UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

FACULTAD 5

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERO EN CIENCIAS INFORMÁTICAS

Título:

Extrusión de entidades 2D a lo largo de curvas abiertas en el visor 3D de la herramienta AsiXMec 1.0.

Autor: Adiel Pacheco Malagón.

Tutor: Ing. Juan Carlos Ayala Alonso.

Co-tutores: Ing. José Ángel Lores Estrada. Ing. Karel Piorno Charchabal.

Ciudad de la Habana, 4 de julio de 2016.

"Año 57 de la Revolución."

DECLARACIÓN DE AUTORÍA

Declaro ser autor de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

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

_____

Firma del Autor: Adiel Pacheco Malagón.

_____

Firma del Tutor: Ing. Juan Carlos Ayala Alonso.

_____

Firma del Co-Tutor: Ing. José Ángel Lores Estrada.

_____

Firma del Co-Tutor: Ing. Karel Piorno Charchabal.

Datos de contacto

Tutor: Ing. Juan Carlos Ayala Alonso.

Edad: 28

Ciudadanía: cubana

Institución: Universidad de las Ciencias Informáticas (UCI)

Título: Ingeniero en Ciencias Informática

Categoría Docente:

E-mail: jcayala@uci.cu

Co-Tutor: Ing. José Ángel Lores Estrada

Edad: 27

Ciudadanía: cubana

Institución: Universidad de las Ciencias Informáticas (UCI)

Título: Ingeniero en Ciencias Informática

Categoría Docente: Instructor

E-mail: lores@uci.cu

Co-Tutor: Ing. Karel Piorno Charchabal.

Edad: 28

Ciudadanía: cubana

Institución: Universidad de las Ciencias Informáticas (UCI)

Título: Ingeniero en Ciencias Informática

Categoría Docente: Instructor

E-mail: kpiorno@uci.cu

Dedicatoria

A los forjadores de mi carácter: mi abuelo y mi padre.
A mi mamá, por siempre ser tan especial para mí y amarme tanto.

Agradecimientos

Agradecer a mis padres y a mi abuelo, a ellos le debo todo lo que soy.

Agradecer a mi hermana y mi novia por su cariño y comprensión.

Agradecer a mi familia, mis abuelos, mis tíos, mis primos, a todos por el amor y el cariño.

Agradecer a mis tutores por su paciencia y ayuda, sin ellos no hubiera sido posible la realización de este trabajo.

A mis amigos por haberme aguantado todos estos años.

Resumen

El proyecto Diseño y Simulación de Estructuras Mecánicas del Centro Vertex, perteneciente a la Universidad de las Ciencias Informáticas se encuentra desarrollando la segunda versión de la herramienta de Diseño Asistido por Computadoras “AsiXMec 1.0”. En el estado actual de AsiXMec 1.0, existen piezas importantes para la ingeniería que no pueden realizarse, como cables, tuberías entre otras. El presente trabajo tiene como objetivo agregarle a la herramienta AsiXMec 1.0 un módulo para extruir entidades 2D sobre curvas abiertas en el visor 3D de AsiXMec 1.0, o lo que es lo mismo, la funcionalidad “Solevado”. Para esto se utiliza el lenguaje de programación C++ en el Entorno de Desarrollo Integrado QtCreator con el *framework* Qt y la biblioteca OpenCascade. Como resultado principal se obtiene un módulo integrado a AsiXMec 1.0 sobre el cual se puede realizar la operación Solevado.

Palabras clave: curva abierta, entidad 2D, extruir.

Abstract

The Vertex project Design and Simulation of Mechanical structures belonging to the University of Informatics Sciences is developing the second version of the tool of Assisted Design by Computer “AsiXMec 1.0”. In the AsiXMec 1.0 current state there are important pieces that cannot be develop such as cables, pipes and hooks among others. The aim of this Diploma Paper is to add to AsiXMec 1.0 a module to extrude 2D entities over open curves in the AsiXMec 1.0’s 3D viewer or its Spanish name “Solevado”. To achieve this, the C++ language was used in the Integrated QtCreator with the Qt framework and the OpenCascade library. As result, we obtain a module integrated to AsiXMec 1.0 over which the Loft operation can be done.

Key words: open curve, to extrude, 2D entities.

Índice

Introducción

Con el actual desarrollo de las Tecnologías de la Información y las Comunicaciones (TIC) y gracias al nacimiento de la Informática Gráfica (IG), se ha logrado visualizar y modelar sólidos en dos y tres dimensiones (2D y 3D). Esto ha dado origen a un nuevo paradigma de representación visual conocido como Diseño Asistido por Computadoras (*Computer Aided Design CAD*). En los últimos años se ha evidenciado un auge del uso y dependencia de modeladores bidimensionales y tridimensionales. Son muchos los ejemplos de utilización de *software* de este tipo, entre los cuales puede mencionarse la animación en el cine, la elaboración de planos en la arquitectura, la creación de piezas para la industria, entre otros.

El proyecto Diseño y Simulación de Estructuras Mecánicas (DISEM) del Centro Vertex, perteneciente a la Universidad de las Ciencias Informáticas (UCI), se encuentra en el desarrollo de nuevos módulos para la herramienta AsiXMec 1.0.

AsiXMec 1.0 es un sistema para el diseño y visualización de prototipos virtuales de piezas mecánicas, puede llegar a ser de gran utilidad en la industria, en el desarrollo de investigaciones, en la educación superior y en otros sectores claves de la economía, así como para otros usuarios y seguidores del CAD. Es un modelador que permite a los usuarios manipular, crear, editar, importar y exportar archivos .STEP, .IGES, .STL, compatibles con otras aplicaciones similares para los sistemas operativos Linux y Windows. Su interfaz gráfica tiene soporte multilenguaje (inglés y español) y cuenta con menú en forma de *ribbon*. Como modelador paramétrico, no debe ser confundido con los programas tradicionales de CAD. La ventaja fundamental del CAD paramétrico radica en que la geometría del modelo es controlada por parámetros que definen su diseño en tamaño y forma, normalmente estos parámetros son variables como

alto, ancho, profundidad, o son usados como fórmulas. Los parámetros se almacenan en un contenedor que se encuentra en el archivo de diseño, al modificar alguno de estos parámetros el diseño cambia para reflejar la modificación.

AsiXMec 1.0 provee funcionalidades que ayudan al usuario en el proceso de modelación, como los *snap* en el proceso de creación de entidades 2D, la previsualización de todas las operaciones y facilidades para la selección de los componentes visualizados. Se captura el intento de diseño mediante las restricciones 2D gestionadas por un *solver* especializado y el mecanismo de parametrización que posibilita la edición y manejo del historial de operaciones en tres dimensiones. Cuenta con herramientas para mediciones, gestión de dimensiones y un editor de parámetros.

Con una arquitectura basada en *plugins* posee una elevada capacidad de extensibilidad y adaptabilidad para la incorporación de nuevos módulos y la creación de sistemas especializados o a la medida. Actualmente el *software* solo ha sido desarrollado con bibliotecas multiplataforma aunque hasta ahora solo ha sido compilado en Ubuntu, pero se ha evitado cualquier dependencia que en el futuro pueda frenar la compilación en Windows.

Esta herramienta, cuya primera versión ya fue liberada, cuenta con un módulo que se encarga de realizar extrusiones. En una herramienta CAD, la extrusión no es más que extender una entidad 2D en profundidad para obtener a partir de ella una superficie o cuerpo 3D (Lores y otros, 2013). Actualmente el módulo de extrusión de AsiXMec 1.0 permite realizar extrusiones solamente a lo largo de la dirección definida por la propia cara a extruir, no siendo así con la extrusión a lo largo de la dirección definida por cualquier curva, lo que imposibilita a los diseñadores e ingenieros mecánicos realizar un gran número de diseños de piezas mecánicas como tuberías, conductos de ventilación, ganchos, cables. Lo

anteriormente planteado es de gran importancia, pues le brindaría a la industria de una herramienta de producción nacional con similar nivel de las más utilizadas internacionalmente. Debido a lo anteriormente planteado se hace necesaria la implementación de una funcionalidad que permita la extrusión a lo largo de curvas abiertas, para hacer un *software* más completo y funcional.

Dada la **problemática** antes expuesta, se formula el siguiente **problema de la investigación:** ¿Cómo lograr la extrusión de entidades 2D a lo largo de curvas abiertas en el visor 3D de la herramienta AsiXMec 1.0?

En aras de resolver el problema planteado se define como **objeto de estudio:** La extrusión de entidades 2D, definiendo como **campo de acción** la extrusión de entidades 2D a lo largo de una curva abierta.

Con el propósito de dar solución al problema de la investigación, se plantea como **objetivo general:** Agregar al módulo de extrusión de la herramienta AsiXMec 1.0 una funcionalidad que permita realizar extrusiones de entidades a lo largo de curvas abiertas.

Con el fin de darle cumplimiento al objetivo general propuesto, se proponen las siguientes **tareas de la investigación:**

- Revisión bibliográfica para generar el marco teórico conceptual de la investigación que represente las principales definiciones y conceptos relacionados con el tema y las tendencias en herramientas similares.
- Realización del levantamiento de requisitos funcionales.
- Diseño de un modelo de clases que responda a las exigencias del módulo a desarrollar.
- Implementación del módulo que brinde la solución al problema planteado.
- Realización de pruebas al módulo implementado.

Para la realización de la investigación y elaboración del presente trabajo se emplearon los siguientes **métodos científicos de investigación:**

Métodos teóricos:

Análisis Histórico-Lógico: Mediante este método se analizó la evolución y desarrollo del objeto de la investigación, identificando las características generales y esenciales de su funcionamiento.

Analítico-Sintético: Se utilizó al analizar toda la información relacionada con el tema de tesis, para extraer los elementos más importantes de cada documento consultado.

Modelación: Se empleó para realizar una representación simplificada de la realidad a través de diagramas.

Métodos empíricos:

Observación científica: Se empleó para la identificación del problema.

Pruebas: Se realizaron diferentes pruebas al módulo implementado para determinar si se comporta según los resultados esperados.

El documento está estructurado de la siguiente forma:

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

El Capítulo 2, Propuesta de solución, incluye la descripción, diseño y análisis de la solución que se propone para darle respuesta a la problemática planteada. Se especifican los requisitos funcionales y los no funcionales.

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

Capítulo 1 Fundamentación Teórica

1.1 Introducción

En el presente capítulo se expone el marco teórico de la investigación. Para darle solución al problema propuesto es necesario realizar un grupo de acciones investigativas con el objetivo de comprender el campo de acción que se plantea. De esta forma se tiene una base de conocimiento sólida, la cual proporciona la ayuda necesaria para poder adentrarse en el desarrollo de una solución al problema existente. Para cumplir con este objetivo se divide el presente capítulo en epígrafes los cuales tratarán los puntos del proceso de revisión de esta investigación. En un momento inicial se expondrá una panorámica sobre las herramientas CAD y la operación extrusión en la cual quedarán plasmadas sus características generales. Posteriormente se analizarán un grupo de herramientas similares y la metodología de desarrollo de *software*. Seguidamente se expondrá la tecnología seleccionada así como la herramienta de Ingeniería de Software Asistida por Computadoras (CASE) utilizada.

1.2 Herramientas CAD

Antes de la década del 80 del siglo pasado los diseños industriales producidos en el mundo eran realizados con lápiz sobre un papel como un dibujo común. Para realizar correcciones se borraba la parte incorrecta del diseño y se redibujaba. Para realizar cambios muy grandes, era necesario rehacer completamente el diseño. En ocasiones, luego de fabricado el producto, se descubren deficiencias en su diseño que dan al traste con la imposibilidad de su puesta en explotación. Para solucionar este problema surgen los sistemas CAD, los cuales han evolucionado hasta la actualidad.

El CAD es la aplicación de la informática al proceso de diseño y atiende prioritariamente aquellas tareas exclusivas del diseño, tales como el dibujo técnico y la documentación del mismo, además permite realizar otras tareas complementarias relacionadas principalmente con la presentación y el análisis del diseño realizado (Salmon, y otros, 1987).

Las herramientas CAD utilizan modeladores geométricos para manejar las entidades. Los modeladores geométricos en 2D generan geometrías y perfiles que se utilizan posteriormente para crear superficies y sólidos. Son herramientas poderosas que permiten modelar todo tipo de objetos. Los modeladores pueden ser 3D, en este caso su complejidad y sus potencialidades aumentan. Según sus propios sitios web, con los modeladores de estas herramientas se pueden realizar operaciones y transformaciones como la traslación, escalado, rotación, extrusión, revolución entre otras (Autodesk Corporation, 2016) (Inventor, 2016) (SolidWork, 2016).

1.3 Extrusión

La extrusión es un proceso utilizado para crear objetos con sección transversal definida y fija (Gualoto, 2015). Mediante esta operación se obtiene una superficie con la forma deseada. En la industria fue Joseph Bramah quien en 1797 patentó el primer proceso de extrusión para hacer un tubo de plomo. Éste consistía en el precalentamiento del metal para luego pasarlo por un troquel mediante un émbolo a mano. El proceso no fue desarrollado hasta 1820, cuando Thomas Burr construyó la primera prensa hidráulica. Hasta ese momento el proceso se llamó *squirting*. En 1894 Alexander Dick expandió el proceso de extrusión al cobre y aleaciones de bronce (Sheppard, T. 1999).

Existen varias formas de extruir, siendo la extrusión en línea recta la más conocida. Esta consiste en “estirar” una figura plana en una dirección, que frecuentemente coincide con uno de los ejes de coordenadas. Esta funcionalidad está presente en AsiXMec 1.0, pero no se ha podido implementar la extrusión de caras a lo largo de una curva abierta, también conocida por el nombre de solevado.

Los objetos solevados son formas bidimensionales extruidas a lo largo de un tercer eje (Aldaya y Fernández, 2015). Estos objetos se crean a partir de dos o más objetos *spline*s existentes. Una de estas *splines* será el recorrido y el resto de *splines* actúan como secciones transversales, o formas, del objeto solevado. Se crean objetos de forma que servirán como recorrido para cualquier número de formas de sección transversal. El recorrido se convierte en la estructura que soporta las secciones transversales que forman el objeto. Si el recorrido sólo cuenta con una forma, el programa asume que hay una forma idéntica en cada extremo del recorrido. De este modo, se genera la superficie entre las formas.

1.4 Herramientas homólogas

Actualmente existen disímiles herramientas para la creación de animaciones y modelos 3D. Las herramientas de diseño gráfico permiten realizar el modelado, la manipulación de objetos y entornos virtuales. El conocimiento de estas herramientas para la investigación que se realiza es muy importante porque permite identificar elementos que pudieran ser útiles para realizar la propuesta de solución. A continuación se expondrá una pequeña reseña de algunas de las más importantes.

1.4.1 Google SketchUp

Es un *software* de modelado 3D, que permite a los diseñadores explorar, comunicar y presentar complejos conceptos de diseño. El conjunto de herramientas y el sistema de dibujo inteligente facilitan la creación y manipulación de modelos, mientras que las funciones de importación y exportación ofrecen la flexibilidad y funcionalidad necesarias para el proceso de trabajo profesional. SketchUp fue diseñado con el objetivo de que pudiera usarse de una manera intuitiva y flexible. SketchUp permite conceptualizar y modelar imágenes en 3D de edificios, coches, personas y cualquier objeto o artículo que imagine el diseñador o dibujante. Además el programa incluye una galería de objetos, texturas e imágenes listas para descargar. SketchUp posee un conjunto de herramientas para priorizar la facilidad de uso y aprendizaje. Es ideal para poner “en papel” o boceto, una idea. Todos los modelos SketchUp están formados por dos elementos: aristas y caras. Las aristas son líneas rectas y las caras son formas bidimensionales que se crean cuando varias aristas forman un bucle plano (Google Co., 2016).

Google SketchUp posee el botón Sígueme, que realiza la operación de solevar. Para realizar esta operación se siguen los siguientes pasos, ver Figura 1 (SketchUp, 2016):

1. Identifique la arista de la geometría que desea modificar. Esta arista marcará el recorrido a seguir.
2. Dibuje un perfil de la cara que desee que siga el recorrido. Asegúrese de que este perfil sea aproximadamente perpendicular al recorrido.

3. Seleccione la herramienta Sígueme en el menú Herramientas. El cursor adquiere la forma de un cuadro inclinado con una flecha.

4. Haga clic en el perfil que ha creado.

5. Arrastre el ratón por el recorrido.

6. Haga clic de nuevo para ejecutar el comando Sígueme cuando llegue al final del recorrido.

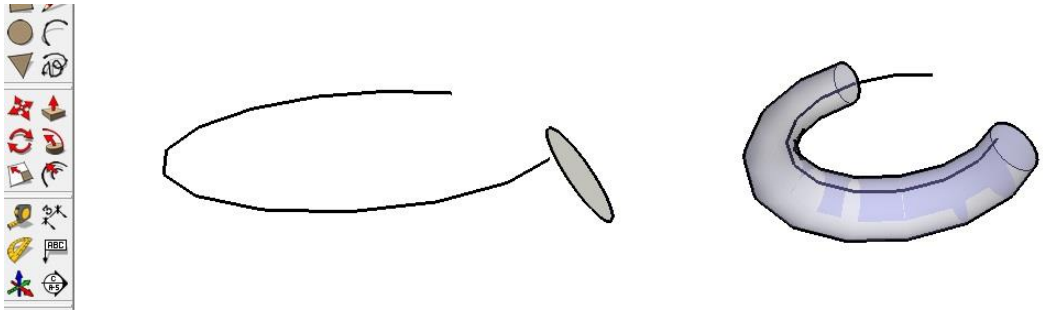

Figura 1: Solevado en Google SketchUp.

Esta operación es similar a la que se desea implementar. Su principal característica radica en que le brinda al usuario la posibilidad de solevar el objeto por una parte de la curva y no a todo lo largo de ella. Esto a su vez puede verse como una deficiencia, pues además de carecer de automatización, le deja al usuario toda la responsabilidad de la exactitud del resultado final.

1.4.2 Blender

Blender es una *suite* de código abierto para la creación de contenido 3D, disponible para los principales sistemas operativos bajo la Licencia Pública General (GPL). Originalmente desarrollado por la compañía *"Not a Number"* (NaN), el programa inicialmente fue distribuido de forma gratuita pero sin el código fuente, con un manual disponible para la venta aunque posteriormente pasó a ser *software* libre (Blender Foundation, 2015).

Está orientado a artistas y profesionales del diseño y multimedia, puede ser usado para crear visualizaciones 3D estáticas o vídeos de alta calidad. Blender es una *suite* completamente integrada y multiplataforma, que permite el modelado, animación de los modelos, además de la creación y reproducción de videojuegos; todo esto en un paquete fácil de descargar, además de estar disponible para las plataformas: Windows, Linux, Irix, Sun Solaris, FreeBSD o Mac OS X (Motores de Juego, 2007).

Posee herramientas que pueden ser usadas mediante un atajo de teclado, disminuyendo el tiempo de realización. Posee la mayoría de las funcionalidades existentes en las herramientas CAD. Ejemplo de esto son las herramientas: Subdividir, Extruir, Rotar, Escalar, Mover, Suavizar, Girar, entre muchas otras. Cuenta con una importante serie de modificadores para lograr efectos deseados en los modelos, como Bevel, Array, Boolean, Curve, Displacement. A diferencia de muchos programas privativos como el 3D Max y el Maya, posee una herramienta de Esculpir (Blender Foundation, 2015).

En Blender el proceso de extrusión es intuitivo. Trabajando en el modo de edición, se selecciona el objeto que se desea extruir, que pueden ser vértices, lados o caras, se presiona el botón Extrude Individual y con el mouse se arrastra hasta donde es necesario llevar la extrusión. También se puede utilizar la tecla E para extruir. Para lograr una extrusión a lo largo de una curva abierta se puede utilizar el modificador Array, que permite crear una serie de repeticiones del objeto base, obteniendo estructuras complejas a partir de una sencilla, ver Figura 2. Ésta solución no es viable para AsiXMec 1.0 ya que este es un modelador paramétrico, además, lo que se desea no es copiar un objeto varias veces, sino extruirlo.

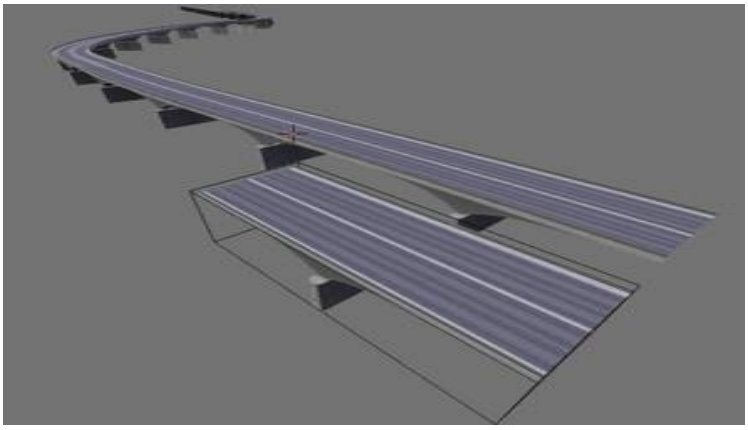

Figura. 2: Carretera creada usando el modificador Array.

1.4.3 Autodesk Inventor

Autodesk Inventor ofrece una gama completa y flexible de programas para diseño mecánico en 3D, simulación de productos, creación de herramientas, ingeniería a la carta y comunicación de diseños. Inventor lleva más allá de la tercera dimensión hacia prototipos digitales ya que permite producir un modelo 3D de gran precisión que puede ayudar a diseñar, visualizar y simular los productos antes de ser construidos. La creación de prototipos digitales con Inventor contribuye a que las compañías puedan diseñar mejores productos, reducir los costos de desarrollo y llegar al mercado más rápido (Inventor, 2016).

La operación de Solevación en Inventor permite crear sólidos o superficies mediante el barrido de un perfil a lo largo de una trayectoria 2D o 3D. El perfil (secciones) puede ser una polilínea, un *spline*, un círculo, una elipse, una cara de un sólido. La trayectoria (railes) puede ser una línea, un arco, una polilínea, un *splin*e. Para realizarlo se define uno o varios perfiles así como las trayectorias necesarias. Posteriormente se selecciona la opción Solevación donde se eligen los elementos deseados para realizar el solevado así como algunas opciones

para la creación del mismo. Se tiene la opción de crear un cuerpo sólido o solo la cubierta de este, también se puede elegir si la trayectoria pasará por el centro de los perfiles o, si hay más de una trayectoria, si se ajustará exactamente a estos, ver Figura 3. Para realizar estos solevados se requieren varias curvas y varias caras. Esta es una solución compleja teniendo en cuenta que en esta investigación se pretende alcanzar este resultado pero utilizando una sola cara y una sola curva.

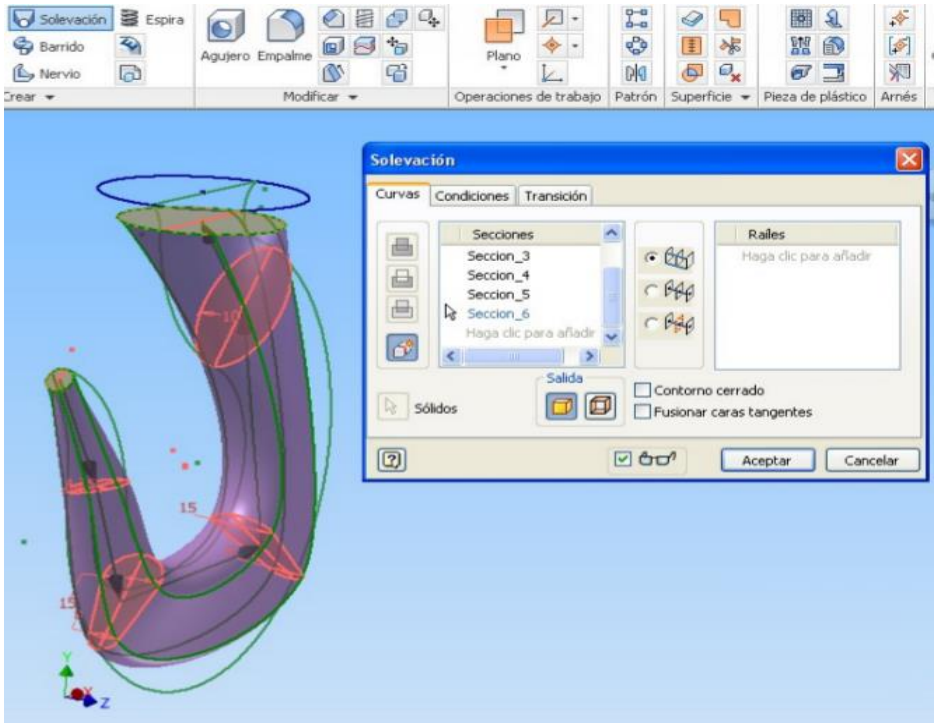

Figura 3: Creación de un solevado en Inventor.

1.4.4 SolidWork

SolidWork es un *software* de diseño mecánico en 3D que utiliza un entorno gráfico basado en Microsoft Windows desarrollado por *Dassault Systèmes S.A.* y lanzado en 1995. Las soluciones de SolidWork cubren todos los aspectos del proceso de desarrollo de productos con un flujo de trabajo integrado que incluye

las etapas de diseño, validación, diseño sostenible, comunicación y gestión de datos. Los diseñadores y los ingenieros pueden abarcar fácilmente varias disciplinas, lo que acorta el ciclo de diseño, aumenta la productividad y agiliza la introducción de los productos en el mercado (SolidWork, 2016). Además, utiliza el gestor de diseño (*Feature Manager*) que facilita la modificación rápida de operaciones tridimensionales y de croquis de operación sin tener que rehacer los diseños ya plasmados en sus documentos asociados.

En SolidWork existe un botón nombrado Recubrir con el que se realizan solevados. Para realizarlo se crea el perfil. Posteriormente se selecciona una curva que se cruce con el perfil y se selecciona la opción Recubrir, ver Figura 4.

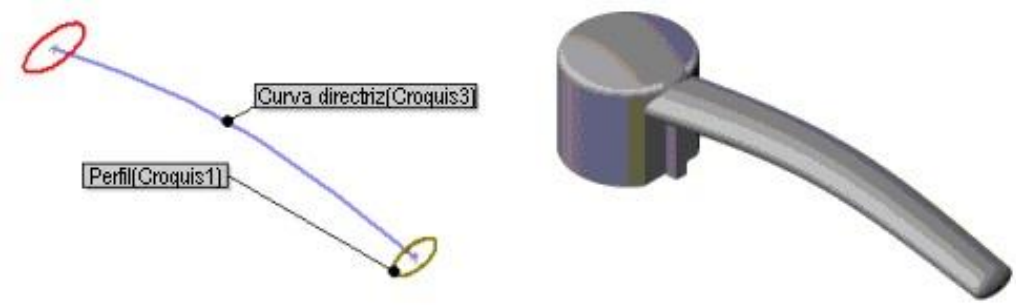

Figura 4: Creación de un solevado en SolidWork.

1.4.5 Autodesk 3D Max

3D Studio Max (conocido como 3D Max) es una herramienta que permite crear modelados y animaciones en 3D, a partir de una serie de vistas. Fue desarrollado por Autodesk Media & Entertaiment y es uno de los programas de animación 3D más utilizado y respetado en todo el mundo. 3D Max tiene unas capacidades de edición fuerte, una arquitectura de plugins y una larga presencia en la plataforma de Microsoft Windows, entre otras características (Díaz, y otros, 2010).

Es uno de los más sencillos para iniciarse en el mundo de la animación 3D para la creación de video, juegos y multimedia. Su arquitectura abierta, su baja curva de aprendizaje y sus potentes herramientas lo convierten en uno de los programas líderes del diseño y la animación 3D en varios de ámbitos, como: arquitectura, publicidad, televisión y video, cine, artes escénicas, desarrollo de juegos. Permite además exportar y guardar a varios formatos, incluso diferentes de los que trae por defecto la herramienta. Este programa exige un ordenador potente, aunque puede ejecutarse en máquinas inferiores, pero con un menor rendimiento. Este es un *software* propietario por lo tanto se dificulta su licencia de uso por su elevado costo, el cual ronda los $1470 USD (Autodesk Corporation, 2016).

En 3D Max puede usarse el modificador *Sweep* para realizar la extrusión de un objeto a lo largo de una curva abierta. Otra solución a este problema es la utilización de un objeto *Loft* en 3D Max. Los objetos de tipo *Loft* permiten crear estructuras tridimensionales a partir de 2 *splines* o figuras bidimensionales. De estas 2 figuras, una se designa como el recorrido (*Path*) y la otra como la forma (*Shape*) la cual se desplazará siguiendo el recorrido.

Para crear un *Loft*, es necesario tener en la escena al menos dos *Splines*. Se puede comenzar tanto por el *Path*, como por el *Shape*. Se selecciona primero una *spline*, después se va a *Create* > *Geometry* > *Compound Objetc* > *Loft*, ver Figura 5 y una vez allí, en la ventana *Creation Metod*, se selecciona la otra *spline*, ver Figura 6. Esta es la solución que se considera más cercana al objetivo propuesto para realizar la operación que se desea implementar. Se utilizará una combinación de la forma de realizar un solevado en 3D Max e Inventor.

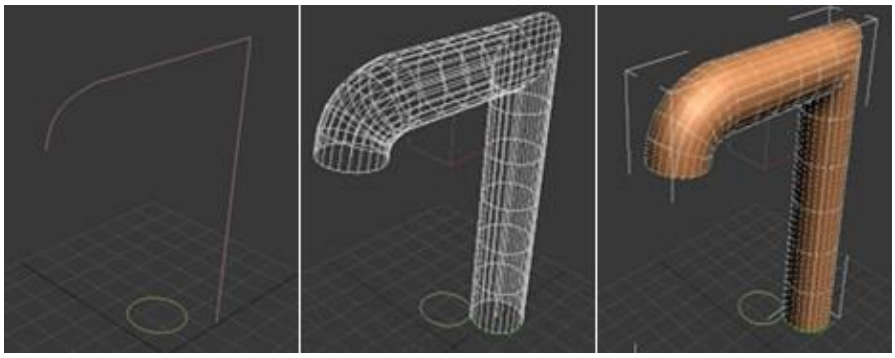

Figura 5: Creación de un *Loft*.

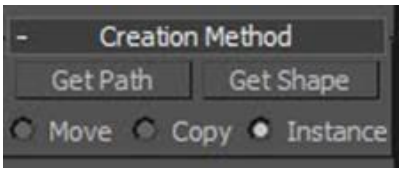

Figura 6: *Creation Method.*

A partir de aquí se puede controlar el suavizado de la superficie con el parámetro *Smoothing* que redondea las caras contiguas, sin marcar la arista y puede tener 2 valores:

Smooth Length: Suaviza a lo largo, entre los distintos pasos de *Path*, ver Figura 7.

Smooth Width: Suaviza a lo ancho, entre los pasos de la forma, ver Figura 8.

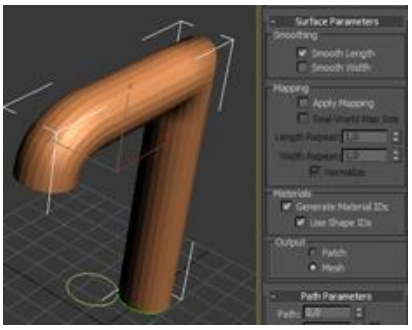

Figura 7: *Smooth Length*.

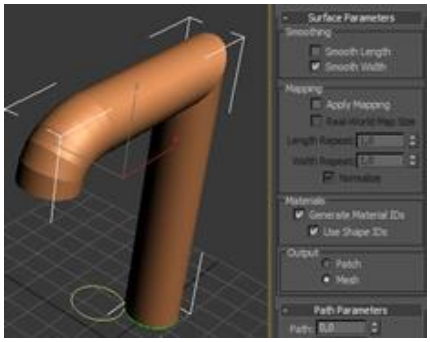

Figura 8: *Smooth Width*.

1.5 Metodología de desarrollo de *software*

Una metodología de desarrollo de *software* es un conjunto de procedimientos, técnicas, herramientas y soporte documental que sirve para guiar el proceso de desarrollo de *software*. Se basa en uno de los modelos de procesos o una combinación de ellos. Puede seguir uno o varios modelos de ciclo de vida, es decir, el ciclo de vida indica qué es lo que hay que obtener a lo largo del desarrollo del proyecto (Pressman, 2002). Dentro del proceso de desarrollo de *software*, la metodología de desarrollo de *software* se define como un conjunto de procedimientos, técnicas, herramientas y un soporte documental que ayuda a los desarrolladores a realizar un nuevo *software* con calidad (Méndez, 2010) (Piattini, 1996).

Teniendo en cuenta lo antes expuesto, puede entenderse una metodología de desarrollo de *software* como una guía útil para saber qué hay que hacer, cuando y cómo debe hacerse. Por este motivo, la selección de la metodología adecuada se puede considerar un paso importante para el desarrollo del proyecto. Hoy en día existen varias metodologías, las que se clasifican en dos tipos principales: las metodologías tradicionales y las metodologías ágiles.

A continuación se procede a la caracterización de la metodología usada para el desarrollo de la aplicación.

1.5.1 XP

La Programación Extrema es una metodología ligera de desarrollo de *software* que se basa en la simplicidad, la comunicación y la realimentación o reutilización del código desarrollado. Está centrada en potenciar las relaciones interpersonales como clave para el éxito en el desarrollo de *software*, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores y propiciando un buen clima de trabajo (Letelier y otros, 2006).

Los principios y prácticas que propone esta metodología son de total sentido común pero llevadas al extremo y de ahí proviene su nombre. En *"Extreme Programming Explained. Embrace Change",* Kent Beck, padre de XP, describe la filosofía de trabajo sin cubrir los detalles técnicos y de implantación de las prácticas. Posteriormente, otros autores de experiencias como Martin Fowler, Abrahamsson, Salo, Ronkainen y el propio Kent Beck se han encargado de dicha tarea (Abrahamsson, y otros, 2002). XP se plantea dos objetivos fundamentales: la satisfacción del cliente y potenciar al máximo el trabajo en grupo. Construye un proceso de diseño evolutivo que se basa en mejorar un sistema simple en cada iteración. Todo el diseño se centra en la iteración actual y no se hace nada anticipadamente para necesidades futuras, resultando en un proceso de diseño disciplinado y adaptable de una manera que indiscutiblemente la hace la más desarrollada entre todas las metodologías ágiles (Stephens, y otros, 2003).

La metodología XP posee cuatro características esenciales: historias de usuario, roles, prácticas y fases. Las historias de usuario son el artefacto utilizado en XP para especificar los requisitos del *software*. Se trata de tarjetas de papel en las cuales el cliente describe brevemente las características que el sistema debe poseer. El tratamiento de las historias de usuario es muy dinámico y flexible, en cualquier momento las historias de usuario pueden romperse, reemplazarse por

otras más específicas o generales, añadirse nuevas o ser modificadas. Cada historia de usuario es lo suficientemente comprensible y delimitada para que los programadores puedan implementarla en unas semanas (Beck, y otros, 2000).

En cuanto a los roles, al ser una metodología que se adapta con facilidad a las características y necesidades del proyecto, es posible que no siempre se haga la misma distribución. En la propuesta original de la metodología, Beck propone 7 roles: Programador, Cliente, Probador, Encargado de Seguimiento, Entrenador, Consultor y Jefe. Las prácticas que utiliza XP no son específicas de esta metodología, sin embargo es en XP donde se consigue el mayor beneficio de ellas mediante su aplicación conjunta y equilibrada puesto que se apoyan unas en otras, supliendo sus fallas entre sí (Beck, y otros, 2000). Las prácticas propuestas son: El juego de la planificación, entregas pequeñas, metáforas, diseño simple, pruebas, refactorización, programación en parejas, propiedad colectiva del código, integración continua, 40 horas por semana, cliente in-situ y estándares de programación. El ciclo de vida ideal de XP se divide en seis fases: Exploración, Planificación de la Entrega, Iteraciones, Producción, Mantenimiento y Muerte del Proyecto (Beck, y otros, 2000).

1.6 Selección de tecnología

Las herramientas de desarrollo de *software* desempeñan un papel primordial en la creación de aplicaciones. A lo largo de la vida de un proyecto de desarrollo de *software*, el programador puede utilizar diversas tecnologías y herramientas, las cuales muchas veces son definidas por supervisores o seleccionada por expertos. A continuación de exponen las herramientas que se consideran las más adecuadas para desarrollar la aplicación.

1.6.1 C++

Actualmente el proyecto DISEM perteneciente al Centro Vertex utiliza el lenguaje C++ para desarrollar sus aplicaciones. Además es el lenguaje en el que mayor experiencia se tiene por parte del equipo de desarrollo. Permite la programación estructurada y con la utilización de este lenguaje el programador tiene el control total de lo que está haciendo. Proporciona facilidades para la programación orientada a objetos y para el uso de plantillas o programación genérica (CPlusPlus, 2015). Por estas cualidades es que se elige este lenguaje para el desarrollo de la solución que se desea obtener mediante el transcurso de este trabajo.

1.6.2 *Framework* Qt

Es una biblioteca multiplataforma para desarrollar interfaces gráficas de usuario. Utiliza C++ de forma nativa, aunque además permite programar en otros lenguajes. Corre sobre las plataformas más usadas mundialmente. Además de posibilitar el desarrollo de interfaces gráficas de usuario, incluye otras series de características como son el acceso a base de datos SQL, *parser* de XML, y una API unificada multiplataforma para el manejo de ficheros (QtSoftware, 2015).

Cuenta con un Entorno de Desarrollo Integrado (IDE de sus siglas en inglés) propio llamado QtCreator, que se adapta a las necesidades de los desarrolladores de Qt y en él se juntan tanto las bibliotecas Qt como últimas herramientas de desarrollo adicionales como parte del SDK de Qt entre las que se encuentra el QtDesigner, una herramienta de Qt para el diseño y la creación de interfaces gráficas de usuario (GUI), por lo que ofrece todo lo necesario para empezar con el desarrollo de Qt en una sola instalación multiplataforma (QtSoftware, 2015). En la última década, Qt ha pasado de ser un producto

utilizado por unos pocos a uno que es utilizado diariamente por miles de clientes y decenas de miles de desarrolladores de código abierto en todo el mundo (Blanchette, y otros, 2008).

1.6.3 Biblioteca OpenCASCADE

OpenCASCADE (abreviatura de las siglas en inglés ”*Computer Aided Software for Computer Aided Design and Engineering*”), es una biblioteca para Qt desarrollada inicialmente a principios de los años 90 por “Matra Datavision”, utilizada por profesionales del área del diseño, como diseñadores, ingenieros y arquitectos. Permite diseñar y modificar modelos bidimensionales o tridimensionales con facilidad. Esta biblioteca incluye componentes elaborados en C++ para el modelado de sólidos y superficies, visualización, intercambio de datos y desarrollo rápido de aplicaciones (OpenCascade, 2016). Está especializada en la creación de herramientas CAD/CAE/CAM proporcionando un conjunto de funciones que facilitan el desarrollo de este tipo de herramientas. Open CASCADE incorpora además un paquete completo de efectos especiales y texturas que se pueden añadir a los diseños de modo que se puedan personalizar por completo (Mixprogramas, 2016).

1.7 Herramientas CASE

Visual Paradigm para UML (VP-UML) es una herramienta de diseño UML (*Unified Modeling Language*, Lenguaje de Modelado Unificado) y herramienta CASE (*Computer Aided Software Engineering*, Ingeniería de *Software* Asistida por Computadora) diseñada para la ayuda al desarrollo de *software*. VP-UML soporta estándares de la industria claves, tales como UML, SysML, BPMN, XMI,

ofrece un completo conjunto de herramientas de los equipos de desarrollo de *software* necesario para la captura de requisitos, la planificación de programas, la planificación de controles, la clase de modelado, modelado de datos entre otros (VisualParadigm, 2011).

VP-UML presenta un número de características positivas que sustentan su uso dentro del proceso de desarrollo de *software*. Es una herramienta profesional que soporta el ciclo de vida completo del desarrollo de *software*: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Es multiplataforma y posibilita la creación de ingeniería inversa (VisualParadigm, 2011).

Conclusiones Parciales

En este capítulo se analizaron los conceptos fundamentales con los que está relacionada la investigación. De los tipos de extrusión existentes, la investigación se centra en la extrusión a lo largo de curvas abiertas. Además, se analizaron herramientas similares así como las principales metodologías de desarrollo *de soft*ware existentes seleccionándose XP como la metodología a utilizar. También se abordó sobre la tecnología a utilizar y la herramienta CASE que se emplea.

Capítulo 2 Análisis y Diseño

2.1 Introducción

En el presente capítulo se presentan las dos primeras fases de la metodología XP (exploración y planificación) en las que se verá reflejado todo el análisis y diseño del proyecto. Mediante las historias de usuario definidas por el cliente, se describen cada uno de los requisitos funcionales así como su prioridad respecto a la implementación. Luego se estiman los esfuerzos de cada funcionalidad y en la planificación de iteraciones se procede a seleccionar qué funcionalidad se implementa en cada iteración. Finalmente se completa todo el diseño correspondiente a la propuesta en cuestión, detallando todos los aspectos relacionados con la arquitectura del sistema y el estándar de código definido.

2.2 Flujo Actual del Proceso

Fases de XP.

El proceso de desarrollo en XP generalmente cuenta con 4 fases y tiene un éxito total cuando el cliente selecciona correctamente el valor del negocio basándose en la capacidad del equipo para entregar el producto a tiempo. En primer lugar están las fases de Exploración, Planificación de la Entrega, en las cuales se describirán cada una de las funcionalidades mediante las historias de usuario confeccionadas por el cliente, además de definir la prioridad y la estimación del esfuerzo necesario para cada una. Se realizará además un plan de entrega y una planificación de las historias a realizar en cada iteración.

Una vez finalizadas estas etapas, le siguen las fases de Implementación y Pruebas, en las que se aplicarán las pruebas adicionales además de tomar las decisiones necesarias en relación a los cambios introducidos en la versión

actual. Además el sistema se mantiene funcionando, mientras que se producen nuevas versiones, hasta se genera la documentación final pues el cliente no presenta más historias de usuario para incluir en el sistema (Beck, y otros, 2000).

Roles de XP

Programador: Es el encargado de confeccionar el código del sistema y encargarse de las pruebas unitarias correspondientes.

Cliente: Describe cada una de las historias de usuario y de asignar su prioridad decidiendo cuál implementar en cada iteración en correspondencia con su valor.

Encargado de pruebas: Ejecuta las pruebas ayudando al cliente. Es el encargado de las herramientas de soporte así como de difundir los resultados.

Encargado de seguimiento: Verifica el grado de acierto entre las estimaciones y el tiempo real dedicado.

Entrenador: Provee las guías y es el encargado de que las prácticas se apliquen correctamente.

Gestor: Es el vínculo entre clientes y programadores, su labor principal es la coordinación.

2.3 Modelo Conceptual

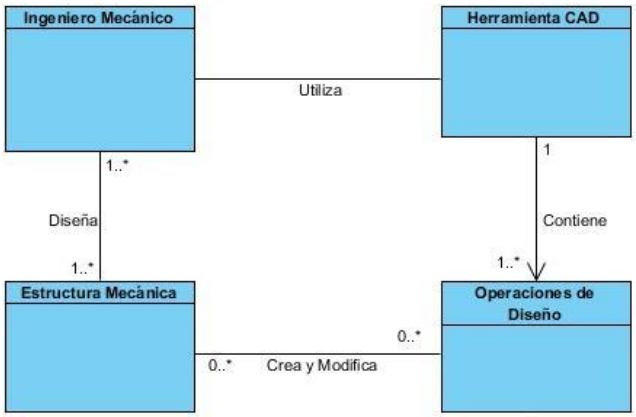

Figura 9: Modelo Conceptual.

El proceso de utilización de un sistema CAD es intuitivo. Un ingeniero mecánico (u otra persona capacitada para el trabajo), utiliza herramientas CAD para realizar piezas y estructuras mecánicas que necesita, ya sea porque no se consiguen en el mercado, porque posee la tecnología para fabricarlas u otro motivo. Para realizarlo estas herramientas tienen un grupo de funcionalidades que le permiten crear estas piezas y posteriormente aplicándole transformaciones y otras operaciones modificarlas hasta obtener el resultado esperado. AsiXMec 1.0, como toda herramienta CAD, cumple con esta filosofía.

2.4 Exploración

Esta fase se caracteriza por una completa retroalimentación con el cliente, pues es el encargado de definir la prioridad de cada uno de los requisitos funcionales del sistema, mediante las historias de usuario. El tiempo empleado para la culminación de este período depende directamente de la familiaridad que tenga el equipo de desarrollo con las tecnologías y herramientas a utilizar.

2.4.1 Requisitos Funcionales

Luego de desarrollado el modelo conceptual se analiza qué debe hacer el sistema para darle cumplimiento a los objetivos planteados. Para ello se enumeran a través de requerimientos funcionales que son capacidades o condiciones que el sistema debe cumplir. De acuerdo con los objetivos planteados, los requerimientos funcionales del sistema propuesto son:

RF1: Capturar la cara sobre la que se trabajará.

RF2: Detectar la curva sobre la cual se hará la extrusión.

RF3: Realizar extrusión a lo largo de curvas abiertas.

2.4.2 Requisitos no Funcionales

Determinar los requisitos no funcionales de un sistema implica realizar la correcta identificación de las características generales que la aplicación debe tener, con el fin de garantizar la usabilidad y aceptación por parte del usuario. Cada una de estas propiedades o cualidades pueden abarcar diferentes aspectos relacionados con el *hardware* y *software* necesarios, o con el diseño, soporte y usabilidad. Siempre con la intención de asegurar la calidad del producto final. En relación con este trabajo de igual manera se determinan los requisitos no funcionales indispensables para la implementación del componente visual propuesto.

Reusabilidad:

- El sistema se desarrollará con una arquitectura modular, con el objetivo de lograr la reutilización en la futura implementación de cualquier aplicación CAD.

Apariencia o interfaz externa:

- El diseño visual del módulo está definido de forma similar a la herramienta Autodesk Inventor.
- Optimizado para una resolución de 1024x768.

***Software*:**

- Sistema operativo: Linux, preferentemente Ubuntu 14.04 con arquitectura de 64 bits.
- Dependencias de los paquetes ftgl-dev y libxmu-dev.

***Hardware*:**

- Las estaciones de trabajo donde se utilice la herramienta deben poseer al menos como capacidad recomendada 1 GB de memoria RAM.

Usabilidad:

- Para hacer uso del módulo dentro del sistema es necesario poseer

conocimientos elementales de computación y haber trabajado previamente en alguna herramienta de diseño como 3D Max o Autodesk Inventor.

- Distribución y etiquetado de opciones bien definidos dentro del modelador para facilitar la interacción al usuario y hacerlo intuitivo.

Soporte:

- Sistema bajo plataforma de *software* libre.
- El usuario contará con un manual de usuario que le servirá para orientarse en la función que va a realizar sobre el sistema u otra tarea en general.

2.4.3 Historias de Usuario

Las historias de usuario son una técnica utilizada para describir cada uno de los requisitos funcionales y no funcionales, en ella el cliente especifica las características de la aplicación, es decir es una representación de las necesidades del sistema desde el punto de vista del cliente. Las historias de usuario son un recurso muy flexible pues cada una se caracteriza por ser muy comprensible y delimitada, de modo que el programador pueda desarrollarla en pocas semanas. Cada historia de usuario puede durar de una a tres semanas, al mismo tiempo pueden ser descompuestas en varias tareas de programación. En esta metodología no hay que preocuparse si no se identifican todas las historias de usuario necesarias, pues en cada iteración, se registraran los cambios y según estos cambios se planifica la siguiente iteración. Las historias de usuario serán usadas posteriormente tanto en la implementación como en las pruebas del producto, para confirmar si el programa cumple con lo descrito en cada una de ellas (Escribano, 2002).

Historia de Usuario	
Número: 1	**Nombre:** Crear la interfaz de usuario.
Usuario: Desarrollador	**Puntos estimados:** 2
Riesgo en desarrollo: Alto	**Prioridad en el negocio:** Alta
Descripción: Se crean las interfaces y se integran a AsiXMec 1.0.	

Tabla 1: HU Crear la interfaz de usuario correspondiente a la nueva funcionalidad.

Historia de Usuario	
Número: 2	**Nombre:** Capturar la cara.
Usuario: Desarrollador	**Puntos estimados:** 3
Riesgo en desarrollo: Alto	**Prioridad en el negocio:** Alta
Descripción: El sistema utiliza las clases ya existentes para detectar la cara con que se trabajará.	

Tabla 2: HU Capturar la entidad activa sobre la que se trabajará.

Historia de Usuario	
Número: 3	**Nombre:** Detectar la curva.
Usuario: Desarrollador	**Puntos estimados:** 3
Riesgo en desarrollo: Alto	**Prioridad en el negocio:** Alta
Descripción: El usuario crea un nuevo plano en otro *sketcher* y en este se crea la curva. El sistema utiliza las clases de OpenCASCADE y Qt para detectar la curva sobre la que se hará la extrusión.	

Tabla 3: HU Detectar la curva sobre la cual se hará la extrusión.

Historia de Usuario	
Número: 4	**Nombre:** Realizar la extrusión.
Usuario: Desarrollador	**Puntos estimados:** 3
Riesgo en desarrollo: Alto	**Prioridad en el negocio:** Alta
Descripción: El sistema utiliza los objetos detectados para hacer la extrusión, generándose una nueva operación.	

Tabla 4: HU Realizar extrusión a lo largo de curvas abiertas.

2.5 Planificación

En esta fase el cliente establece la prioridad de cada historia de usuario y se realiza una estimación del esfuerzo necesario para su futuro desarrollo. Se diseña un cronograma en conjunto con el cliente, programando cada una de las entregas. Los desarrolladores establecen la estimación para la implementación de las historias de usuario basándose en el tiempo o el alcance.

2.5.1 Estimación del Esfuerzo

En este aspecto de la planificación el equipo de desarrollo realiza una estimación del esfuerzo de cada una de las historias de usuario, de modo que se puedan valorar las prioridades conjuntamente con el cliente. Es importante tener en cuenta que para esta fase se hace necesario realizar un análisis profundo de los requerimientos con el fin de determinar correctamente el tiempo de desarrollo.

Número	Historia de Usuario	Estimación (Semanas)
1	Crear la interfaz de usuario.	3
2	Capturar la cara.	3
3	Detectar la curva.	3
4	Realizar extrusión a lo largo de la curva abierta.	3

Tabla 5: Estimación del esfuerzo.

2.5.2 Plan de Iteraciones

Luego de diseñar cada una de las historias de usuario se realizarán varias iteraciones antes de entregar el producto. El cliente decide qué Historias de Usuario se implementarán en cada iteración, de modo que en la última iteración el sistema estará listo para entrar en la fase de producción. A continuación se describen las 2 iteraciones correspondientes a la aplicación:

Iteración 1

En esta primera iteración se procederá a implementar las Historias de Usuario 1 y 2 pues se consideran de vital importancia para el desarrollo del componente. Primeramente se agregarán a AsiXMec 1.0 las clases necesarias para el funcionamiento del módulo, incluyendo las interfaces correspondientes. También se integrarán el *framework* Qt y la biblioteca OpenCascade. Posteriormente se implementará la segunda Historia de Usuario en la que se detecta uno de los componentes de la extrusión: la forma o *shape*.

Iteración 2

En una segunda iteración se implementarán las Historias de Usuario 3 y 4 en las

que se detecta el otro componente de la extrusión: el *path* o recorrido y se utilizan los objetos detectados para realizar el solevado, generándose la nueva operación. Para el desarrollo de esta funcionalidad se hace necesario el estudio del *framework* Qt.

2.5.3 Plan de duración de iteraciones

Como parte de la técnica establecida por la metodología de desarrollo XP se crea un plan de duración de las iteraciones que se llevarán a cabo durante el proceso de desarrollo. En dicho plan se mostrará duración de cada iteración en el orden en que serán implementadas.

Iteración	HU en el orden a implementar	Duración
1	Crear la interfaz de usuario correspondiente.	3
	Capturar la cara sobre la que se trabajará.	3
2	Detectar la curva.	3
	Realizar extrusión a lo largo de curvas abiertas.	3

Tabla 6: Plan de duración de las iteraciones.

2.6 Propuesta de Solución

La metodología seleccionada para el desarrollo de este trabajo se basa en retroalimentación continua entre el cliente y el equipo de desarrollo, por lo que el diseño no requiere de la utilización de extensos diagramas de modelado. En su lugar se establecen las tarjetas CRC (Contenido, Responsabilidad y Colaboración), las cuales son las encargadas de establecer la relación entre los objetos, usuarios y cada una de las clases que componen el sistema, aunque se

puede considerar el uso de algún diagrama siempre que contribuya a la integración del equipo de desarrollo.

2.6.1 Tarjetas CRC

El principal objetivo que presentan las tarjetas CRC, es permitir que todo el equipo participe en el diseño de la aplicación, además de que permite el trabajo con una metodología basada en objetos. Las tarjetas CRC están constituidas por 3 secciones en las que se mostrarán el nombre de la clase, las responsabilidades de la clase, las que constituyen las principales funcionalidades que debe cumplir. En una tercera sección se representan sus colaboraciones, que son las otras clases con las que se relaciona para cumplir sus responsabilidades. A continuación se muestran cada una de las tarjetas correspondientes a este trabajo:

Tarjeta CRC	
Clase: LoftState	
Responsabilidades: ExportState() start() rollback() exportFilter()	**Colaboraciones**: LoftDialog LoftManager SweepManagerState

Tabla 7: Tarjeta CRC Clase LoftState

Tarjeta CRC	
Clase: LoftManager	
Responsabilidades:	**Colaboraciones:**
LoftManager()	LoftState
displayPrevisualization()	LoftCommand
getBuild()	SweepManager
getCommand()	
validateBeforeExecution()	

Tabla 8: Tarjeta CRC Clase LoftManager

Tarjeta CRC	
Clase: LoftCommand	
Responsabilidades:	**Colaboraciones:**
LoftCommand()	LoftDriver
CreateFeatureNode()	LoftManager
registerFeatureType()	SweepManager
executeDriver()	

Tabla 9: Tarjeta CRC Clase LoftCommand

Tarjeta CRC	
Clase: LoftDriver	
Responsabilidades:	**Colaboraciones:**
LoftDriver()	LoftCommand
getId()	
Execute()	
getDriverError()	

Tabla 10: Tarjeta CRC Clase LoftDriver

Tarjeta CRC	
Clase: LoftBuilder	
Responsabilidades: LoftBuilder() makePart()	**Colaboraciones:** SweepBuilder SweepManager

Tabla 11: Tarjeta CRC Clase LoftBuilder.

Tarjeta CRC	
Clase: LoftDialog	
Responsabilidades:	**Colaboraciones:** LoftState

Tabla 12: Tarjeta CRC Clase LoftDialog.

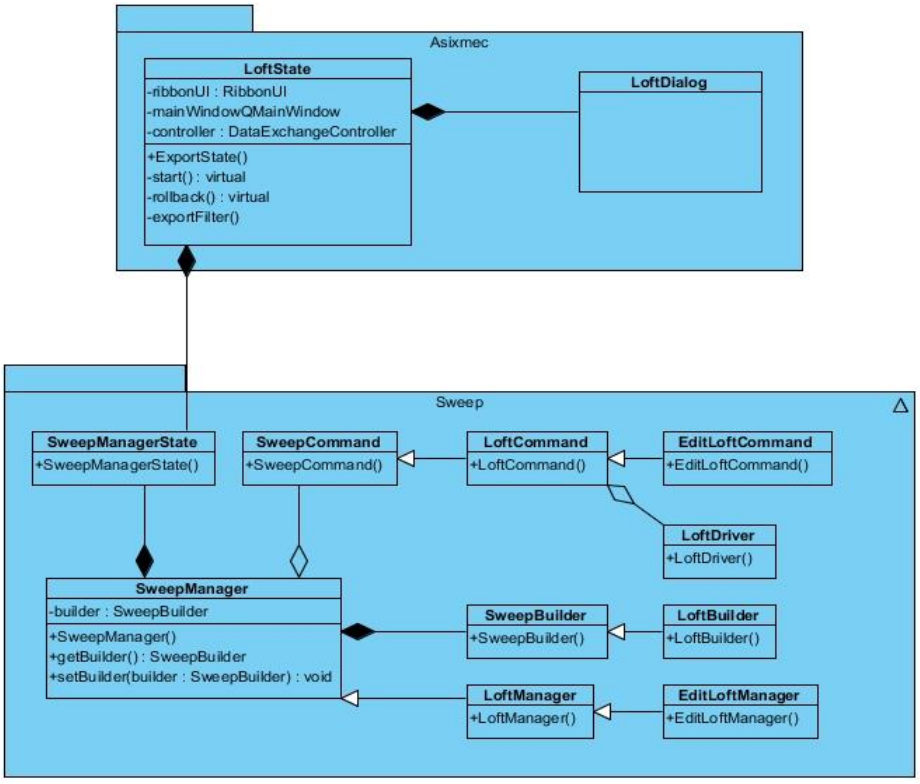

Figura 10: Diagrama de clases descrito por las tarjetas CRC.

2.6.3 Arquitectura del sistema

La arquitectura de *software* es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos, los principios que orientan su diseño y su evolución (Reynoso, 2004).

En pocas palabras, la arquitectura de *software* es una vista del sistema que se encarga de describir aspectos globales del sistema, además de incluir y reflejar la forma en que los componentes interactúan y colaboran entre sí. La arquitectura de *software* forma parte del diseño del sistema por lo que está afectado no solo por la estructura y el comportamiento de la aplicación sino que también depende de la funcionalidad el rendimiento y la flexibilidad de comprensión, entre otros aspectos.

Arquitectura modular

En programación un módulo es una porción de un programa de computadora. De las varias tareas que debe realizar un programa para cumplir con su función u objetivos, un módulo realizará, comúnmente, una de dichas tareas. De forma general, un módulo recibe como entrada la salida que haya proporcionado otro módulo o los datos de entrada al sistema si se trata del módulo principal de éste. Debe proporcionar una salida que, a su vez, podrá ser utilizada como entrada de otro módulo o bien contribuirá directamente a la salida final del sistema.

Los módulos suelen estar organizados jerárquicamente en niveles, de forma que hay un módulo principal que realiza las llamadas oportunas a los módulos de nivel inferior. Cuando un módulo es convocado, recibe como entrada los datos proporcionados por otro módulo, el que ha hecho la llamada; luego realiza su tarea. A su vez este módulo convocado puede llamar a otro u otros módulos de nivel inferior si fuera necesario; cuando ellos finalizan sus tareas, devuelven la salida pertinente al módulo que lo llamó. Cada uno de los módulos de un

programa debería cumplir las siguientes características:

- Tamaño relativamente pequeño: Esto facilita aislar el impacto que pueda tener la realización de un cambio en el programa, bien para corregir un error, o bien por rediseño del algoritmo correspondiente.
- Independencia modular: Cuanto más independientes son los módulos entre sí más fácil y flexiblemente se trabajará con ellos, esto implica que para desarrollar un módulo no es necesario conocer detalles internos de otros módulos. Como consecuencia de la independencia modular un módulo cumplirá características de caja negra, es decir abstracción y aislamiento de los detalles mediante encapsulamiento.

La independencia de la arquitectura modular mejora el rendimiento, pudiendo realizarse programación en equipo y desarrollar módulos paralelamente. También contribuye a la reutilización de software.

2.6.4 Patrones de diseño

Los patrones de diseño nacen producto a la experiencia adquirida durante el desarrollo de sistemas informáticos, por lo que constituyen una respuesta a las problemáticas comunes encontradas en la implementación de un *software*. Cada uno de estos patrones de alguna manera ha contribuido a evolucionar el diseño orientado a objetos por lo que el utilizarlos constituye una buena práctica de programación (Gracia, 2005).

Los desarrolladores orientados a objetos con experiencia acumulan un repertorio tanto de principios generales como de soluciones basadas en aplicar ciertos estilos que les guían en la creación de *software*. Estos principios y estilos, si se codifican con un formato estructurado que describa el problema y la solución y se les da un nombre, podrían llamarse patrones. Un patrón es un par

problema/solución con nombre que se puede aplicar en nuevos contextos, con consejos acerca de cómo aplicarlo en nuevas situaciones y discusiones sobre sus compromisos (Larman, 1999).

Objetivos de los patrones de diseño:

- Facilitar el aprendizaje de las nuevas generaciones de diseñadores a partir de un conocimiento ya existente.
- Generar estándar a la hora de crear un diseño de *software*.
- Proporcionar un elemento reusable en el diseño de sistemas de *software*.

Patrón Bajo Acoplamiento

El patrón Bajo Acoplamiento propone una escasa dependencia entre las clases, por lo que el acoplamiento es una medida de la fuerza con que una clase está conectada a otras clases. Un sistema sustentado por un alto acoplamiento en general constituye un síntoma de un mal diseño pues puede presentar problemas al reutilizar el código además de que un cambio en alguna de las clases supone generar un cambio global que dificulta el entendimiento cuando están aisladas. Por tanto resulta una mejor alternativa el asignar las responsabilidades tratando de mantener un grado de acoplamiento bajo (Visconti, 2011).

Beneficios del Bajo Acoplamiento:

- Las clases no se afectan por cambios en otros componentes.
- Al ganar en independencia, facilita el entendimiento del sistema.
- Facilita la reutilización (Larman, 1999).

La herramienta que da origen a esta investigación establece a este patrón de diseño, como una de las guías fundamentales para estructurar su sistema, pues establece una dependencia mínima para cada una de las clases que lo conforman. Este patrón se pone de manifiesto en todas las clases del módulo

creado, las cuales están poco relacionadas entre sí.

Patrón Alta Cohesión

Este patrón de diseño resuelve el problema de mantener la complejidad dentro de los límites manejables, asumiendo como principal objetivo el que cada elemento debe realizar una labor única dentro del sistema. El establecer una alta cohesión asegura que una clase no se sature con tareas, por lo que reparte cada una de las responsabilidades específicas en diferentes entidades. Por tanto se puede comprender como cohesión a la medida de la fuerza con que se relacionan los objetos o de la cantidad de trabajo que realizan (Visconti, 2011).

Beneficios de una alta cohesión.

- Se incrementa la claridad y facilita la comprensión del diseño.
- Se simplifican el mantenimiento y las mejoras.
- Se soporta a menudo bajo acoplamiento.
- El grano fino de funcionalidad altamente relacionada incrementa la reutilización porque una clase cohesiva se puede utilizar para un propósito muy específico (Larman, 1999).

Dentro del proceso de desarrollo de este trabajo se hace imprescindible establecer una alta cohesión, asegurando que cada clase tenga una responsabilidad específica, contribuyendo a la comprensión y mantenimiento del sistema propuesto. El patrón Alta Cohesión se evidencia cuando las distintas funcionalidades de las clases diseñadas colaboran para producir algún comportamiento bien definido. El proceso seguido desde la selección por parte del usuario de la cara a solevar, hasta la aparición del sólido generado en el visor 3D, funciona de manera armónica mediante la conjunción acoplada de las responsabilidades asignadas a cada clase.

Patrón *State*

Este patrón se utiliza cuando el comportamiento de un objeto cambia dependiendo del estado en el que se encuentre el mismo. En ocasiones un objeto tiene diferentes comportamientos según el estado en que se encuentra. Esto puede resultar complicado a la hora de manejar el cambio de comportamientos y los estados de dicho objeto. El patrón *State* soluciona esta complicación creando un objeto por cada estado posible del objeto que lo llama. En lugar de definir en el objeto de contexto las operaciones que dependen del estado, deléguelas en su objeto del estado actual. El patrón Estado es un mecanismo para implementar un modelo de transición de estados en *software* (Larman, 1999).

El patrón State se pone de manifiesto en la clase LoftState. Su utilización garantiza que se realicen de manera correcta cada una de las operaciones que se realizan para obtener un solevado.

Conclusiones Parciales

En este capítulo se analizó en detalle la propuesta de solución que se desea implementar, además de exponer el análisis y diseño de la investigación, incluido en las dos primeras fases de la metodología de desarrollo XP. Primeramente en la fase de exploración se diseñaron cada una de las historias de usuario, se logró estimar el esfuerzo necesario para la implementación cada historia de usuario. Del mismo modo durante la planificación del sistema, se documentaron cada una de las 4 iteraciones, con el fin estimar su prioridad y de pronosticar un plan de entrega coherente con la propuesta presentada. Mediante las tarjetas CRC se conformó el diseño de la aplicación y se propuso una arquitectura por componentes.

Capítulo 3 Implementación y Pruebas.

Introducción

En el presente capítulo se exponen cada uno los componentes que integran el proceso de implementación y prueba de la aplicación en cuestión. Según las pautas que presenta la metodología de desarrollo XP, el proceso de implementación se caracteriza por ser iterativo, de modo que se describen cada una de las iteraciones comprobando que se cumplen cada uno de los objetivos planteados en el plan de iteraciones. Finalmente se procede a realizar conjuntamente con el cliente y el equipo de desarrollo, las pruebas correspondientes a cada Historia de Usuario, verificando que el sistema cumple todas las funcionalidades.

3.1 Implementación

3.1.1 Propuesta de solución

En el presente epígrafe se explica el funcionamiento de cada una de las clases del módulo.

Clase LoftManager:

La clase es una especialización de SweepManager. Es la encargada de, con los datos que le llegan desde la interfaz a través de clases mediadoras (*actions*), instanciar la entidad a solevar y todos los demás atributos necesarios para llevar a cabo la operación. SweepManager cuenta con un mapa de estados de tipo *SweepManagerState* que es el encargado de hacerle saber a la aplicación en cada momento qué hacer, de acuerdo al estado en que se encuentre el usuario.

Clase SweepManagerState:

En esta clase se pone en práctica el patrón *State*, del que se profundizara más adelante en este capítulo. Posee un método virtual puro *onClick*() donde se definen las operaciones específicas que se han de hacer con los eventos del mouse de acuerdo al estado en que se encuentre la aplicación en tiempo de ejecución. Este estado va cambiando en la clase *SweepManager* en dependencia de la acción que esté ejecutando el usuario y lo que se espere en cada momento. Tras la llamada al método *onClick*() para cada estado específico, se ejecutarán las acciones necesarias para seleccionar:

- La cara o *shape* del solevado (clase hija SelectingProfileState).

- El recorrido o *path* (clase hija SelectingAxisState).

Clases LoftCommand:

Es la clase encargada de, una vez recopilados los atributos necesarios para ejecutar la operación (caras y ejes), en la clase LoftManager, encapsularlos y hacer la llamada al driver que efectúa la operación que genera y almacenan en el árbol de OCAF (estructura de datos establecida por la biblioteca OpenCascade) los sólidos creados luego de ejecutada la operación.

Clase LoftBuilder:

Esta clase hereda de la clase SweepBuilder y es utilizada por la clase SweepManager. Es en ella donde se trabaja directamente con las clases brindadas por OpenCASCADE que aportan a la solución. Su principal funcionalidad es el método *MakeOperation()* donde se obtienen el sólido a generar. Cuenta además con un método virtual puro *MakePart()*, que se redefine en las clases hijas y es donde se hace uso de las clases de OpenCASCADE

para generar el sólido.

3.2 Desarrollo de las Iteraciones

Durante la fase de Planificación se detallaron las historias de usuarios correspondientes a cada una de las iteraciones a desarrollar, las cuales tienen como objetivo principal atender las necesidades requeridas por el cliente. En el transcurso de las iteraciones se lleva a cabo una revisión del plan de iteraciones, modificándose en caso de ser preciso. Como parte de este plan, se descomponen las historias de usuarios en tareas de programación o ingeniería, asignando a un equipo de desarrollo (o una persona), responsable de su implementación, aplicando la práctica de la programación en parejas. Estas tareas no tienen que necesariamente ser entendidas por el cliente, pueden ser escritas en lenguaje técnico y son para el uso estricto de los programadores.

Teniendo en cuenta la planificación realizada con anterioridad, se llevó a cabo el desarrollo del sistema en cuatro iteraciones, obteniéndose como finalidad un producto con todas las restricciones y características deseadas por el cliente. A continuación se detallan cada una de las iteraciones.

3.2.1 Iteración 1

Historia de Usuario	Tiempo de implementación	
	Estimado	Real
Crear la interfaz de usuario correspondiente, ver Figura 11.	3	3
Capturar la cara, ver Figura 12.	3	3

Tabla 13: Planificación de la iteración 1.

Figura 11: Interfaz de usuario de la operación Solevado.

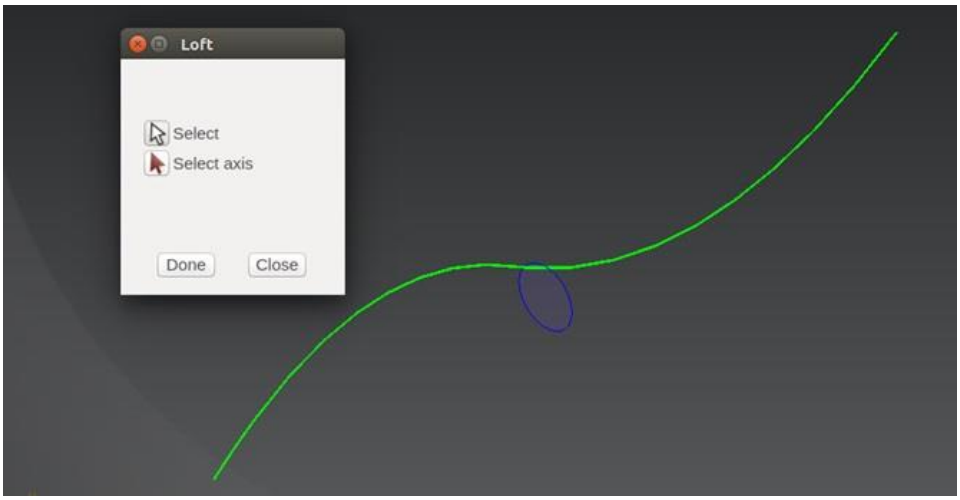

Figura 12: Selección de la cara.

Historia de Usuario	Tareas de Programación
Crear la interfaz de usuario correspondiente.	Crear la interfaz de usuario.
	Insertar el icono de la funcionalidad en el *ribbon* de AsiXMec 1.0.
	Crear las clases implicadas en el módulo.
Capturar la cara.	Implementar las clases LoftManager, LoftDriver y LoftCommand.
	Actualizar las clases SweepManagerState y SweepManager.

Tabla 14: Desagregación de tareas en la iteración 1.

Tarea de Programación	
No. de tarea: 1	**No. de HU:** 1
Nombre de la tarea: Crear la interfaz de usuario.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1
Fecha inicio: 1-2-2016	**Fecha Fin:** 5-2-2016
Programador Responsable: Adiel Pacheco	
Descripción: Se crea la interfaz y se integra a AsiXMec.	

Tabla 15: Tarea 1, Crear la interfaz de usuario.

Tarea de Programación	
No. de tarea: 2	**No. de HU:** 1
Nombre de la tarea: Insertar el icono de la funcionalidad en el *ribbon* de AsiXMec 1.0.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1
Fecha inicio: 8-2-2016	**Fecha Fin:** 12-2-2016
Programador Responsable: Adiel Pacheco	
Descripción: Se inserta el ícono del solevado en el *ribbon* de AsiXMec 1.0.	

Tabla 16: Tarea 2, Insertar el icono de la funcionalidad en el ribbon de AsiXMec 1.0.

Tarea de Programación	
No. de tarea: 3	**No. de HU:** 1
Nombre de la tarea: Crear las clases implicadas en el módulo.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1
Fecha inicio: 15-2-2016	**Fecha Fin:** 19-2-2016
Programador Responsable: Adiel Pacheco	
Descripción: Se agregan a AsiXMec 1.0 las clases que serán utilizadas.	

Tabla 17: Tarea 3, Crear las clases implicadas en el módulo.

Tarea de Programación	
No. de tarea: 4	**No. de HU:** 2
Nombre de la tarea: Implementar las clases LoftManager, LoftDriver y LoftCommand.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 2
Fecha inicio: 7-3-2016	**Fecha Fin:** 18-3-2016
Programador Responsable: Adiel Pacheco	
Descripción: Se implementan en las clases LoftManager, LoftDriver y LoftCommand los métodos necesarios para detectar la cara.	

Tabla 18: Tarea 4, Implementar las clases LoftManager, LoftDriver y LoftCommand.

Tarea de Programación	
No. de tarea: 5	**No. de HU:** 2
Nombre de la tarea: Actualizar las clases SweepManagerState y SweepManager.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1
Fecha inicio: 21-3-2016	**Fecha Fin:** 25-3-2016
Programador Responsable: Adiel Pacheco	
Descripción: Se actualizan las clases SweepManagerState y SweepManager con los métodos necesarios para detectar la cara.	

Tabla 19: Tarea 5, Actualizar las clases SweepManagerState y SweepManager.

3.2.2 Iteración 2

Historia de Usuario	Tiempo de implementación	
	Estimado	Real
Detectar la curva, ver Figura 13.	3	4
Realizar extrusión a lo largo de curvas abiertas, ver Figura 14.	3	4

Tabla 20: Planificación de la iteración 2.

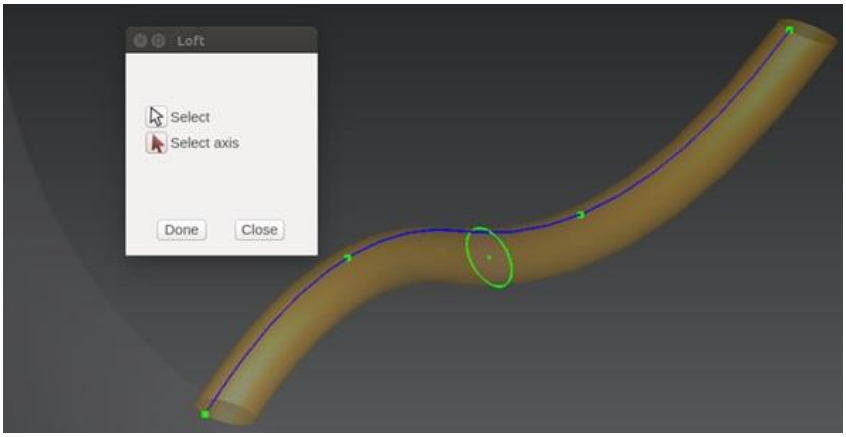

Figura 13: Selección de la curva.

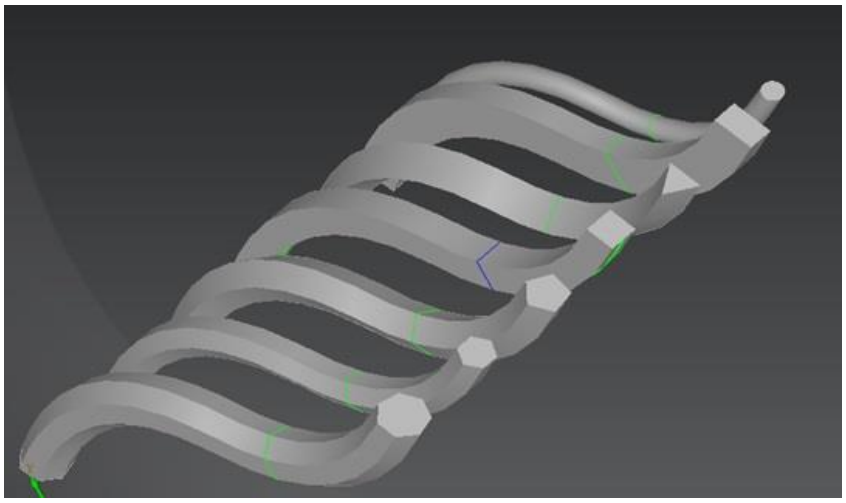

Figura 14: Extrusiones a lo largo de una curva.

Historia de Usuario	Tareas de Programación
Detectar la curva.	Actualizar las clases LoftManager, LoftDriver y LoftCommand.
	Actualizar las clases SweepManagerState y SweepManager.
Realizar extrusión a lo largo de curvas abiertas.	Implementar la clase LoftBuilder.
	Registrar el ítem en el Navegador de objetos.
	Establecer las traducciones a español e inglés.
	Vincular la operación Solevado a AsiXMec 1.0.

Tabla 21: Desagregación de tareas en la iteración 2.

Tarea de Programación	
No. de tarea: 6	**No. de HU:** 3
Nombre de la tarea: Actualizar las clases LoftManager, LoftDriver y LoftCommand.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 2
Fecha inicio: 4-4-2016	**Fecha Fin:** 15-4-2016
Programador Responsable: Adiel Pacheco	
Descripción: Se le agregan a las clases LoftManager, LoftDriver y LoftCommand los métodos necesarios para detectar la curva.	

Tabla 22: Tarea 6, Actualizar las clases LoftManager, LoftDriver y LoftCommand.

Tarea de Programación	
No. de tarea: 7	**No. de HU:** 3
Nombre de la tarea: Actualizar SweepManagerState y SweepManager.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1
Fecha inicio: 18-4-2016	**Fecha Fin:** 29-4-2016
Programador Responsable: Adiel Pacheco	
Descripción: Se le agregan a las clases SweepManagerState y SweepManager los métodos necesarios para detectar la curva.	

Tabla 23: Tarea 7, Actualizar las clases SweepManagerState y SweepManager.

Tarea de Programación	
No. de tarea: 8	**No. de HU:** 4
Nombre de la tarea: Implementar la clase LoftBuilder.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1
Fecha inicio: 2-5-2016	**Fecha Fin:** 6-5-2016
Programador Responsable: Adiel Pacheco	
Descripción: Se implementa la clase LoftBuilder, realizándose la operación.	

Tabla 24: Tarea 8, Implementar la clase LoftBuilder.

Tarea de Programación	
No. de tarea: 9	**No. de HU:** 4
Nombre de la tarea: Registrar el ítem en el Navegador de objetos.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1
Fecha inicio: 9-5-2016	**Fecha Fin:** 13-5-2016
Programador Responsable: Adiel Pacheco	
Descripción: Se crea el ítem de la operación. Se visualiza en el navegador.	

Tabla 25: Tarea 9, Registrar el ítem en el Navegador de objetos.

Tarea de Programación	
No. de tarea: 10	**No. de HU:** 4
Nombre de la tarea: Establecer las traducciones a español e inglés.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1
Fecha inicio: 16-5-2016	**Fecha Fin:** 20-5-2016
Programador Responsable: Adiel Pacheco	
Descripción: Se realizan las traducciones de los *tokens* a ambos idiomas.	

Tabla 26: Tarea 10, Establecer las traducciones a español e inglés.

Tarea de Programación	
No. de tarea: 11	**No. de HU:** 4
Nombre de la tarea: Vincular la operación Solevado a AsiXMec 1.0.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1
Fecha inicio: 23-5-2016	**Fecha Fin:** 27-5-2016
Programador Responsable: Adiel Pacheco	
Descripción: Se vincula el Solevado a AsiXMec 1.0, pudiendo realizársele operaciones como *Fillet* y *Chamfer,* ver Figura 15.	

Tabla 27: Tarea 11, Vincular la operación Solevado a AsiXMec 1.0.

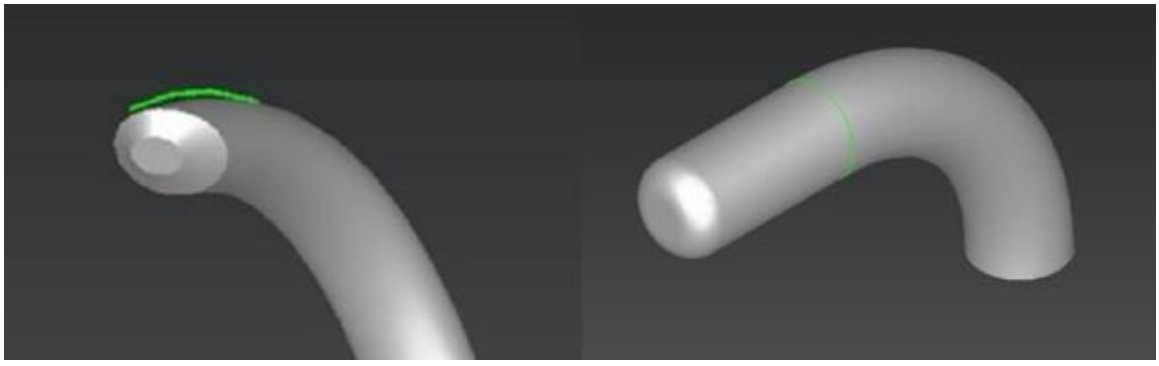

Figura 15: Operaciones de *Chamfer* y *Fillet sobre un objeto Solevado*.

En la utilización de la biblioteca OpenCASCADE se detectó que la API BRepOffsetAPI_MakePipe no funciona correctamente cuando la curva tiene secciones cerradas, ver Figura 16 o cuando esta tiende a cerrarse, ver Figura 17. Cuando esto ocurre se recomienda cambiar la curva y si esto no es posible usar una cara más pequeña.

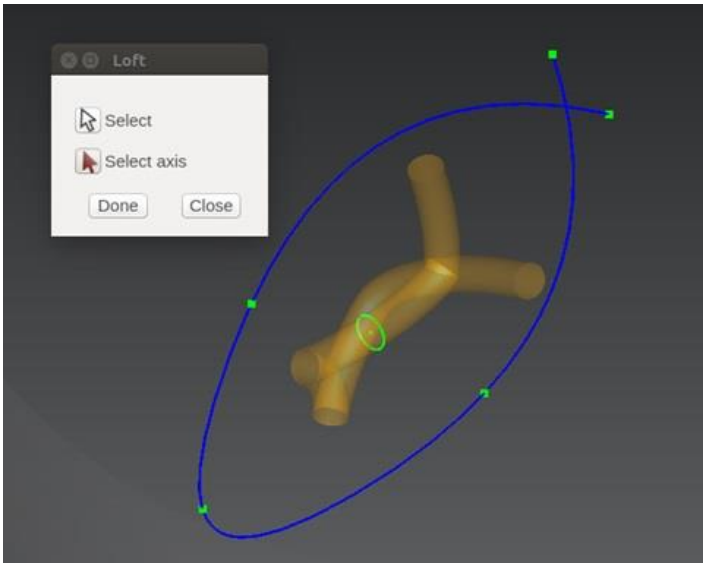

Figura 16: Funcionamiento incorrecto de la API al usar curvas con secciones cerradas.

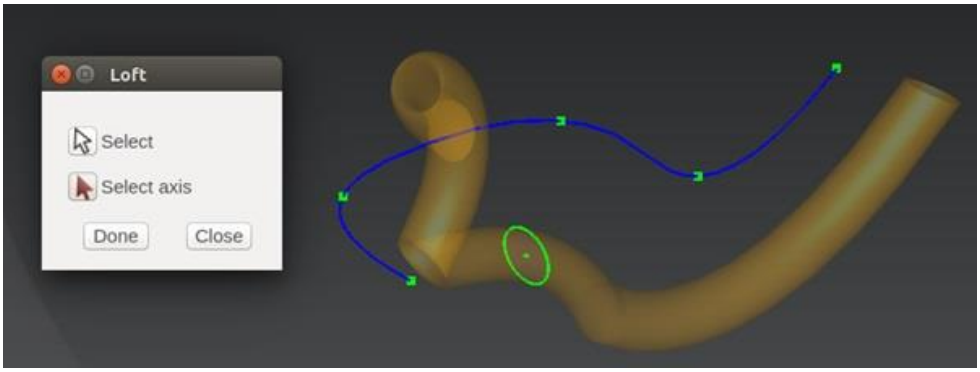

Figura 17: Funcionamiento incorrecto de la API con curvas que tienden a cerrarse.

3.3 Estándar de codificación

El código de AsiXMec 1.0 se rige por el estándar de codificación del proyecto DISEM establecido por el documento “Estándares de codificación para el lenguaje C++ utilizado en el centro Vertex” (Lombera, 2012);

Líneas: Se empleara una sola instrucción por línea de código para mejorar la legibilidad, comprensión y limpieza en el trabajo, ver Figura 18.

```
25  void LoftManager::resetState()
26  {
27      axisSelected = false;
28      profileSelected = false;
29      axisEdge.Nullify();
30      SweepManager::resetState();
31  }
```

Figura 18: Ejemplo de estilo de código en cuanto a líneas.

Bloques: Se utilizan llaves para delimitar todos los bloques de sentencias de control y bucles. Las llaves están en la misma columna y el código que contienen lleva, como mínimo, una tabulación adiciona, ver Figura 19.

```
34  TopoDS_Shape LoftBuilder::makePart(const TopoDS_Shape &profile, const gp_Ax1& dir)
35  {
36      bool v = axis.isValid();
37
38      TopoDS_Edge myEdge = axis.edge();
39      TopoDS_Wire edgeWire = BRepBuilderAPI_MakeWire(myEdge);
40
41      BRepBuilderAPI_MakeShape *sweepBuilderAPI = new BRepOffsetAPI_MakePipe(edgeWire,profile);
42
43      if(sweepBuilderAPI->IsDone())
44      {
45          TopoDS_Shape pipeResult = sweepBuilderAPI->Shape();
46          return pipeResult;
47      }
48      return TopoDS_Shape();
49  }
```

Figura 19: Ejemplo de estilo de código en cuanto a bloques.

Espacios en blanco: Se hace uso abundante de líneas y espacios en blanco para hacer más claro y comprensible el código. Las funciones y los bloques de código son aislados unos de otros, usando varias líneas en blanco para facilitar su lectura y poner de manifiesto su carácter de unidad de programación. De esta forma se obtiene un código de fácil lectura en el que es sencillo localizar funciones y bloques de código, ver Figura 20.

```
SweepManager::SweepManager()
{
   stateMap.insert(pair<State, SweepManagerState*> (SELECTING_PROFILE, new SelectingProfileState(this)));

   stateMap.insert(pair<State, SweepManagerState*> (SELECTING_TO, new SelectingToState(this)));

   stateMap.insert(pair<State, SweepManagerState*> (SELECTING_SOLID, new SelectingSolidState(this)));

   state = SELECTING_PROFILE;
   outputMode = SOLID;
   profileSelected = false;
   solidSelected = false;
   toSelected = false;

   aisPrevisaulization = new AIS_Shape(TopoDS_Shape());
   aisPrevisaulization->SetSelectionMode(-1);
}
```

Figura 20: Ejemplo de estilo de código en cuanto a espacios en blanco.

Comentarios: Los comentarios (pueden ser minimizados), facilitan la comprensión del código y aportan información útil sobre las sentencias, bloques, variables, funciones que afectan. Se utilizaran la menor cantidad posible, usándolos solo donde sea necesario aclarar aspectos importantes, ver Figura 21.

```
public:
    LoftDriver();
    /**
    * @brief Return the id of the class
    * @details
    * @return The id
    * @param
    */
    static const Standard_GUID getId();
```

Figura 21: Ejemplo de estilo de código en cuanto a comentarios.

3.4 Pruebas

La fase de pruebas es un aspecto de vital importancia dentro del ciclo de vida de un proyecto que encaminado por la metodología XP, puesto que permiten certificar con exactitud la calidad de un producto. Estas pruebas disminuyen considerablemente el número de errores, reduciendo el tiempo de transcurrido entre la aparición y detección de un error, fortaleciendo del mismo modo la seguridad del producto en cuestión. En XP el proceso de pruebas se divide en dos ramas, las pruebas de unitarias y las pruebas de aceptación. Las pruebas unitarias son protagonizadas por los implementadores, pues son los encargados de verificar directamente el código del producto. Por otro lado las pruebas de aceptación son ejecutadas por un equipo más numeroso, ya que el cliente es el encargado de diseñarlas; por tanto inciden directamente en la satisfacción del cliente con el *software* al final de cada iteración y al principio de la siguiente. El módulo se somete a pruebas de aceptación por ser las más adecuadas al trabajo realizado además de ser inmediatamente necesario para avanzar de una iteración a otra.

3.4.1 Pruebas de Aceptación

Las pruebas de aceptación son diseñadas a partir de cada iteración y basadas en las historia de usurarios, en ella es el cliente el encargado de comprobar que ha sido correctamente implementada. Estas pruebas son reconocidas por su similitud con las llamadas “Pruebas de caja negra”, pues como los clientes son los principales responsables de la inspección, también son los autorizados de diseñar la prioridad o el orden de resolución en caso de que falle alguna funcionalidad (Escribano 2002).

Caso de Prueba de Aceptación	
Código: UH1_PA1	**HU:** 1
Nombre: Crear la interfaz de usuario.	
Descripción: Comprobar que se generó una interfaz de usuario que permite realizar todas las funcionalidades.	
Condiciones de Ejecución: El cliente debe comprobar que la interfaz realiza su función.	
Entrada / Pasos de Ejecución: Utilizar la interfaz.	
Resultado Esperado: La interfaz es funcional.	
Evaluación de la Prueba: Satisfactoria.	

Tabla 28: Prueba de Aceptación de la HU 1

Caso de Prueba de Aceptación	
Código: UH2_PA1	**HU:** 2
Nombre: Capturar la cara.	
Descripción: El sistema utiliza las clases ya existentes para detectar la cara con que se trabaja.	
Condiciones de Ejecución: El cliente debe comprobar que se detecta la cara.	
Entrada / Pasos de Ejecución: Detectar la cara mediante la interfaz.	
Resultado Esperado: Se detecta la cara.	
Evaluación de la Prueba: Satisfactoria.	

Tabla 29: Prueba de Aceptación de la HU 2

Caso de Prueba de Aceptación	
Código: UH3_PA1	**HU:** 3
Nombre: Detectar la curva.	
Descripción: El sistema utiliza las clases de OpenCASCADE y Qt para detectar la curva sobre la que se hará la extrusión.	
Condiciones de Ejecución: El cliente debe comprobar que se detecta la curva.	
Entrada / Pasos de Ejecución: Detectar la curva mediante la interfaz.	
Resultado Esperado: Se detecta la curva.	
Evaluación de la Prueba: Satisfactoria.	

Tabla 30: Prueba de Aceptación de la HU 3

Caso de Prueba de Aceptación	
Código: UH4_PA1	**HU:** 4
Nombre: Realizar la extrusión.	
Descripción: El sistema utiliza los objetos detectados para hacer la extrusión, generándose una nueva operación.	
Condiciones de Ejecución: El cliente debe comprobar que se realiza la extrusión correctamente.	
Entrada / Pasos de Ejecución: Realizar la extrusión utilizando la interfaz correspondiente.	
Resultado Esperado: Se realiza la extrusión.	
Evaluación de la Prueba: Satisfactoria.	

Tabla 31: Prueba de Aceptación de la HU 4.

3.4.2 Resultados de las pruebas

Se realizaron 3 iteraciones de pruebas al módulo implementado que arrojaron deficiencias que fueron solucionadas. Esas deficiencias fueron útiles para perfeccionar el funcionamiento del módulo implementado, ver Figura 22. Después de pasar por el último periodo de pruebas, se obtuvo una alta satisfacción por parte del cliente.

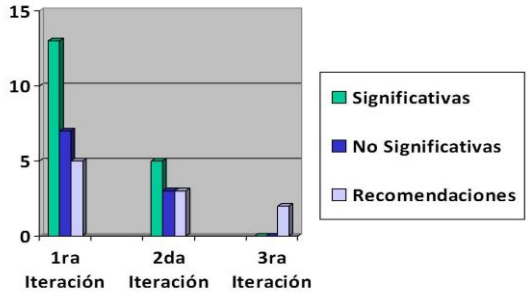

Figura 22: Iteraciones de pruebas de *software* con sus resultados.

Conclusiones Parciales

En este capítulo se representó todo el proceso relacionado con la implementación del *softw*are, describiendo cada una de las iteraciones planificadas. Se detallaron cada una de las tareas de ingeniería e implementación correspondientes a cada iteración, corroborando que las mismas se desplegaron según lo previsto por el plan correspondiente. Se ejecutaron además cada una de las pruebas propuestas por la metodología XP, con el fin de validar la calidad de la aplicación. Se aplicaron las pruebas de aceptación donde el cliente comprobó las funcionalidades descritas mediante las historias de usuarios. Al finalizar este capítulo se concluye todo el proceso de desarrollo que ocupa a esta investigación.

Conclusiones Generales

Una vez finalizado todo el proceso de desarrollo generado por esta investigación, se arribó a las siguientes conclusiones:

- La clase BRepOffsetAPI_MakePipe, utilizada para realizar el solevado, presenta deficiencias en el trabajo con curvas de ángulos cerrados.
- El módulo permite añadir la operación solevación a los elementos 2D presentes dentro del *sketcher* del modelador de la herramienta AsiXMec 1.0, originando sólidos 3D que sirvan de base para el diseño de piezas mecánicas.

Recomendaciones

- Permitir al usuario seleccionar el plano en que desea crear la curva en la interfaz de creación del solevado.
- Implementar la funcionalidad de *Shell* a lo largo de una curva.

Referencias Bibliográficas.

Abrahamsson, P, Salo, O y Ronkainen, J. 2002. *Agile software development methods. Review and analysis.* [En línea] 2002. http://www.vtt.fi/inf/pdf/publications/2002/P478.pdf. ISBN 951-38-6009-4.

Aldaya, V y Fernández, J.A. 2015. *Tutorial de modelado de la presa de Béznar.* s.l. : Universidad de Granada, 2015.

Autodesk Corporation. 2016. Buy Autodesk Software. [En línea] 2016. [Citado el: 4 20, 2016.] http://www.autodesk.com/store.

Beck, Kent y Fowler, M. 2000. *Planning Extreme Programming.* Boston: Addison-Wesley, 2000. ISBN: 0-201-71091-9.

Blanchette, Jasmin y Summerfield, Mark. 2008. *C++ GUI Programming with Qt 4.* Massachusetts: Prentice Hall, 2008. ISBN: 0-13-235416-0.

Blender Foundation. 2015. Blender Foundation. [En línea] 2015. [Citado el: 12 9, 2015.] http://www.blender.org.

CPlusPlus. 2015. CPlusPlus. [En línea] 2015. [Citado el: 12 16, 2015.] http://www.cplusplus.com.

Díaz, Sara Iris y Domínguez Cuní, Darluin. 2010. *Editor de calles 3D.* s.l.: UCI, 2010.

Escribano, G. F. 2002. *Introducción a Extreme Programming.* 2002.

Gracia, J. 2005. *Patrones de Diseño. Diseño de Software Orientado a Objetos.* 2005.

Google Co. 2016. SketchUp. [En línea] 2016. [Citado el: 2 19, 2016.] http://www.sketchup.com.

Gualoto, Freddy Mauricio. 2015. *Diseño y construcción de una trituradora y extrusora para la producción de hilo plástico empleado en impresoras 3D.* [En línea] 2015. [Citado el: 4 21, 2015.] http://dspace.ups.edu.ec/bitstream/123456789/9089/1/UPS-KT01128.pdf.

Inventor. 2016. Inventor - Software CAD 3D y de Diseño Mecánico 3D. [En línea] 2016. [Citado el: 2 12, 2016.] http://latinoamerica.autodesk.com/adsk/servlet/pc/index?id=14601337&siteID=7411870.

Larman, Craig. 1999. *UML y Patrones, Introducción al análisis y diseño orientado a objetos.* Boston: Addison Wesley, 1999. ISBN: 0-13-111155-8.

Lombera, Hassán. 2012. *Estándares de codificación para el lenguaje C++ utilizado en el Centro de Diseño y Simulación de Estructuras Mecánicas.* s.l. : UCI, 2012.

Méndez, Alejandra. 2010. *Investigación Documental Metodologías de desarrollo de software. Michoacán* : Instituto Tecnológico Superior de Apatzingán, 2010.

Mixprogramas. 2016. Mixprogramas. [En línea] 2016. [Citado el: 2 16, 2016.] www.mixprogramas.com/open-cascade-6-5-1.

Motores de Juego. 2007. Motores de Juego. [En línea] Universidad Europea de Madrid, 2007. [Citado el: 2 16, 2016.] http://www.esi.uem.es/jccortizo/motores.html.

OpenCascade. 2016. [En línea] Open Cascade SA, 2016. [Citado el: 2 16, 2016.] http://www.opencascade.org.

Piattini, M. 1996. *Análisis y diseño detallado de aplicaciones informáticas de gestión.* Madrid : Ra-Ma, 1996.

Pressman, Roger S. 2002. *Ingeniería de Software, un enfoque práctico. Quinta Edición.* México : McGraw-Hill Companies, 2002. ISBN 8848132149.

QtSoftware. 2015. [En línea] 2015. [Citado el: 12 16, 2015.] http://qt.nokia.com/products.

Reynoso, Carlos. 2004. *Introducción a la Arquitectura de Software.* Buenos Aires: Universidad de Buenos Aires, 2004.

Salmon, R y Slater, M. 1987. *Computer Graphics: Systems and Concepts.* s.l. : Addison Wesley, 1987. ISBN: 978-0201146561

Sheppard, T. 1999. *Extrusion of aluminium alloys.* Bournemouth: Bournemouth University Department of Product Design and Manufacture, 1999.

SketchUp. 2016. Getting Started in SketchUp. *SketchUp Help Center.* [En línea] 2016. [Citado el: 4 7, 2016.] help.sketchup.com/ko.

SolidWork. 2016. 3D CAD Design Software SOLIDWORK. [En línea] 2016. [Citado el: 2 12, 2016.] http://www.solidworks.com/.

Stephens, M y Rosenberg, D. 2003. *Extreme Programming Refactored: The Case Against XP.* 2003. Berkeley: Apress ISBN 978-1-4302-0810-5.

Visconti, M y Astudillo, H. 2011. *Fundamentos de Ingeniería de Software.* 2011.

VisualParadigm. 2011. Visual Paradigm. Ayuda en línea de Visual Paradigm. [En línea] 2011. [Citado el: 12 16, 2015.] http://www.visual-paradigm.com/product/vpuml.

Bibliografía consultada.

Abrahamsson, P, Salo, O y Ronkainen, J. 2002. Agile software development methods. Review and analysis. [En línea] 2002. http://www.vtt.fi/inf/pdf/publications/2002/P478.pdf. ISBN 951-38-6009-4.

Aldaya, V y Fernández, J.A. 2015. *Tutorial de modelado de la presa de Béznar.* s.l. : Universidad de Granada, 2015.

Aluminios Cordova. 2014. Aluminios Cordova. [En línea] 2014. [Citado el: 4 21, 2016.] http://www.aluminioscordoba.com/aluminios.html.

Autodesk Corporation. 2016. Buy Autodesk Software. [En línea] 2016. [Citado el: 4 20, 2016.] http://www.autodesk.com/store.

Beck, Kent y Fowler, M. 2000. *Planning Extreme Programming.* Boston : Addison-Wesley, 2000. ISBN 0-201-71091-9.

Blanchette, Jasmin y Summerfield, Mark. 2008. *C++ GUI Programming with Qt 4.* Massachusetts : Prentice Hall, 2008. ISBN 0-13-235416-0.

Blender Foundation. 2015. Blender Foundation. [En línea] 2015. [Citado el: 12 9, 2015.] http://www.blender.org..

Coad, P. 1995. *Object Models: Stategies, Patterns and Applications.* Englewood Cliffs : Prentice-Hall, 1995.

Coad, P, E, Lefebvre y J, De Luca. 1999. *Java Modeling In Color With UML: Enterprise Components and Process.* s.l.: Prentice Hall, 1999.

CPlusPlus. 2015. CPlusPlus. [En línea] 2015. [Citado el: 12 16, 2015.] http://www.cplusplus.com.

Dassault Systemes. 2016. Perspectiva general de recubrir. *Ayuda de SolidWork.* [En línea] Dassault Systemes, 2016. [Citado el: 3 28, 2016.] http://help.solidwork.com.

Díaz, Sara Iris y Domínguez Cuní, Darluin. 2010. Editor de calles 3D. s.l. : UCI,

2010.

Escribano, G. F. 2002. Introducción a Extreme Programming. 2002.

Gabardini, Juan y Campos, Lucas. 2004. *Balanceo de Metodologías Orientadas al Plan y Ágiles. Herramientas para la Selección y Adaptación.* Buenos Aires : s.n., 2004.

GenBeta. 2015. GenBeta. [En línea] 2015. [Citado el: 8 de 12 de 2015.] http://www.genbeta.com/herramientas/librecad-un-programa-sencillo-para-iniciarse-en-el-mundo-del-cad.

González, Minardo. 2011. XIV Convención y Feria Internacional Informática 2011. [En línea] 2011. http://www.informaticahabana.cu.. ISBN 978-959-7213-01-07.

Google Co. 2016. SketchUp. [En línea] 2016. [Citado el: 2 19, 2016.] http://www.sketchup.com.

Gracia, J. 2005. *Patrones de Diseño. Diseño de Software Orientado a Objetos.* 2005.

Gualoto, Freddy Mauricio. 2015. Diseño y construcción de una trituradora y extrusora para la producción de hilo plástico empleado en impresoras 3D. [En línea] 2015. [Citado el: 4 21, 2015.] http://dspace.ups.edu.ec/bitstream/123456789/9089/1/UPS-KT01128.pdf.

Inventor. 2016. Inventor - Software CAD 3D y de Diseño Mecánico 3D. [En línea] 2016. [Citado el: 2 12, 2016.] http://latinoamerica.autodesk.com/adsk/servlet/pc/index?id=14601337&siteID=7411870.

Larman, Craig. 1999. *UML y Patrones, Introducción al análisis y diseño orientado a objetos.* Boston: Addison Wesley, 1999. ISBN: 0-13-111155-8.

Larman, Craig. 2003. *Agile and Iterative Development: A Manager's Guide* Addison Wesley. ISBN: 0-13-111155-8

Letelier, Patricio y Penadés, Mª del Carmen. 2006. *Metodologías ágiles para el desarrollo de software: eXtreme Programming (XP).* 2006.

LibreCAD. 2016. LibreCAD. [En línea] 2016. [Citado el: 12 de 2 de 2016.] http://librecad.org/cms/home.html.

Lombera, Hassán. 2012. *Estándares de cosificación para el lenguaje C++ utilizado en el Centro de Diseño y Simulación de Estructuras Mecánicas.* s.l. : UCI, 2012.

Lores, José Ángel, Márquez, Marvyn Amado, Pérez, Gendor. *Módulo sweep para la herramienta AsiXMec.* Habana : s.n., 2013.

Méndez, Alejandra. 2010. Investigación Documental Metodologías de desarrollo de software. Michoacán : Instituto Tecnológico Superior de Apatzingán, 2010.

Mesa, Fernando Javier y Terrero Hernández, Roger. 2010. Gestor de bibliotecas de script en Python para el desarrollo de Paseos Virtuales sobre Blender. s.l. : UCI, 2010.

Mixprogramas. 2016. Mixprogramas. [En línea] 2016. [Citado el: 2 16, 2016.] www.mixprogramas.com/open-cascade-6-5-1.

Motores de Juego. 2007. Motores de Juego. [En línea] Universidad Europea de Madrid, 2007. [Citado el: 2 16, 2016.] http://www.esi.uem.es/jccortizo/motores.html.

OpenCascade. 2016. [En línea] Open Cascade SA, 2016. [Citado el: 2 16, 2016.] http://www.opencascade.org.

Palacio, Juan. 2007. Flexibilidad con Scrum. 2007.

Piattini, M. 1996. *Análisis y diseño detallado de aplicaciones informáticas de gestión.* Madrid : Ra-Ma, 1996.

PlataformaArquitectura. 2012. Plataforma Arquitectura. [En línea] 2012. [Citado el: 12 8, 2015.] http://www.plataformaarquitectura.cl/2012/02/04/autocad-cumple-30-anos-en-el-mercado.

Poppendieck, M y Poppendieck, T. 2003. Lean Software Development: An Agile Toolkit for Software Development Managers. Madrid : Addison Wesley, 2003.

Pressman, Roger S. 2002. *Ingeniería de Software, un enfoque práctico. Quinta Edición.* México : McGraw-Hill Companies, 2002. ISBN 8848132149.

QtSoftware. 2015. [En línea] 2015. [Citado el: 12 16, 2015.] http://qt.nokia.com/products.

Rado, Lisandra Zaylín. 2011. Análisis y Diseño de una herramienta para la creación, edición y visualización de Centros Expositivos Virtuales. UCI : s.n., 2011.

Reynoso, Carlos. 2004. *Introducción a la Arquitectura de Software.* Buenos Aires : Universidad de Buenos Aires, 2004.

Salmon, R y Slater, M. 1987. *Computer Graphics: Systems and Concepts.* s.l. : Addison Wesley, 1987.

Schwaber, K, Beedle, M y Martin, R. 2001. Agile Software Development with SCRUM. s.l. : Prentice Hall, 2001.

Sheppard, T. 1999. Extrusion of aluminium alloys. Bournemouth : Bournemouth University Department of Product Design and Manufacture, 1999.

SketchUp, 2016. Getting Started in SketchUp. *SketchUp Help Center.* [En línea] 2016. [Citado el: 4 7, 2016.] help.sketchup.com/ko.

SolidWork, 2016. 3D CAD Design Software SOLIDWORK. [En línea] 2016. [Citado el: 12 de 2 de 2016.] http://www.solidworks.com/.

Stephens, M y Rosenberg, D. 2003. Extreme Programming Refactored: The Case Against XP. 2003.

Ubuntips, 2011. Ubuntu, Software Libre y algo más... [En línea] 2011. [Citado el: 12 16, 2015.] http://www.ubuntips.com.ar/2011/04/15/blender-2-57/.

Visconti, M y Astudillo, H. 2011. *Fundamentos de Ingeniería de Software.* 2011.

VisualParadigm, 2011. Visual Paradigm. Ayuda en línea de Visual Paradigm. [En línea] 2011. [Citado el: 12 16, 2015.] http://www.visual-paradigm.com/product/vpuml.