tiene una clase ViewProvider encargada de la visualización del objeto creado en el visor, además del ícono por el cuál va a estar representado dicho objeto. A continuación, se describe por cada una de las funcionalidades los parámetros que poseen:

Extrusión:

- **Base:** Objeto al cuál se le realiza la extrusión.
- **Dir:** Vector que indica la dirección y longitud de la extrusión.
- **Solid:** Propiedad que indica si el resultado del objeto a extruir va a ser un sólido o una superficie.

Reflejar característica:

- **Source:** Objeto al cual se le aplica la operación de reflejar característica.
- **Base:** Punto base puede ser usado para mover el plano de reflexión paralelo al plano estándar seleccionado. Sólo una de las componentes X, Y o Z es efectiva para un plano estándar dado.

Tabla 2: Efecto de aplicar la operación reflejar característica en el plano de reflexión.

Plano estándar	**Punto Base**	**Efecto**
XY	Z	Mueve el plano de reflexión a lo largo del eje Z.
XY	X, Y	No tiene efecto.
XZ	Y	Mueve el plano de reflexión a lo largo del eje Y.
XZ	X, Z	No tiene efecto.
YZ	X	Mueve el plano de reflexión a lo largo del eje X.
YZ	Y, Z	No tiene efecto.

- **Normal:** Dirección del plano de reflexión.

Redondeado:

- **Base:** Objeto al cual se le aplica la operación de redondeado.
- **Edges:** Aristas del objeto a las que se le aplica el redondeado.

Revolución:

- **Source:** Objeto al cual se le aplica la operación de revolución.
- **Base:** Coordenadas X, Y, Z que especifican cuánto se mueve el origen del eje de revolución, relativo al origen de coordenadas.
- **Axis:** Eje de revolución.
- **Angle:** Ángulo que especifica cuánto gira la revolución.
- **Solid:** Propiedad que indica si el resultado del objeto a revolucionar va a ser un sólido o una superficie.

Chaflanado:

- **Base:** Objeto al cual se le aplica la operación de chaflanado.
- **Edges:** Aristas del objeto a las que se le aplica el chaflanado.

Superficie reglada:

- **Curve1 y Curve2:** Aristas entre las cuales se crea la superficie reglada.

Espesor:

- **Thickness:** Define el espesor de las caras del objeto.
- **Faces:** Son las caras por donde se abre el hueco al objeto.
- **Skin:** Obtiene un elemento como un florero, sin cabeza, pero con la parte inferior.
- **Pipe:** Obtiene un elemento como un tubo, sin cabeza y sin fondo.

Operaciones booleanas:

- **Base y Tool:** Objetos con los cuales se realiza la operación booleana.

Sketch con soporte:

- **Support:** Objeto al cual está vinculado un sketch.

Vaciado:

- **Support:** Objeto al cual está vinculado un sketch.
- **Length:** Longitud del vaciado.

Para desarrollar las funcionalidades se utilizan las API de “Open CASCADE Community Edition”. Una API es un conjunto de subrutinas, funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción. Una API representa la capacidad de comunicación entre componentes de software. Se trata del conjunto de llamadas a ciertas bibliotecas que ofrecen acceso a ciertos servicios desde los procesos y representa un método para conseguir abstracción en la programación, generalmente (aunque no necesariamente) entre los niveles o capas inferiores y los superiores del software.

Uno de los principales propósitos de un API consiste en proporcionar un conjunto de funciones de uso general, por ejemplo, para dibujar ventanas o iconos en la pantalla. De esta forma, los programadores se benefician de las ventajas del API al hacer uso de su funcionalidad, evitándose el trabajo de programar todo desde el principio (27). A continuación, se describen algunas API utilizadas en la solución.

Tabla 3: APIs usadas en la propuesta de solución.

API	**Descripción**
BrepBuilderAPI_MakeWire	Construye un alambre dado un número de aristas.
BRepPrimAPI_MakePrism	Construye un prisma dado una figura y un vector de dirección.
BRepBuilderAPI_Transform	Aplica una transformación geométrica a una figura.
BRepBuilderAPI_MakeFace	Construye una cara.
BRepAlgoAPI_BooleanOperation	Ejecuta una operación booleana entre dos figuras.
BRepAlgoAPI_Cut	Realiza una operación booleana de corte entre dos figuras.
BRepBuilderAPI_MakeEdge	Crea una arista.

2.4 Requisitos

Los requisitos para un sistema son la descripción de los servicios proporcionados por el sistema y sus restricciones operativas. Estos requisitos reflejan las necesidades de los clientes de un sistema que ayude a resolver algún problema como el control de un dispositivo, hacer un pedido o encontrar información (28).

Los requisitos pueden clasificarse en funcionales y no funcionales. Los funcionales permiten conocer detalladamente las características y funciones del sistema. Los no funcionales dan a conocer las cualidades que deberá cumplir el sistema con el objetivo de brindar a los usuarios una mayor usabilidad, disponibilidad y rendimiento.

En la implementación de la solución no se tendrán en cuenta las operaciones de editar y eliminar un objeto creado mediante el uso de una funcionalidad, debido a que no es responsabilidad de las funcionalidades, sino del núcleo de la aplicación, de editar y eliminar los objetos mediante el árbol de entidades.

2.4.1 Requisitos funcionales

RF 1. Integrar el módulo Sketcher a la estructura del módulo pieza.

RF 2. Realizar la extrusión de una figura.

RF 3. Realizar el vaciado de una figura.

RF 4. Definir el espesor de una figura.

RF 5. Revolucionar una figura.

RF 6. Redondear las aristas seleccionadas de una figura.

RF 7. Hacer un chaflán en las aristas seleccionadas de una figura.

RF 8. Crear una superficie reglada entre dos aristas.

RF 9. Reflejar característica de un objeto.

RF 10. Crear sketch con soporte en la cara plana de un sólido.

RF 11. Realizar una operación booleana entre dos sólidos (Unión, Diferencia, Intersección, Sección).

2.4.2 Requisitos no funcionales

Software:

RNF 1. Sistema operativo basado en GNU-Linux de 64 bits.

RNF 2. Tener en el equipo la biblioteca Oce (Open CASCADE Community Edition).

Hardware:

RNF 3. RAM: 1Gb, Espacio libre en disco: 1Gb.

Funcionalidad:

RNF 4. Precisión: debe poseer una alta precisión, en cuanto a la cantidad de cifras significativas, en los datos introducidos y mostrados.

Usabilidad:

RNF 5. El sistema podrá ser utilizado por cualquier persona que posea conocimientos básicos de mecánica.

RNF 6. Se podrá cancelar cualquier operación desde la interfaz principal.

RNF 7. Comprensibilidad: al pasar el cursor sobre un botón se debe mostrar la operación que este representa, así como imágenes que apoyen la comprensión de dicha funcionalidad.

2.4.3 Historias de usuario

Las historias de usuario son tarjetas de papel en las cuales el cliente describe brevemente las características que el sistema debe poseer, sean requisitos funcionales o no funcionales. El tratamiento de las historias de usuario es muy dinámico y flexible, en cualquier momento se pueden modificar, reemplazar por otras más específicas o generales o añadirse nuevas. Cada historia de usuario debe ser lo suficientemente comprensible y delimitada para que los programadores puedan implementarla en unas semanas (29).

A continuación, se muestra una de las historias de usuario realizadas en la propuesta de solución. Para el estudio de las demás historias de usuario remitirse al Anexo B.

Tabla 4: Historia de Usuario “Realizar la extrusión de una figura”.

Número 2	**Nombre del requisito:** Realizar la extrusión de una figura
Programador: Raciel Perdomo Gómez	**Iteración Asignada:** 1era
Prioridad: Alta	**Tiempo estimado:** 7 días
Riesgo en Desarrollo: N/A	**Tiempo Real:** 7 días

Descripción:

1- **Objetivo:**

Permitir la extrusión de una figura dada.

2- **Acciones para lograr el objetivo (precondiciones y datos):**

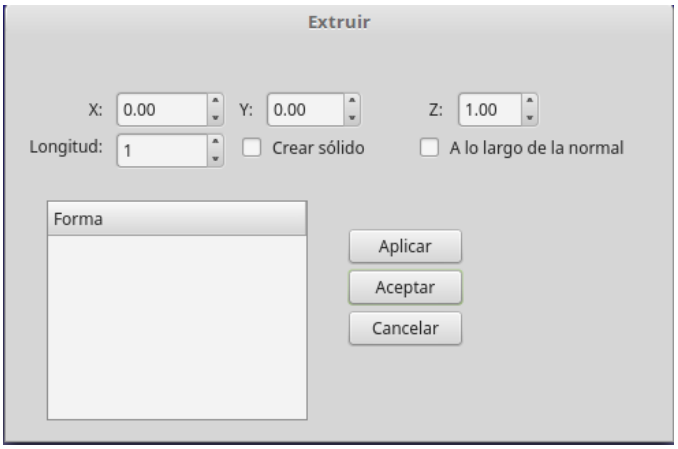
- Para realizar la extrusión de una figura, esta debe ser una cara plana que es la que se va a alargar sobre un eje definido.
- Se debe especificar un vector de dirección definido por un punto (X,Y,Z).
- Se debe especificar una longitud de extrusión.

3- **Comportamiento válidos y no válidos (flujo central y alternos):**

- Los valores de cada componente del vector de dirección deben ser números reales.
- Para realizar la extrusión debe estar seleccionado una figura que sea extruible.
- Si se marca el campo Along normal, el objeto a extruir tiene que ser un plano.

4- **Flujo de la acción a realizar:**

- El usuario introduce los valores correspondientes al vector de dirección (componente X, Y, Z del vector) y la longitud de extrusión.
- Al presionar el botón aplicar o aceptar, se realiza la extrusión, si se marca el campo Create solid, se crea un sólido, de lo contrario solo se extruyen las caras de la figura seleccionada.
- Al presionar el botón cancelar se cerrará la ventana.

Observaciones:
Prototipo de interfaz: 

2.5 Diseño

El diseño está definido según la IEEE (“Institute of Electrical and Electronics Engineers”) de dos formas: el proceso para definir la arquitectura, los componentes, las interfaces y otras características del sistema y el resultado del proceso (mencionado)” (30).

Según la segunda definición, el resultado del diseño debe describir la arquitectura utilizada, que es como el software se descompone y organiza; además de las interfaces y la descripción detallada de los componentes que posibiliten su posterior desarrollo (30).

2.5.1 Estilo y patrón arquitectónico del software

Un estilo arquitectónico es una transformación impuesta al diseño de todo un sistema y su objetivo es establecer una estructura para todos los componentes de dicho sistema (31).

La familia de estilos arquitectónicos seleccionada es Llamada y retorno pues permite que un diseñador de software obtenga una estructura de programa que resulta relativamente fácil modificar y cambiar de tamaño. Miembros de esta familia son las arquitecturas de programa principal y subrutina, los sistemas basados en llamadas a procedimientos remotos, los sistemas orientados a objeto y los sistemas jerárquicos en capas (31).

Dicho estilo se evidencia en la clase DlgExtrude, donde la rutina principal **on_accept_clicked** hace uso de varias subrutinas como **canExtrude, makeExtrusion** y

findShapes, donde se pasan los datos por parámetros y la subrutina le devuelve a la rutina principal el resultado esperado.

2.5.2 Arquitectura en capas

La arquitectura basada en capas se enfoca en la distribución de roles y responsabilidades de forma jerárquica proveyendo una forma muy efectiva de separación de responsabilidades. El rol indica el modo y tipo de interacción con otras capas, y la responsabilidad indica la funcionalidad que está siendo desarrollada. Entre las desventajas de este modelo es que la estructuración de los sistemas puede resultar difícil y el rendimiento se puede ver afectado, pues se puede incurrir en que una funcionalidad debe pasar por muchas capas para alcanzar la capa superior que solicitó su servicio (32).

Se decide el empleo de este modelo en la confección del componente porque soporta bien los cambios y el desarrollo incremental, además que es la arquitectura con la cual está implementado el núcleo y demás módulos de la aplicación CAD que se está desarrollando en el Grupo de Investigación. En el desarrollo de las funcionalidades, esta arquitectura se evidencia en las clases ViewProviders, las cuales pertenecen a la capa encargada de manejar la parte visual de la aplicación, como la visualización de los íconos y los objetos en el visor, la clase Extrusion la cual pertenece a la capa encargada de manejar los datos de las figuras, y la clase DlgExtrude, que pertenece a la capa de interfaces de las funcionalidades.

2.5.3 Patrones de diseño

Los patrones de diseño son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces. La propuesta de solución contendrá los siguientes patrones de diseño:

- **Experto:** mediante su uso, se asignan responsabilidades a la clase que cuenta con la información necesaria (33). Se evidenciará en la clase Extrusion, que se encarga de la extrusión de las figuras.
- **Creador:** ayuda a identificar quien debe ser el responsable de la creación de nuevos objetos (33). Se evidencia en la clase dlgextrude pues tiene la información necesaria para la creación de objetos tipo Extrusion.
- **Bajo acoplamiento:** estimula asignar una responsabilidad de modo que reduzca el impacto de los cambios y que no incremente el acoplamiento que es una medida de la fuerza en que las clases se relacionan (33). Se evidencia en las clases DlgExtrude

que usa las funcionalidades de la clase ViewProviderPart, que es usada por todas las clases de la solución, y cualquier cambio dentro de una de las funcionalidades de dicha clase no produce un gran impacto en el funcionamiento de la aplicación.

- **Alta cohesión:** es una medida de cuan relacionadas o enfocadas están las responsabilidades de una clase (33). Se evidencia en la relación entre las clases dlgextrude, y la clase Extrusion para la extrusión de figuras pues colaboran para producir un comportamiento bien definido.

2.5.4 Diagrama de clase del diseño

Un diagrama o modelo de clases en UML es un tipo de diagrama de estructura estática que describe la estructura de un sistema mostrando las clases del sistema, sus atributos, operaciones (o métodos), y las relaciones entre los objetos (herencia, agregación, asociación, entre otras). En la Figura 7, se muestra el diagrama de clase diseñado para la funcionalidad extrusión:

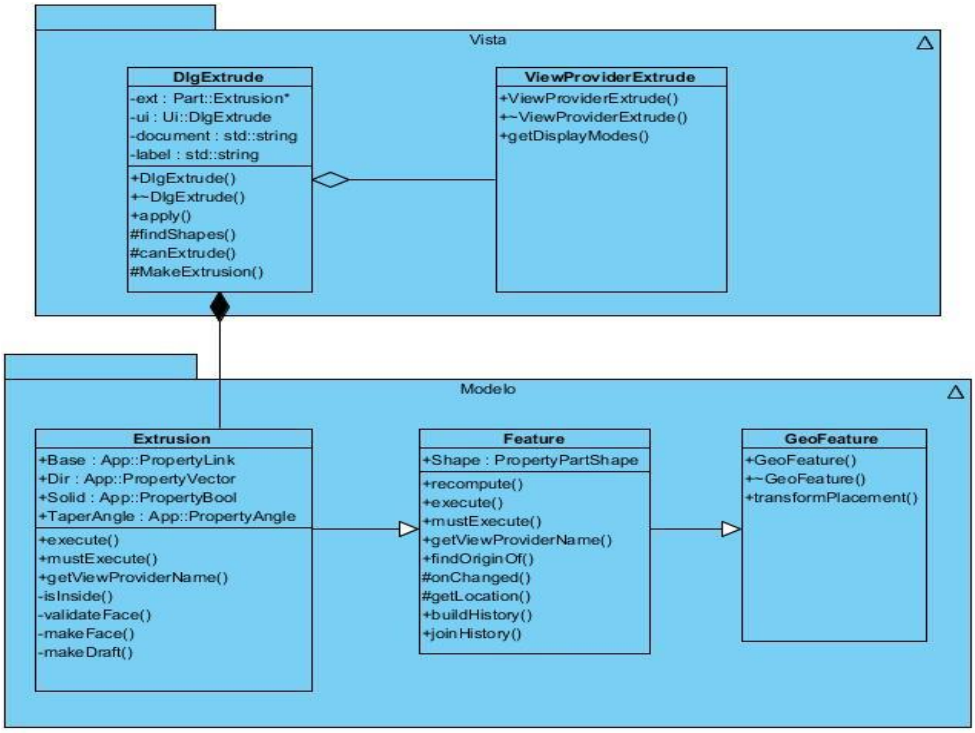


Figura 7: Diagrama de clases de funcionalidad Extrusión.

2.5.5 Diagrama de secuencia del diseño

Un diagrama de secuencia es una forma de diagrama de interacción que muestra los objetos como líneas de vida a lo largo de la página y con sus interacciones en el tiempo

representadas como mensajes dibujados como flechas desde la línea de vida origen hasta la línea de vida destino. Los diagramas de secuencia son buenos para mostrar qué objetos se comunican con qué otros objetos y qué mensajes disparan esas comunicaciones. Los diagramas de secuencia no están pensados para mostrar lógicas de procedimientos complejos (34).

En la figura 8, se muestra el diagrama de secuencia de la historia de usuario: Extruir figura.

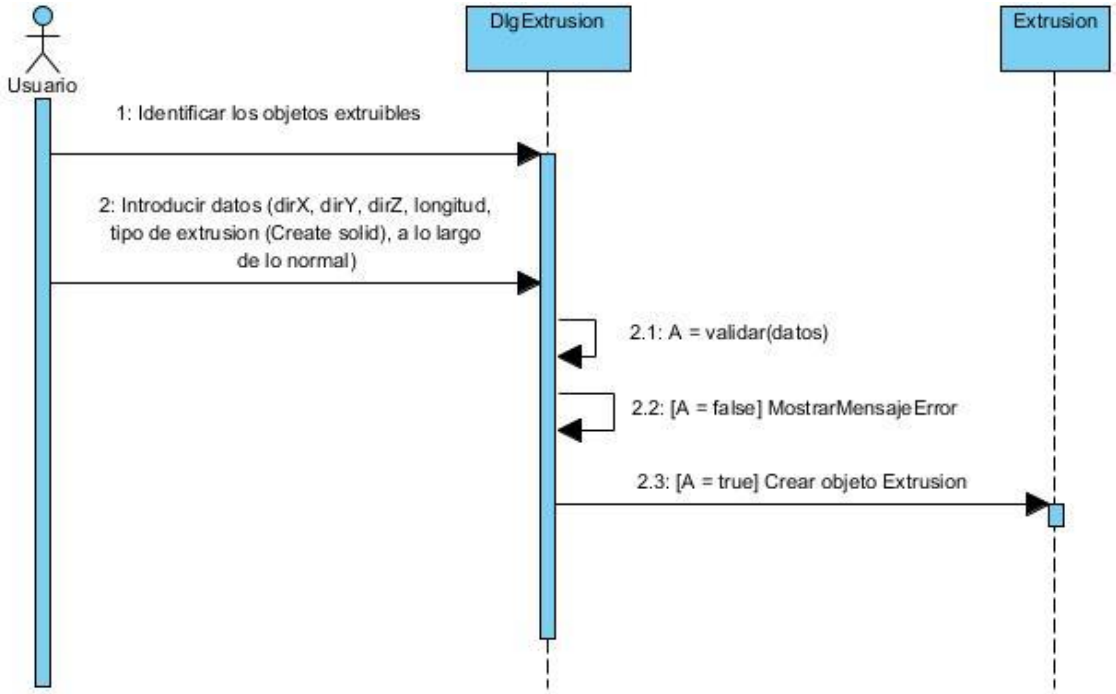


Figura 8: Diagrama de secuencia funcionalidad Extruir.

2.6 Consideraciones del capítulo:

- A partir del estudio del marco teórico se definió la propuesta de solución y mediante el uso de las herramientas y metodología definidos en el capítulo anterior fue posible realizar un levantamiento de requisitos descritos en historias de usuario permitiendo definir las funcionalidades a implementar para dar cumplimiento a los objetivos planteados al inicio de la investigación.
- Se definió el modelo de diseño de clases donde se aplicaron los patrones de diseño que permitirán el correcto desarrollo de las funcionalidades.
- El diseño arquitectónico más apropiado es una Arquitectura por Capas que permita la comunicación con el núcleo de la aplicación y con otros módulos de la misma.

- Los artefactos obtenidos en este capítulo sientan las bases para la implementación de la solución propuesta.

Capítulo 3: Implementación y pruebas

Una vez definida las Historias de Usuario en el capítulo anterior, el equipo de desarrollo cuenta con suficiente conocimiento sobre el sistema y está en condiciones de comenzar la implementación de las Historias de Usuarios según la planificación concebida. Paralelamente al desarrollo de las funcionalidades y también después de esto se realizarán las pruebas para detectar y corregir posibles errores. Además, se crea el diagrama de componentes del sistema y se definen los estándares de codificación empleados para el desarrollo de la solución.

3.1 Implementación

La fase de implementación del software posee como entradas los artefactos de la fase anterior (diseño), como: diagramas de clases, especificación de arquitectura, patrones a emplear en el sistema, entre otros. En la implementación se define el estándar de codificación a emplear, se realizan las implementaciones a las historias de usuarios, se define el diagrama de componentes del sistema, entre otras actividades.

3.1.1 Estándar de codificación

Un estándar de codificación es un conjunto de reglas o pautas utilizadas al escribir el código fuente de una aplicación. Seguir un estándar de codificación ayuda a los programadores a leer y entender el código fuente conforme al estilo, y ayuda a evitar la introducción de errores (35).

A continuación, se expone el estándar empleado en la solución.

Tabla 5: Estándar de codificación del componente.

Descripción	**Ejemplo**
Definición de Objetos, Clases, Funciones y Atributos	

Todos los nombres de las clases implementadas comenzarán con letra mayúscula. En caso de poseer un nombre compuesto se escribirán de acuerdo a la normativa CamelCase-UpperCamelCase.	class Foo{
	cuerpo de la clase
	}
	class FooFirst{
	cuerpo de la clase
	}

Siempre se declara para todas las clases implementadas su respectivo destructor de clase.	virtual ~Foo()
Las declaraciones de funciones siempre comenzarán en letra inicial minúscula. En caso de ser un nombre compuesto se regirá por la normativa CamelCase-lowerCamelCase.	<Tipo dato retorno> función () <Tipo dato retorno> funcionCompuesta () <Tipo dato retorno> funcionDobleCompuesta()
Los atributos siempre estarán escritos con letra minúscula. En caso de ser un nombre compuesto se regirá por la normativa CamelCase-loweCamelCase.	<Tipo dato> atributo; <Tipo dato> atributoNombreCompuesto();
Definición de parámetros dentro de las funciones y constructores de clases	
Los nombres de los identificadores de los parámetros en las funciones deben estar escritos con minúscula separados a un espacio cada uno y en caso de ser compuesto utilizar normativa CamelCaselowerCamelCase.	<Tipo dato retorno> función(<tipo><id1>, <tipo><id2>, <tipo><idN>)
Los identificadores de los parámetros dentro de los constructores de las clases deben estar escritos con minúscula separados a un espacio cada uno y en caso de ser compuesto utilizar normativa CamelCase-lowerCamelCase.	Clase(<tipo><id1>, <tipo><id2>, <tipo><idN>)
Definición de expresiones	

Para una mejor comprensión en la lectura y legibilidad del código los operadores binarios exceptuando los punteros, función de llamado a miembros, escritura de un arreglo y	x + y;
	x == y;
	idFuncion.miembro();

paréntesis de una función se escribirán con un espacio entre ellos.	idFuncion->miembro (); array[];
Definición de estructuras de control y bucles	

Las estructuras de control y los bucles estarán definidos de igual manera en ambos casos siguiendo el estándar determinado por el framework de Qt.	Para las estructuras if, else, if else:
	<estructura control>(condición){
	Bloque de código
	}
	Para los bucles while, for, do while y otros:
	<bucle>(condiciones){
	Bloque de código
	}

Comentarios en el código según el estándar de C++	
Comentarios pequeños.	//comentario sencillo
Otros comentarios	/* *Comentario */
Comentario de versión, descripción de clase y otras características de la clase o paquete.	/* ********** *Comentario amplio * ********** */

3.1.2 Diagrama de componentes

Un diagrama de componentes es un diagrama tipo del Lenguaje Unificado de Modelado. Representa cómo un sistema de software es dividido en componentes y muestra las dependencias entre estos componentes. Los componentes físicos incluyen archivos, cabeceras, bibliotecas compartidas, módulos, ejecutables, o paquetes. Los diagramas de

Componentes prevalecen en el campo de la arquitectura de software, pero pueden ser usados para modelar y documentar cualquier arquitectura de sistema (36).

En la Figura 9, se muestra el diagrama de componentes de la funcionalidad Extruir.

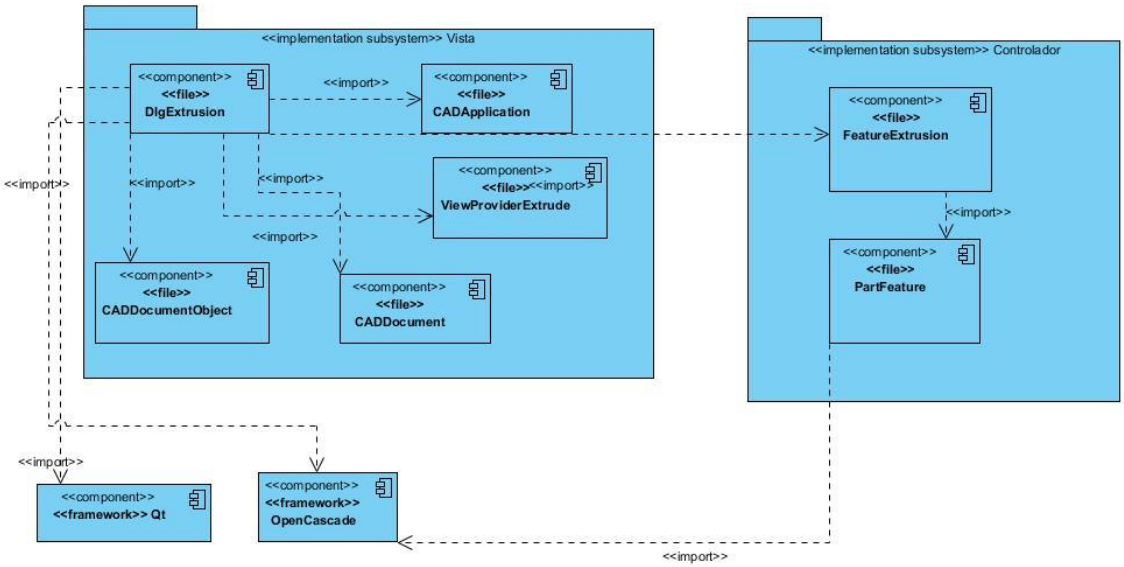


Figura 9: Diagrama de componentes funcionalidad Extruir.

3.2 Pruebas

Las pruebas de software son las investigaciones empíricas y técnicas cuyo objetivo es proporcionar información objetiva e independiente sobre la calidad del producto a la parte interesada. Es una actividad más en el proceso de control de calidad. Las pruebas son básicamente un conjunto de actividades dentro del desarrollo de software. En dependencia del tipo de pruebas, estas actividades podrán ser implementadas en cualquier momento de dicho proceso de desarrollo. Existen distintos modelos de desarrollo de software, así como modelos de pruebas. A cada uno corresponde un nivel distinto de involucramiento en las actividades de desarrollo (37).

3.2.1 Niveles de prueba

Para aplicarle pruebas a un sistema se deben tener en cuenta una serie de objetivos en diferentes escenarios y niveles de trabajo, debido a que las pruebas son agrupadas por niveles que se encuentran en distintas etapas del proceso de desarrollo (38). Los niveles de prueba son:

- **Prueba unitaria o de Unidad:** es una forma de comprobar el correcto funcionamiento de una unidad de código. Sirve para asegurar que cada unidad funcione correctamente y eficiente por separado. Además de verificar que el código

hace lo que tiene que hacer, se comprueba que sea correcto los nombres y tipos de los parámetros, el tipo de dato que devuelve. Las pruebas del camino básico y de bucles son técnicas muy efectivas para descubrir una gran cantidad de errores en los caminos (38).

- **Prueba de integración:** son aquellas que se realizan en el ámbito de desarrollo de software una vez que se han aprobado las pruebas unitarias y lo que prueban es que todos los elementos unitarios que componen el software, funcionan juntos correctamente probándolos en grupo. Se centra principalmente en probar la comunicación entre los componentes y sus comunicaciones ya sea hardware o software (38).
- **Pruebas de Sistema:** está constituida por una serie de pruebas diferentes cuyo propósito primordial es ejercitar profundamente el sistema basado en computadora. Aunque cada prueba tiene un propósito diferente, todas trabajan para verificar que se han integrado adecuadamente todos los elementos del sistema y que realizan las funciones apropiadas (38). Algunas de estas son: pruebas de recuperación, seguridad, resistencia, entre otras.
- **Pruebas de aceptación:** también conocida como prueba de aceptación del usuario, es un tipo de ensayos se realicen con el fin de verificar si el producto ha sido desarrollado de acuerdo con las normas y criterios establecidos y cumple con todos los requisitos especificados por el cliente. Este tipo de pruebas se lleva a cabo generalmente por un usuario o cliente donde se desarrolla el producto externamente por otra parte (38).

Para validar el correcto funcionamiento de la herramienta se le realizarán pruebas Unitarias, de Sistema y de Aceptación.

3.2.2 Métodos de prueba

El principal objetivo del diseño de casos de prueba es obtener un conjunto de pruebas que tengan la mayor probabilidad de descubrir los defectos del software. Para llevar a cabo este objetivo, se emplearán los dos métodos de prueba:

- **Prueba de Caja Blanca:** se centran en los detalles procedimentales del software, por lo que su diseño está fuertemente ligado al código fuente. El ingeniero de pruebas escoge distintos valores de entrada para examinar cada uno de los posibles flujos de ejecución del programa y cerciorarse de que se devuelven los valores de

salida adecuados. Al estar basadas en una implementación concreta, si ésta se modifica, por regla general las pruebas también deberán rediseñar (31).

- **Prueba de Caja Negra:** son diseñadas para validar los requisitos funcionales sin fijarse en el funcionamiento interno de un programa, solo se fijan en las funciones que realiza el software. Las técnicas de prueba de caja negra se centran en el ámbito de información de un programa, de forma que se proporcione una cobertura completa de prueba (31).

3.2.3 Diseño de Casos de Prueba

Los Casos de Pruebas han sido realizados sobre la base de las Historias de Usuarios y tienen como objetivo fundamental encontrar la mayor cantidad posible de deficiencias existentes en las funcionalidades implementadas. A continuación, se presenta el Caso de Prueba de la Historia de usuario Extruir figura, para un mayor estudio de estos ver los Anexos.

Tabla 6: Caso de Prueba de la Historia de Usuario “Extruir figura”.

<table>
<tr><td colspan="4">Descripción general</td></tr>
<tr><td colspan="4">Permitir la extrusión de una figura dada.
Condiciones de ejecución
- Para realizar la extrusión de una figura, esta debe ser una cara plana que es la que se va a alargar sobre un eje definido.
- Se debe especificar un vector de dirección definido por un punto (X, Y, Z).
- Se debe especificar una longitud de extrusión.</td></tr>
<tr><td colspan="4">SC 2 Extruir figura</td></tr>
<tr><td>Escenario</td><td>Descripción</td><td>Respuesta del sistema</td><td>Flujo central</td></tr>
<tr><td>EC 1.1 Aceptar</td><td>Se introducen los datos (vector de dirección, longitud de extrusión, sólido), y se</td><td>Extruye la figura plana correspondiente.</td><td>Mod/Part/Gui/DlgExtrude

Mod/Part/App/FeatureExtrusion</td></tr>
</table>

	selecciona la opción aceptar.		
EC 1.2 Aplicar	Selecciona la opción aplicar	Extruye la figura seleccionada sin cerrar la vista.	Mod/Part/Gui/DlgExtude Mod/Part/App/FeatureExtrusion
EC 1.3 Cancelar	Selecciona la opción Cancelar.	Se elimina la vista creada, y se restaura el filtro de la selección.	Mod/Part/Gui/DlgExtrude

3.2.4 Pruebas Unitarias

Para la realización de las pruebas unitarias se empleó Qt Test, el cual es un framework para pruebas de este tipo a aplicaciones y librerías. Qt Test provee todas las funcionalidades comunes de los framework de pruebas unitarias, así como extensiones para probar interfaces gráficas de usuario, (39). En la figura 6, se muestra una imagen de las pruebas unitarias realizadas durante el proceso de implementación, en la que se evidencia la aceptación de todas las verificaciones:

```
********* Start testing of TestUnitTest *********
Config: Using QTest library 4.8.6, Qt 4.8.6
PASS   : TestUnitTest::initTestCase()
PASS   : TestUnitTest::testCanExtrude()
PASS   : TestUnitTest::testMakeExtrusion()
PASS   : TestUnitTest::testFindShapes()
PASS   : TestUnitTest::testApply()
PASS   : TestUnitTest::testMakeFillet()
PASS   : TestUnitTest::testMakeChamfer()
PASS   : TestUnitTest::setupFillet()
PASS   : TestUnitTest::testOn_filletStartRadius_valueChanged()
PASS   : TestUnitTest::testOn_filletEndRadius_valueChanged()
PASS   : TestUnitTest::cleanupTestCase()
Totals: 11 passed, 0 failed, 0 skipped
********* Finished testing of TestUnitTest *********
```

Figura 10: Resultado de prueba unitaria realizada con Qt Test.

A continuación, se muestra la implementación de una prueba unitaria realizada a uno de los métodos de la funcionalidad extruir.

```
#include <QtTest/QtTest>
#include "TopoDS_Shape.hxx"
#include "TopExp_Explorer.hxx"
#include "mod/Part/App/PrimitiveFeature.h"

class TestPart : public QObject
{
    Q_OBJECT
public:
    TestPart();
    virtual ~TestPart();

private slots:
    void testCanExtrude();
};

TestPart::TestPart(){}
TestPart::~TestPart(){}
void TestPart::testCanExtrude ()
{
    Part::Circle *c = new Part::Circle();
    c->Radius.setValue(5.0);
    TopoDS_Shape shape = c->Shape.getValue();
    int output = 0;
    if(shape.IsNull()){
        if(!output)output = 1;
    }else{
        TopAbs_ShapeEnum type = shape.ShapeType();
        if(type == TopAbs_VERTEX || type == TopAbs_EDGE ||
            type == TopAbs_WIRE || type == TopAbs_FACE ||
            type == TopAbs_SHELL){
            if(!output)output = 2;
        }else{
            if(type == TopAbs_COMPOUND)
            {
                TopExp_Explorer xp;
                xp.Init(shape,TopAbs_SOLID);
                while(xp.More()) {
                    if(!output)output = 1;
                }
                xp.Init(shape,TopAbs_COMPSOLID);
                while(xp.More())
                {
                    if(!output)output = 1;
                }

                if(!output)output = 2;
            }
```

```
        }
    }
     QCOMPARE(output, 2);
}
```

3.2.5 Resultados de las pruebas

La realización de pruebas funcionales permitió identificar una serie de no conformidades:

Tabla 7: Detección de No Conformidades.

No. NC	**Requisito Funcional**	**Descripción**	**Complejidad**	**Estado**
1	RF 1	No se oculta la figura plana una vez realizada la extrusión	Media	Resuelta
2	RF 1	La aplicación no da la salida esperada si se pasa un vector de dirección no válido (componentes negativas)	BAJA	Resuelta
3	RF 3	Una vez hecho el redondeado de las aristas de una figura, no se puede seleccionar nada que no sea una arista en el visor.	MEDIA	Resuelta
4	RF 3	Las casillas de verificación de las aristas no se actualizan si se hace la selección directamente desde el visor.	BAJA	Resuelta
5	RF 4	Una vez hecho el chaflán de las aristas de una figura, no se puede seleccionar nada que no sea una arista en el visor.	MEDIA	Resuelta

6	RF 3	Una vez modificado el valor del radio no se envía la señal para notificar dicho cambio	BAJA	Resuelta

En la figura 11, se evidencia la realización de las pruebas, haciéndose un total de tres iteraciones. En la primera iteración se realizan 11 pruebas unitarias y 14 pruebas funcionales, detectándose 4 no conformidades. En una segunda iteración se realizan11 pruebas unitarias y 19 pruebas funcionales, detectándose 2 no conformidades. En la tercera y última iteración, se realizan 9 pruebas unitarias y 13 pruebas funcionales, detectándose 0 no conformidades.

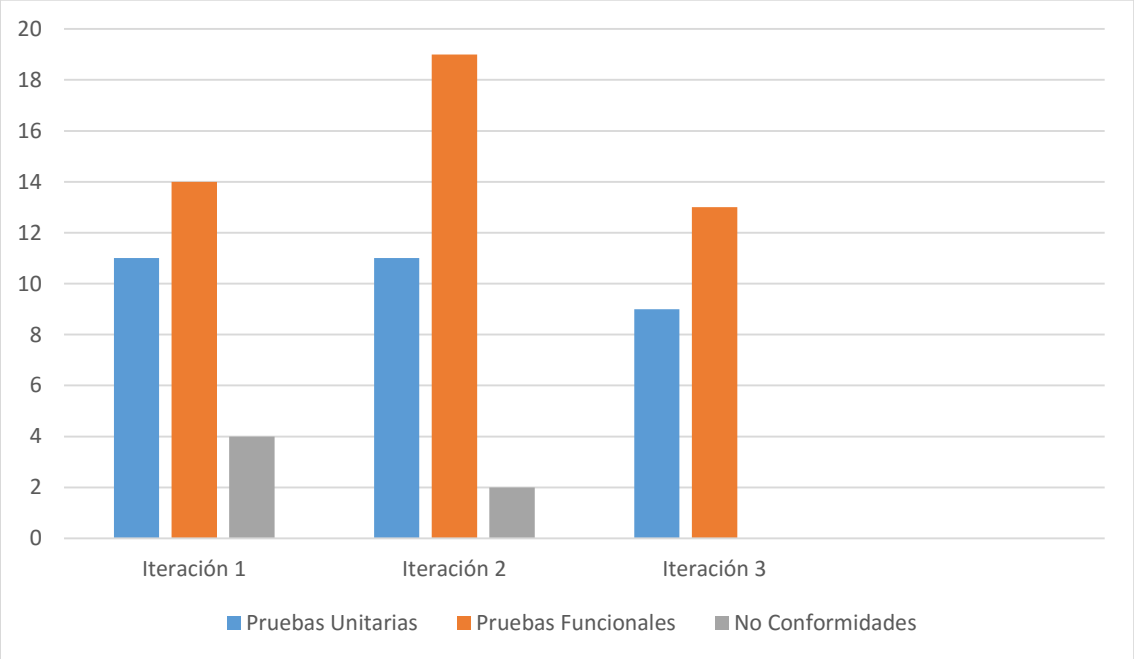


Figura 11: Iteraciones realizadas en el proceso de pruebas.

3.3 Consideraciones parciales del capítulo

- Se hizo referencia a los aspectos fundamentales en la implementación y la validación de las funcionalidades para el modelado paramétrico de piezas y componentes mecánicos en tres dimensiones.
- Se presentó el estándar de codificación que se utilizó según las normas establecidas.
- Mediante la realización de pruebas, se verificó el correcto funcionamiento del software.

Conclusiones generales

Como resultado del proceso de Investigación-Desarrollo realizado se arribó a las siguientes conclusiones generales:

- El agrupamiento de las funcionalidades básicas para el modelado de piezas en tres dimensiones, destinado a su integración en una aplicación CAD, tuvo su base en el análisis integral y reutilización del código fuente de la aplicación “FreeCAD”, en correspondencia con decisiones de proyecto en el grupo de investigación.
- El diseño arquitectónico más apropiado para el desarrollo de las funcionalidades para el modelado a ser empleadas en sistemas de Diseño Asistido por Computadora debe basarse, en lo fundamental, en una Arquitectura por capas que permita la comunicación con el núcleo de la aplicación y con otros módulos de la misma.
- La eficacia de las funcionalidades implementadas, está avalada por los resultados de las pruebas realizadas, que demostraron el correcto funcionamiento de cada una.

Recomendaciones

A partir del estudio realizado y luego de haber analizado los resultados obtenidos, se recomiendan los siguientes elementos a tener en cuenta para futuros trabajos:

- Optimizar la funcionalidad de operaciones booleanas para realizar operaciones entre más de dos figuras.
- Incorporar las funcionalidades para reproducir instancias de características, mediante patrones rectangulares, circulares y combinados.
- Desarrollar las funcionalidades de ayuda al modelado que no fueron abordadas en el trabajo (puntos, líneas y planos de construcción).
- Diseñar interfaces que propicien la explotación fácil y eficiente de las funcionalidades mencionadas.

Referencias bibliográficas

1. **Santos, Sandy García.** *Componente para la modelación de engranajes cilíndricos de dientes rectos.* 2015.

2. **FreeCAD: an open-source parametric 3D CAD modeler. [En línea] [Citado el: 19 de enero de 2017.]**

3. Part Module. [En línea] www.freecadweb.org.

4. Portal, 3D CAD. [En línea] [Citado el: 2017 de mayo de 15.] http://www.3dcadportal.com/asixmec-aplicacion-cubana-para-el-diseno-e-ingenieria-asistida-por-computadora.html.

5. Teresco, John. IndustryWeek. [En línea] 4 de febrero de 2017. http://www.industryweek.com/articles/parametric_technology_corpbrwaltham_mass_268.aspx.

6. Parametric and Feature-Based Modeling. [En línea] [Citado el: 19 de Marzo de 2017.] http://alum.wpi.edu/~gregm/thesis/node11.html.

7. Solid Dna Community. [En línea] 8 de febrero de 2017. http://www.soliddna.com/SEcommunity/page/Solid%20Edge%20Wiki/Home.html/_/solid-edge-wiki/extrude-st.

8. Weisstein, Eric W. Sólido de Revolución. MathWorld. [En línea] 14 de febrero de 2017. http://mathworld.wolfram.com/SolidofRevolution.html.

9. Catia_Dassault_Systèmes. Catia. [En línea] www.3ds.com/products/catia.

10. Catia Serwis Informacyjny. [En línea] 9 de marzo de 2017. http://www.catia.com.pl/tutorial/z2/part_design.pdf.

11. Autodesk. CONSTRUCTION LINE. AUTOCAD Architecture. [En línea] 2016. knowledge.autodesk.com.

12. SolidWorks. Modelado de sólidos en 3D. [En línea] www.solidwork.es.

13. SolidWorks_Corporation. SolidWorks. [En línea] 1993. www.solidworks.com.mx.

14. www.salome-platform.org. Salome. [En línea]

15. Metzger, Michael y Eismann, Sabine. Freeform Surface Modeling. [En línea] [Citado el: 18 de abril de 2017.] http://www.hpl.hp.com/hpjournal/95oct/oct95a6.pdf.

16. Dictionary. [En línea] [Citado el: 20 de abril de 2017.] http://www.dictionary.com.

17. Monografías. [En línea] [Citado el: 20 de abril de 2017.] http://www.monografias.com/trabajos77/metrologia-avanzada-tolerancias-geometricas/metrologia-avanzada-tolerancias-geometricas2.shtml..

18. Introducción al Modelado Sólido. [En línea] [Citado el: 2017 de abril de 21.] http://di002.edv.uniovi.es/~rr/Tema7.pdf.

19. Boundary Representations. [En línea] [Citado el: marzo de 14 de 2017.] https://www.cs.mtu.edu/~shene/COURSES/cs3621/NOTES/model/b-rep.html.

20. An Interactive Introduction to Splines. [En línea] [Citado el: 20 de marzo de 2017.] http://ibiblio.org/e-notes/Splines/Intro.htm.

21. UCI. *Metodología de desarrollo para la Actividad productiva de la UCI.* Ciudad de la Habana : s.n., 2015.

22. Solis A, Camilo J y Figueroa D, Robert G. *Metodologías de Desarrollo de SW.* 2011.

23. Jacobson, Ivar, Rumbaugh, James y Booch, Grady. *El lenguaje unificado de modelado. Manual de referencia.* 2000.

24. Stroustrup, Bjarne. *The C++ Programming Language.* New Jersey : AddisonWesley, 1997. 0201889544.

25. Synergix. WordPress. [En línea] [Citado el: 3 de Abril de 2017.] https://synergix.wordpress.com/2008/07/03/tareas-construir-modelo-de-dominio/.

26. Wikiteka. [En línea] https://www.wikiteka.com/apuntes/dibujo-tecnico-21/.

27. TechTarget. [En línea] 22 de abril de 2017. http://searchmicroservices.techtarget.com/definition/application-program-interface-API.

28. Sommerville, Ian. *Ingeniería de Software. Octava Edición.* 2006.

29. Jeffries, R., Anderson, A y Hendrickson, C. *Extreme Programming Installed.* s.l. : Addison-Wesley, 2001.

30. Society, IEE Computer. *Guide to the Software Engineering Body of Knowledge.* 2004.

31. Pressman, Roger S. *Ingeniería de software. Un enfoque práctico.*

32. Overview, OpenGL. OpenGL. [En línea] [Citado el: 14 de enero de 2017.] https://www.opengl.org/about/#1.

33. Larman, Craig. *UML y Patrones. Introducción al análisis y diseño orientado a objetos.* Mexico : Prentice Hall, 1999. ISBN 970-17-0261-1.

34. Sparx Systems. [En línea] [Citado el: 23 de enero de 2017.] http://www.sparxsystems.com.ar/resources/tutorial/uml2_sequencediagram.html.

35. Estándares de codificación. [En línea] [Citado el: 28 de abril de 2017.] https://www.ohmyroot.com/buenas-practicas-legibilidad-del-codigo/.

36. IVAR JACOBSON, JAMES RUMBAUGH y BOOCH, GRADY. El Lenguaje Unificado de Modelado. Manual de Referencia. 2000.

37. SEDICI. Repositorio Institucional de la UNLP. [En línea] [Citado el: 28 de abril de 2017.] http://sedici.unlp.edu.ar/handle/10915/34969.

38. Departamento de Lenguajes y Sistemas Informáticos. Universidad de Sevilla. [En línea] [Citado el: 28 de abril de 2017.] http://www.lsi.us.es/docencia/get.php?id=361.

39. Qt Test Overview | Qt Test 5.6. [En línea] [Citado el: 5 de Mayo de 2017.] http://doc.qt.io/qt-5/qtest-overview.html .

40. Historia del diseño asistido por computadora. [En línea] 14 de 09 de 2014.

41. Instituto Superior Politécnico José Antonio Hecheverría. [En línea] 11 de 02 de 2015. www.cujae.edu.cu.

42. Estrada, José Angel Lores. *Módulo de transformación a entidades 2D.* La Habana : s.n., 2012.

43. Inventor, Autodesk. Autodesk Inventor. [En línea] 1999. www.autodesk.com.

44. Salazar, Miguel Arcia. *Desarrollo de un componente visual para Qt utilizando el framework OpenCascade.* Ciudad de la Habana : s.n., 2011.

45. ArchiCAD_Graphisoft. ArchiCAD. [En línea] 1982. www.graphisoft.com/products/archicad.

46. SolidEdge_Intergraph. Solid Edge. *Siemens PLM Software.* [En línea] 1996. www.siemens.com/solidedge.

47. Allen, Robert y Garlan, David. *A formal approach to software architectures.* 1992.

48. White-Box Testing. [En línea] [Citado el: 29 de abril de 2017.] http://agile.csc.ncsu.edu/SEMaterials/WhiteBox.pdf.

49. Ruiz Tenorio, Roberto. *Las pruebas de software y su importancia en las organizaciones.* Veracruz: Facultad de Contaduría y Administración. Universidad Veracruzana : s.n., 2010.

50. Reynoso, Carlos y Kicillof, Nicolás. *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.* Universidad de Buenos Aires : s.n., 2004.

51. Stroustrup, Bjarne. *The C++ Programming Language. Third Edtition.* 3ra. AT&T Labs Murray Hill, New Jersey : s.n., 1997. ISBN 0-201-88954-4.

52. Rodríguez Sánchez, Tamara. *Metodología de desarrollo para la Actividad productiva de la UCI.* La Habana, Cuba: Universidad de las Ciencias Informáticas : s.n., 2015.

53. CCCC. [En línea] 18 de marzo de 2017. http://cccc.sourceforge.net.