Universidad de las Ciencias Informáticas
Vertex, Entornos Interactivos 3D, Facultad 5

Versión 2.0 del módulo *Snap* del producto AsiXMec

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

Autora: Midiala Baños Artigas

Tutores: MSc. Marvyn Amado Márquez Rodríguez

Ing. Adrián Peña Peñate

La Habana, 2015

La vida es aquello que te va sucediendo
mientras estás ocupado haciendo otros planes.
John Lennon.

Dedicatoria

Dedico este trabajo a esa personas que siempre me han amado incondicionalmente: mis padres Miriela y Juan Francisco, mi hermano Sandiel, mi abuela Marta y mis tíos Gilberto, Alberto y Reinier.

Agradecimientos

A mi familia, quien siempre me ha ayudado para lograr mis metas y sueños.

A mis eternos amigos Lazara, Juan Carlos, Ernesto, Maturell y Arisney, quienes en los momentos difíciles me demostraron el significado de la palabra "amistad".

A Ulise, por haberme dedicado algunos de sus años.

A Jose, por haber llegado en el momento oportuno.

A las amigas de mi infancia y la adolescencia Ana Lidia y Yeyli.

A mis compañeros de la universidad, por las experiencias vividas junto a ellos.

A esas personas que en algún momento su camino se cruzó con el mío, dejando huellas imborrables.

Declaración de autoría

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales sobre esta, con carácter exclusivo.

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

Midiala Baños Artigas
Autora

MSc. Marvyn Amado Márquez Rodríguez
Tutor

Ing. Adrián Peña Peñate
Tutor

Resumen

El diseño computarizado ha tomado gran auge en todas las esferas sociales. Es por eso que las grandes y pequeñas empresas dedicadas al desarrollo de software, hacen todo lo posible por llevar al mercado herramientas de Diseño Asistido por Ordenador (CAD por sus siglas en inglés) con las mejores funcionalidades, con el objetivo de satisfacer el mayor número de necesidades de los clientes. El presente trabajo está orientado al desarrollo de una segunda versión del módulo *Snap* de la herramienta de diseño computarizado *AsiXMec*, que se desarrolla en el proyecto Diseño y Simulación de Estructuras Mecánicas de la Universidad de las Ciencias Informáticas. Este *software* tiene como propósito apoyar a los ingenieros mecánicos en el diseño de prototipos virtuales de estructuras mecánicas, para su posterior ensamble en empresas manufactureras. La nueva versión surge debido a que al adicionar nuevos *snaps* o modificar los existentes, es necesario compilar toda la aplicación, debido al alto acoplamiento que existe entre el módulo *Snap* y el núcleo del sistema. La propuesta de solución se basa en el uso de *plugins* para reducir el acoplamiento en el *software AsiXMec*, obteniéndose una versión mejorada que permite a *AsiXMec* cargar dinámicamente los *snaps* una vez que se inicializa. Esto le permite al usuario seleccionar desde una interfaz visual los *snaps* que desee que se ejecuten durante la etapa de diseño.

Palabras clave: Acoplamiento, *AsiXMec*, CAD, *Plugin*, *Snap*.

Índice general

Índice de figuras

Índice de tablas

Lista de códigos fuentes

Introducción

La calidad y el tiempo de desarrollo de las aplicaciones son aspectos importantes para incrementar la competitividad de las empresas de desarrollo de *software*. Para insertar en el mercado un producto, este debe satisfacer las funcionalidades que demanda el cliente con calidad y en el menor tiempo de desarrollo posible. La precisión también es una característica deseada por las empresas que se dedican al desarrollo de herramientas *Computer Aided Design* (*CAD*), lo que se evidencia en la creación de algunas técnicas para ayudar al usuario a lograr una mayor exactitud en sus diseños, como es el caso de los *snaps*. Estos son elementos que se activan cuando el puntero se encuentra cerca de determinadas características o puntos de una entidad con la que un diseñador esté trabajando, adquiriendo así un comportamiento de atracción.

Muchos de los *software CAD* que se dedican al diseño y modelado de piezas mecánicas, edificaciones, carrocerías o productos cinematográficos como películas son privativos, lo cual dificulta a los ingenieros mecánicos, arquitectos y diseñadores industriales su acceso, pues sus licencias comerciales son de un alto costo. Ante esta situación surgen nuevos sistemas de diseño que son de código abierto, como es el caso de las herramientas *LibreCAD* y *Blender*. Otra herramienta que se desea que en un futuro sea de código abierto, pues aun su licencia no se ha definido es *AsiXMec*, un sistema que se desarrolla en el proyecto Diseño y Simulación de Estructuras Mecánicas (DISEM) en el Centro de Entornos Interactivos 3D (VERTEX), perteneciente a la Facultad 5 de la Universidad de las Ciencias Informáticas (UCI). Este *software* surge con el propósito de apoyar el diseño de prototipos virtuales de estructuras mecánicas, para su posterior construcción en empresas manufactureras. *AsiXMec* es un modelador geométrico y paramétrico basado en historial de operaciones. Algunas de las funcionalidades que brinda son manipular, crear, editar, importar o exportar archivos compatibles con otros sistemas para Linux como *Salome-Meca*, *DraftSight*, *LibreCAD*, o para *Windows* como *AutoCAD*, *SolidWorks* e *Inventor*.

Entre los módulos que lo componen se encuentra el *sketcher* o boceto, el cual consiste en un área de trabajo donde se crean o modifican entidades geométricas de dos dimensiones (arco, línea, polígono, círculo, punto, rectángulo y elipse). Es en el *sketcher* donde un diseñador define las características que puede tener una pieza mecánica. Un ingeniero mecánico que se encuentre en pleno proceso de diseño de una pieza, puede hacer uso de las funcionalidades que brindan otros de sus módulos, como es el caso del módulo *Snap*, el cual facilita la creación de dichas entidades de forma precisa. Actualmente, este último presenta ciertas dificultades que atentan contra la calidad del *software*. Una de ellas consiste en que si el equipo de desarrollo desea incorporar a la herramienta otros *snaps* (como es el caso del *snap* tangente), que son de gran utilidad para asistir a los usuarios en sus diseños o mejorar los que ya existen, es necesario modificar el núcleo

y compilar completamente el sistema. Esto trae consigo, que si en un futuro se brindan actualizaciones del módulo, es obligatorio proveer el *software* en su totalidad. Por la alta dependencia entre los módulos del sistema si se realizan cambios en el módulo *Snap*, es necesario replicar los cambios a los módulos dependientes. Un ejemplo es el módulo de las restricciones, pues existe una dependencia de parámetros como el tipo de entidad, el tipo de restricción a emplear y el punto que se proyecta sobre la entidad con la cual se trabaja.

A partir de la situación problemática anterior se define el siguiente **problema a resolver**: ¿Cómo reducir el acoplamiento del módulo *Snap*, para que no afecte la calidad del producto *AsiXMec*?

Para dar solución al problema anterior se propone como **objeto de estudio** las técnicas para reducir el acoplamiento en un *software*, y se persigue como **objetivo general** desarrollar la versión 2.0 del módulo *Snap* para apoyar el proceso de creación de entidades 2D en el *sketcher* del producto *AsiXMec*. El **campo de acción** está enmarcado en la creación de *plugins* mediante la arquitectura que brinda el *framework Qt*.

Para el logro de dicho resultado se proponen las siguientes **tareas de investigación**:

- Identificación de las principales herramientas *CAD*.
- Caracterización del módulo *Snap* actual en el *software AsiXMec*.
- Investigación y análisis de las técnicas que existen para reducir el acoplamiento en un *software*.
- Descripción de la arquitectura de *plugins* que provee el *framework Qt*.
- Diseño e implementación de la versión 2.0 del módulo *Snap*.
- Validación del desempeño del módulo propuesto mediante las pruebas de aceptación.

Los métodos empleados para dar solución a las tareas de investigación son los **métodos científicos teóricos y empíricos**.

Métodos Teóricos:

- Analítico-Sintético: Mediante su uso se analizará la información obtenida en el marco teórico de la investigación referente a las principales técnicas de reducción del acoplamiento en un sistema informático, para posteriormente realizar una síntesis que permita arribar a una propuesta de solución al problema.
- Histórico-Lógico: Este método se utiliza en el estudio del estado del arte, para identificar los tipos de *snaps* que existen y las herramientas de diseño computarizado que hacen uso de ellos.
- Modelación: El método será empleado en la creación de un prototipo funcional que muestre los *snap* que contiene el módulo.

Métodos Empíricos:

- Consulta de las fuentes de información: Mediante este método se seleccionará la información necesaria para construir el marco teórico.
- Consulta de especialistas: El método se empleará para recibir los criterios de validación sobre la aplicabilidad y utilidad del trabajo realizado.
- Observación: Este método se utilizará para apreciar la influencia de las técnicas para reducir el acoplamiento, en la calidad del software *AsiXMec*.
- Pruebas: Este método permitirá validar los resultados obtenidos con la solución propuesta a partir de la investigación realizada.

El documento está estructurado en 3 capítulos. En el Capítulo 1 se abordan los elementos teóricos que sirven de base a la investigación del problema planteado, los cuales están vinculados a las técnicas que emplean las herramientas de diseño para posicionar determinados elementos en el área de trabajo, con el objetivo de minimizar el tiempo de desarrollo. Se brinda además un estudio sobre algunas de las técnicas que existen para reducir el acoplamiento de los componentes que integran un *software*. En este capítulo se hace además un análisis de la arquitectura de *plugins* que ofrece el *framework Qt*. En el Capítulo 2 se expone la propuesta de solución para resolver el problema de investigación, así como los principales artefactos generados en la Ingeniería de Software realizada a la aplicación. Por su parte, el Capítulo 3 recoge los resultados obtenidos luego de la realización de las pruebas de calidad a la nueva versión del módulo, así como la validación de la propuesta de solución.

CAPÍTULO 1

Fundamentación Teórica

Introducción

En el presente capítulo se hace un estudio de los principales conceptos utilizados durante la investigación y que su comprensión podría resultar compleja una vez propuesta la solución, especificándose algunas de las tecnologías *CAD* más empleadas en la actualidad, así como los diferentes campos de aplicación que estas poseen. Se analizan además los *snaps* como técnica para posicionar elementos en el *sketcher* de una herramienta de diseño, así como buenas prácticas que se pueden emplear durante la fase de diseño para reducir el acoplamiento entre los módulos que componen la herramienta *AsiXMec*, con el propósito de mejorar su calidad.

1.1. Herramientas de diseño gráfico computarizado

En la actualidad el diseño de nuevos productos o la modificación de los ya existentes, se ha convertido en un elemento fundamental para la mejora de la capacidad de innovación y competitividad de las empresas industriales. La forma de lograr un diseño con mayor precisión, a un menor costo y que mejore los estándares de producción, ha ido incrementándose en los últimos años (Saldaña Pomazunco, 2012). Tradicionalmente se hacía uso de un lápiz y papel para realizar los dibujos, y si en algún momento el documento no tenía la calidad requerida, era necesario reemplazarlo por otro, lo que traía como consecuencia la pérdida de recursos y tiempo de trabajo. Sin embargo, al desarrollarse las Tecnologías de la Informática y las Comunicaciones (TIC), fueron apareciendo nuevas ideas para resolver ese tipo de problemas.

Con el objetivo de ayudar a profesionales o aficionados en la creación de diseños más eficientes, fueron creadas las herramientas *CAD*, las cuales proporcionan una interfaz gráfica para que el usuario interactúe con ellas a través de dispositivos de entrada, como un teclado, un *mouse* o un lápiz óptico. Los campos de aplicación de los sistemas *CAD* pueden ser variados, como la ingeniería, la arquitectura, el diseño automotriz, el diseño gráfico, la publicidad, el diseño de moda y la medicina (Jackson et al., 2013).

Varias empresas se han destacado a través de los años por crear aplicaciones *CAD* con un gran número de funcionalidades, con el objetivo de apoyar a los usuarios en el diseño computarizado. Entre ellas se encuentran *Parametric Technology Corporation (PTC)*, *Dassault Systèmes* y *Autodesk*. Esta última se funda en la década de los años 80 del siglo pasado y que en la actualidad es una de las empresas líderes en el mercado de herramientas *CAD*. *Autodesk* se dedica al desarrollo de *software* de diseño en 2D y 3D para las industrias manufactureras, de infraestructuras, de construcción, así como en la realización de productos para el entretenimiento (Caño, Cruz y Solano, 2007), destacándose en este último campo las herramientas de diseño *Maya*, *3ds Max* y *Blender*, siendo la última mencionada una alternativa libre.

LibreCAD

LibreCAD es una aplicación informática de código libre para el diseño en 2D. Funciona en los sistemas operativos *GNU/Linux*, *Mac*, *Solaris* y *Microsoft Windows*. Fue desarrollada por los miembros de una comunidad con el nombre de *QCad Community Edition*. El desarrollo de *LibreCAD* está basado en las bibliotecas de *Qt4* y se puede ejecutar en plataformas de 32 y 64 *bits*. Gran parte de la interfaz y de los conceptos sobre su uso son similares a los de *AutoCAD*, permitiendo a los usuarios con experiencia en este tipo de herramientas, trabajar con mayor comodidad. *LibreCad* viene con varios *plugins*, plantillas o herramientas para desarrollar diseños geométricos con calidad. Con cada actualización de *LibreCAD* se incorporan nuevas características para realizar proyectos más complejos. Si bien es un programa que usan los profesionales, es también muy intuitivo para los usuarios que se inician en el mundo del diseño computarizado (Community, 2014).

Autodesk AutoCAD

Es un *software* utilizado para el dibujo en 2D y el modelado en 3D, que actualmente se desarrolla y comercializa por la empresa *Autodesk*. Está orientado al dibujo digital de planos de edificios, por lo que es un programa que usan los arquitectos, ingenieros y diseñadores industriales. Este *software* gestiona una base de datos de entidades geométricas, con la que se puede operar a través de una pantalla gráfica en la que se muestran (Lockhart, 2013). *AutoCAD* utiliza una arquitectura basada en *plugins* para añadir nuevas funcionalidades que contribuyen a satisfacer los nuevos requerimientos de los usuarios. Un ejemplo es el *plugin nXtRender*, el cual se integra de manera rápida y fácil con el objetivo de ayudar a crear representaciones realistas de los bocetos realizados. Este *plugin* optimiza el tiempo de diseño al guardar los ajustes junto con el modelo, además permite modificar y actualizar el renderizado sin necesidad de repetir todo el proceso. (Autodesk, 2015).

Autodesk Inventor

Es un paquete de modelado de sólidos en 3D producido por la empresa de *software Autodesk*. Entró en el mercado en 1999 y se agregó a las series de diseño mecánico de *Autodesk* como una respuesta de

la empresa, a la creciente migración de su base de clientes de diseño mecánico en dos dimensiones, hacia la competencia. Este *software* de diseño computarizado apoya a los ingenieros mecánicos en el diseño de piezas (Hansen, 2013). *Inventor* también utiliza *plugins*, como por ejemplo *Screenshot*, el cual se utiliza para simplificar el proceso de documentación de piezas, que funciona a partir de su versión del 2010 en 32 ó 64 *bits*. *Screenshot* permite capturar imágenes o documentos desde la ventana de la aplicación. Opcionalmente permite modificar el color de fondo a blanco y los objetos a negro, además de convertir la imagen a escala de grises. Otra de sus funcionalidades es que puede enviar directamente las imágenes a la impresora. (CadStock, 2010).

SolidWorks

Es un *software CAD* para modelado en 3D, desarrollado por *SolidWorks Corp.*, una filial de *Dassault Systèmes*, S.A., para el sistema operativo *Microsoft Windows*. Su primera versión fue lanzada al mercado en 1995 con el propósito de hacer la tecnología *CAD* más accesible. El programa permite modelar conjuntos de piezas, así como extraer planos técnicos. Uno de los *plugins* más utilizados por los usuarios cuando trabajan con *SolidWorks*, es el *Power Surfacing* o también conocido como *nPower*. Este *plugin* permite importar objetos poligonales que no hayan sido creados con la herramienta, además de diseñar superficies complejas (Systèmes, 2014b).

Computer Aided Three-dimensional Interactive Application (*CATIA*)

Es un programa informático de diseño, ingeniería y fabricación asistida por computadora realizado por la empresa *Dassault Systèmes*. El programa está desarrollado para proporcionar apoyo desde la concepción del diseño hasta la producción y el análisis de productos. Está disponible para sistemas operativos como *Microsoft Windows*, *Solaris*, entre otros. Provee una arquitectura abierta para el desarrollo de aplicaciones o para personalizar el programa, además de que en la actualidad, se profundiza en el manejo de superficies complejas mediante esta herramienta. Uno de los mayores campos de aplicación de *CATIA*, es la industria automotriz para el diseño y desarrollo de componentes de carrocería. Entre los *plugins* que emplea se encuentra *Simulayt Modeler*, dirigido a ayudar a los diseñadores e ingenieros a integrar los diseños. También les permite hacer el análisis y la fabricación de estructuras hechas de materiales de fibra, como las telas. Mediante la simulación de la fabricación de estos productos, los problemas pueden ser previstos al principio del ciclo de desarrollo, lo cual mejora el presupuesto disponible (Systèmes, 2014a).

Solid Edge

Solid Edge es un programa parametrizado de diseño asistido por computadora de piezas tridimensionales. Permite el modelado de piezas de distintos materiales, doblado de chapas, ensamblaje de conjuntos, soldadura, funciones de dibujo en plano para ingenieros, entre otros. Esta herramienta puede gestionar datos de conjunto desde las primeras fases de planificación del proyecto, hasta los ciclos de revisión, fabrica-

ción, mantenimiento y archivado. Con *Solid Edge* se generan modelos virtuales muy precisos incorporando conocimiento de ingeniería para evitar errores costosos y trabajo innecesario (Software, 2014). Entre los *plugins* que emplea se encuentra *EcoDesigner*, desarrollado por la empresa *Trayak*, dedicada a la consultoría y soluciones de *software* centradas en la sostenibilidad de los productos. Este *plugin* permite seleccionar automáticamente el material asignado, el peso y el volumen de todas las piezas de un conjunto, lo que le permite a un usuario poder experimentar con diferentes materiales. En la actualidad también está disponible para la herramienta *Autodesk Inventor* (Trayak, 2014).

1.2. Técnicas de posicionamiento en herramientas de diseño gráfico computarizado

Un elemento que tienen en común varias de las herramientas de diseño gráfico computarizado, son las técnicas para posicionar elementos en el área de trabajo, las que permiten al usuario diseñar objetos con mayor calidad y exactitud. Entre estas técnicas se destacan las guías, las reglas, las restricciones, el *grid* o cuadrícula, y los *snaps*. Cada una de las herramientas *CAD* que se mencionaron en el epígrafe anterior hacen uso de algunas de estas técnicas; sin embargo por el auge que han tomado y las ventajas que facilita a un usuario, otras herramientas de diseño como *CorelDRAW*, una aplicación informática de diseño gráfico vectorial, el editor de imágenes *Adobe Photoshop* y la *suite* de oficina *LibreOffice*, también las emplean.

Guías

Las guías son una técnica para ajustar los objetos con precisión, las cuales se emplean en herramientas de diseño y tratamiento de imágenes como *Adobe Photoshop (Figura 1.1)*. Las operaciones con guías se pueden realizar de diferentes maneras, por lo que hay varias formas para realizar una misma operación. Se presentan como líneas verticales u horizontales que el usuario puede crear o desplazar, teniendo en cuenta lo que esté realizando (Vértice, 2010).

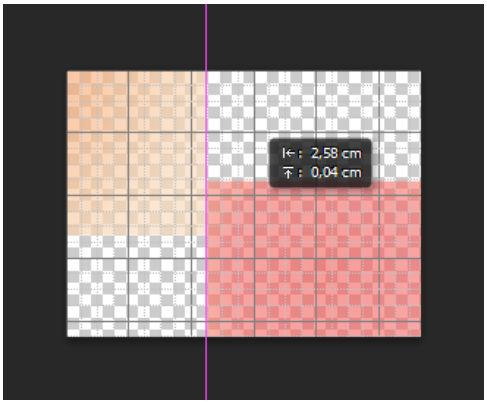


Figura 1.1. Guías en *Adobe Photoshop*.

Reglas

Las reglas se utilizan para colocar los objetos de forma precisa *(Figura 1.2)*. Por lo general, se muestran en la parte superior y lateral izquierda del espacio de trabajo. Al mover el *mouse*, se puede apreciar una línea discontinua tanto vertical como horizontal, que indica el origen de coordenadas del puntero (Vértice, 2010).

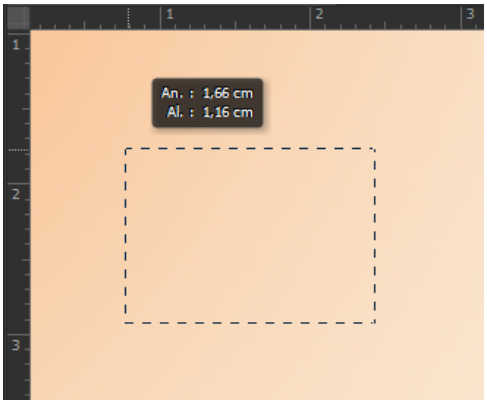


Figura 1.2. Reglas en *Adobe Photoshop*.

Restricciones

Las restricciones ayudan en el proceso de manipulación directa de un objeto, ya que proporcionan al usuario otra vía para ubicar los objetos de forma precisa, lo cual constituye una funcionalidad útil en los sistemas de diseño y modelación. Estas pueden aparecer en forma de símbolos *(Figura 1.3)* o en un menú *(Figura 1.4)* donde es posible seleccionar la restricción concreta con la que se desea trabajar (R. Shih, 2013).

Figura 1.3. Restricciones en la herramienta *Autodesk AutoCAD*.

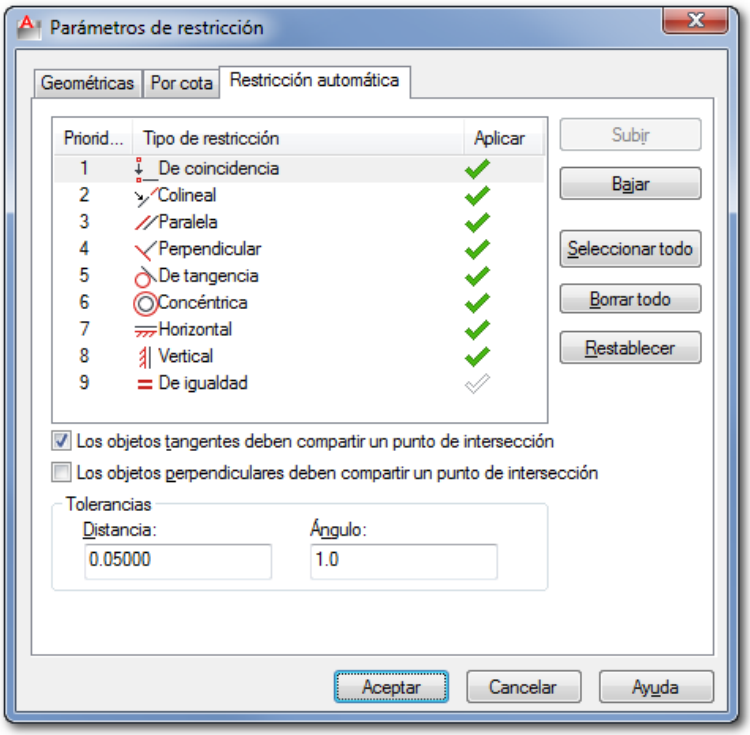


Figura 1.4. Menú de restricciones en la herramienta *Autodesk AutoCAD*.

Grid

El *grid* o cuadrícula es un elemento que ayuda a colocar objetos de manera simétrica *(Figura 1.5)*. Su representación es una rejilla que divide el área de trabajo en rectángulos de iguales dimensiones. El usuario puede definir la dimensión que tendrá cada rectángulo, y puede seleccionar si desea mostrar el grid mientras está trabajando. Algunas de las propiedades de la cuadrícula que se pueden configurar son el color, el estilo de las líneas (continua o discontinua) y el número de divisiones (Vértice, 2010).

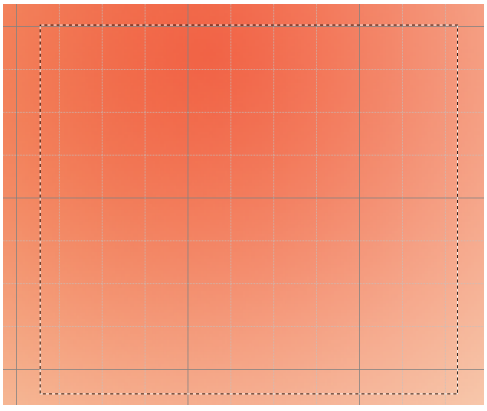


Figura 1.5. *Grid* o cuadrícula en *Adobe Photoshop*.

Por la importancia que presenta la técnica de posicionamiento basada en los *snap* en las tecnologías *CAD*, es necesario profundizar en ella en un próximo epígrafe, ya que es imprescindible que el usuario tenga un amplio conocimiento de los principales conceptos que intervienen en esta técnica.

1.3. Los *snap* en herramientas de diseño gráfico computarizado

En las herramientas *CAD* una de las técnicas empleadas por los usuarios para lograr precisión en sus diseños, son los *snaps* o puntos de referencia a objetos. Un sistema *CAD* permite a un usuario emplear los *snaps* para facilitar la colocación precisa al mover un objeto a una ubicación deseada. El término objeto se refiere a las entidades con las cuales puede trabajar un diseñador; así como su combinación para la creación de otras entidades (Jackson et al., 2013). Algunas herramientas de diseño computarizado como *AutoCad* e *Inventor*, utilizan la expresión *object snap* (R. H. Shih, 2014) para referirse a los *snaps*; sin embargo en el presente trabajo se empleará el término *snap*.

Con el objetivo de que exista una mejor comprensión del término, se describe a continuación el comportamiento del *snap* perpendicular de la herramienta *AsiXMec (Figura 1.6)*. Cuando un usuario está diseñando, al dar clic sobre el *sketcher*, se captura el punto del cursor y se guardan sus coordenadas. Posteriormente se proyecta ese punto sobre la entidad que ya se dibujó en el *sketcher*, con el objetivo de buscar aquel punto en el cual se obtendrá un objeto que cumpla con las características deseadas, adquiriendo así un comportamiento de atracción y facilitando la exactitud del boceto realizado. En este ejemplo la entidad sobre la cual se proyecta el punto es la entidad recta, y el punto que se desea encontrar es aquel donde la entidad que se desea dibujar, es perpendicular a la primera.

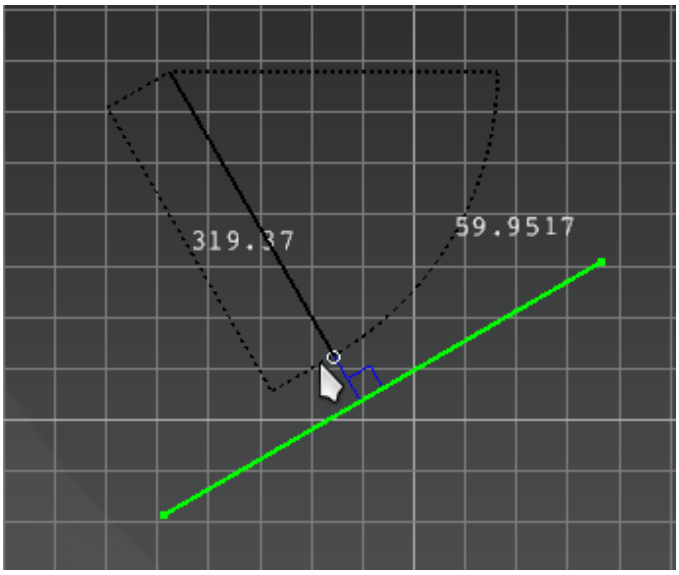


Figura 1.6. Visualización del *snap* paralelo en la herramienta *AsiXMec*.

En estos sistemas los usuarios pueden trabajar con los *snaps* de varias maneras. Estas pueden ser en segundo plano o invocándolos individualmente según se requiera. Cuando se habla de "segundo plano", se refiere a que están implementados para que se ejecuten todo el tiempo mientras se diseñan entidades geométricas el *sketcher*; es decir, que están disponibles siempre que se necesiten, ya que el sistema es capaz de reconocer cuál ejecutar en cada caso. La segunda forma consiste en una interfaz gráfica compuesta por un menú que los contiene *(Figura 1.7)*, donde pueden ser habilitados o deshabilitados en tiempo de diseño (Jackson et al., 2013).

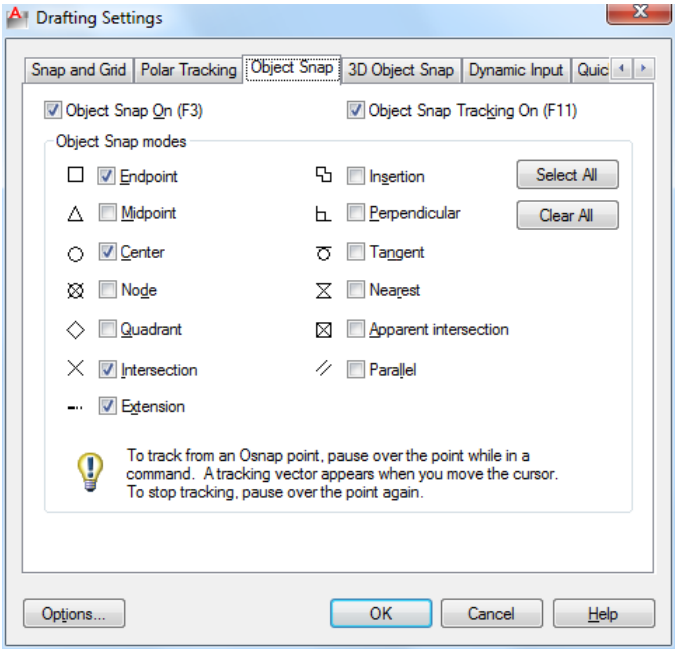


Figura 1.7. *Visualización de los snaps en la herramienta AutoCAD.*

1.3.1. Tipos de *snap*

Muchos de los sistemas de diseño asistido por computadora que emplean esta técnica, utilizan varios *snaps* para asistir al usuario en sus diseños. Cada uno de ellos corresponde a una entidad específica, teniendo en cuenta los puntos de interés que puede tener (Watson, 2015).

- **Punto final**: Se usa para obtener el punto final de una línea, arco u otro objeto que tiene un final definido. Este *snap* se usa para unir líneas y acotar distancias *(Figura 1.8)*.

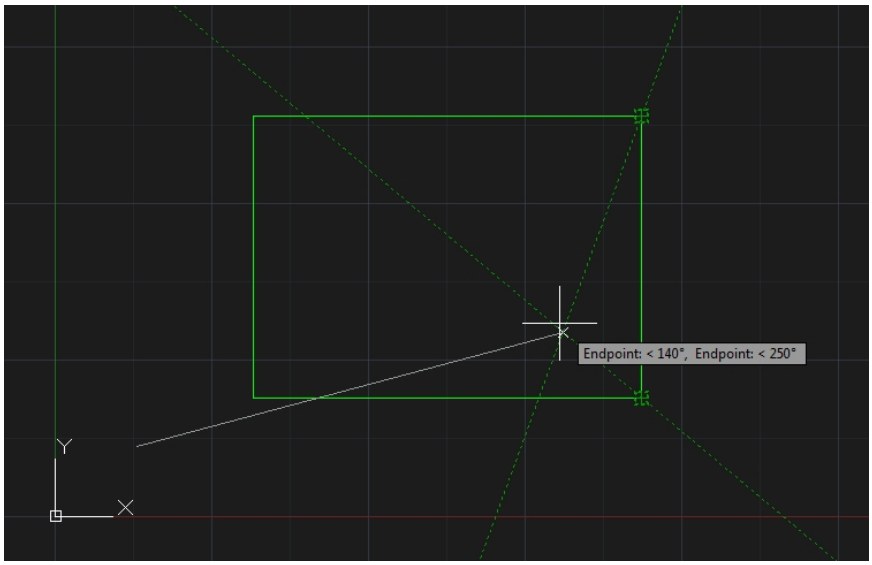


Figura 1.8. *Object snap End point* en la herramienta *AutoCAD*

- **Centro**: Se emplea para encontrar el centro de círculos, arcos o elipses *(Figura 1.9)*.

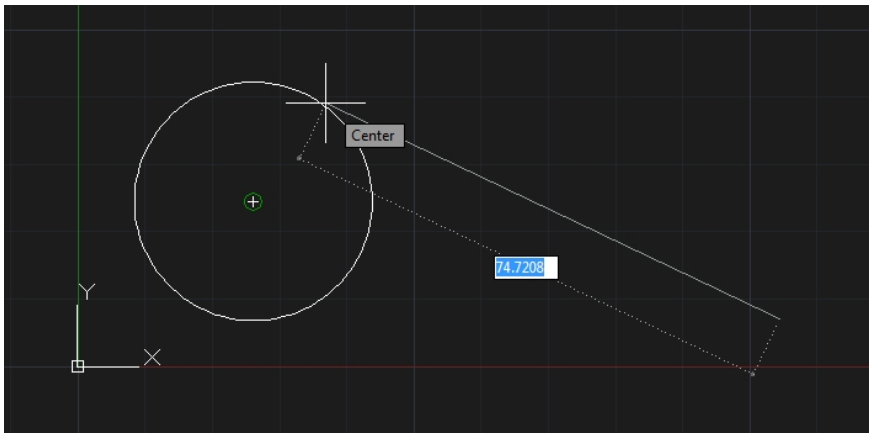


Figura 1.9. *Object snap Center* en la herramienta *AutoCAD*

- **Punto medio**: Se utiliza para encontrar el punto medio de cualquier objeto que tiene un principio y un final. Todas las líneas y arcos poseen un punto medio. Los círculos tienen centro pero no un punto medio *(Figura 1.10)*.

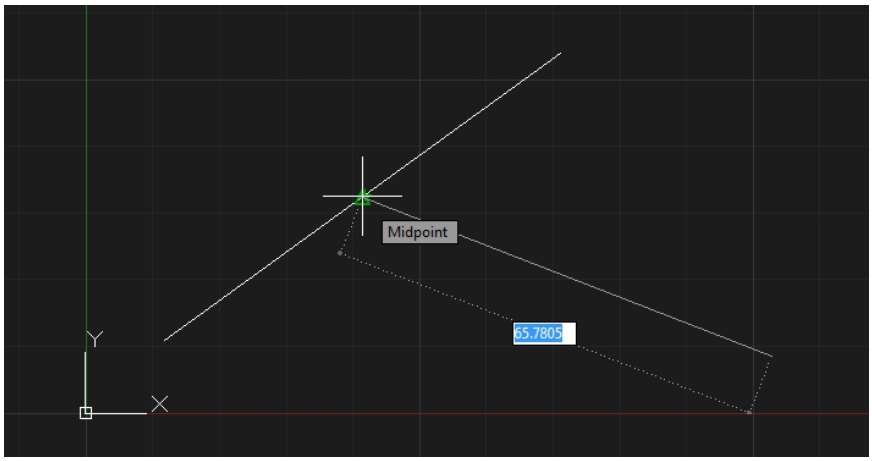


Figura 1.10. *Object snap Middle point* en la herramienta *AutoCAD*

- **Tangente**: Se utiliza para buscar el punto de tangencia en círculos, arcos, elipses y *splines (Figura 1.11)*.

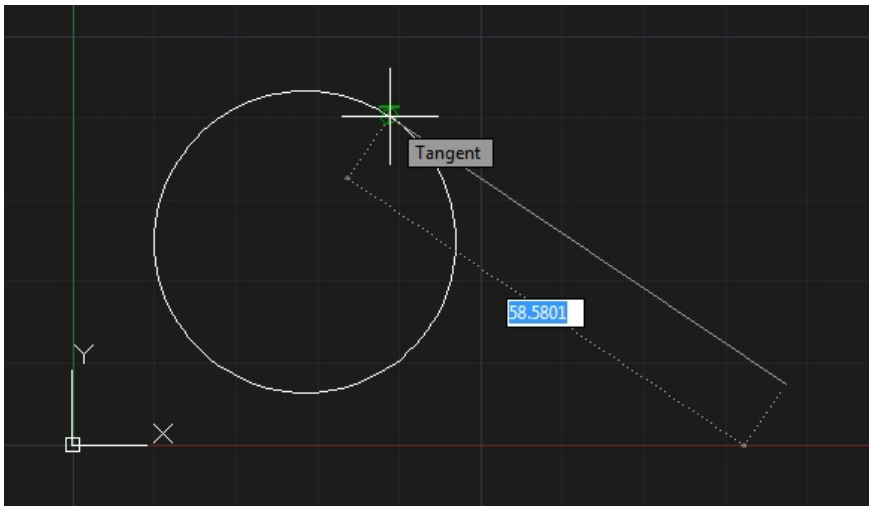


Figura 1.11. *Object snap Tangent* en la herramienta *AutoCAD*

- **Intersección**: Se ajusta a la intersección física de cualquiera de los dos objetos de dibujo; es decir, donde líneas, arcos o círculos, se cruzan entre sí.

- **Intersección aparente**: Se usa cuando dos objetos parecen intersectarse en la pantalla, pero no se cortan verdaderamente en el espacio en 3D. También funciona cuando dos objetos no se intersectan, pero se necesita encontrar el punto donde sí lo harían *(Figura 1.12)*.

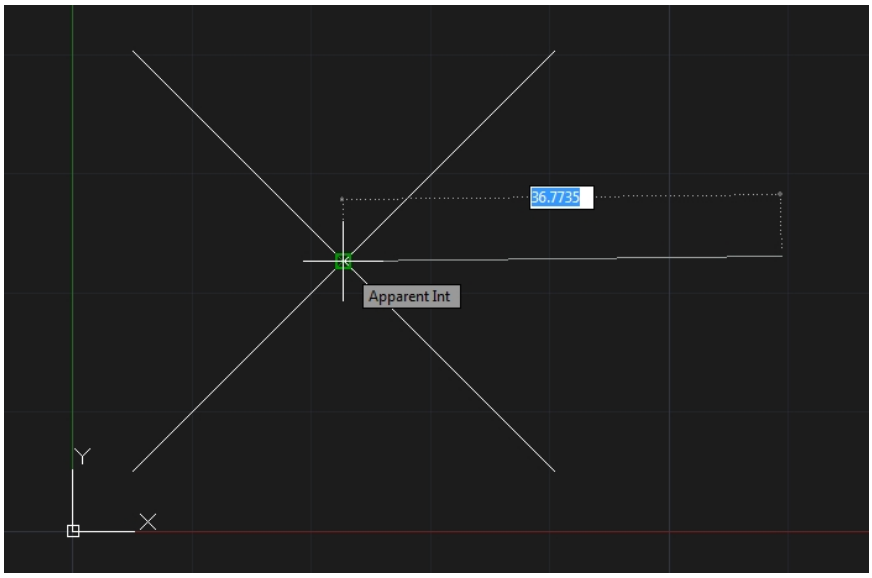


Figura 1.12. *Object snap Apparent intersection* en la herramienta *AutoCAD*

- **Perpendicular**: Se utiliza para dibujar una línea desde un punto hasta otro, con un ángulo recto (90 grados) relativo a un objeto *(Figura 1.13)*.

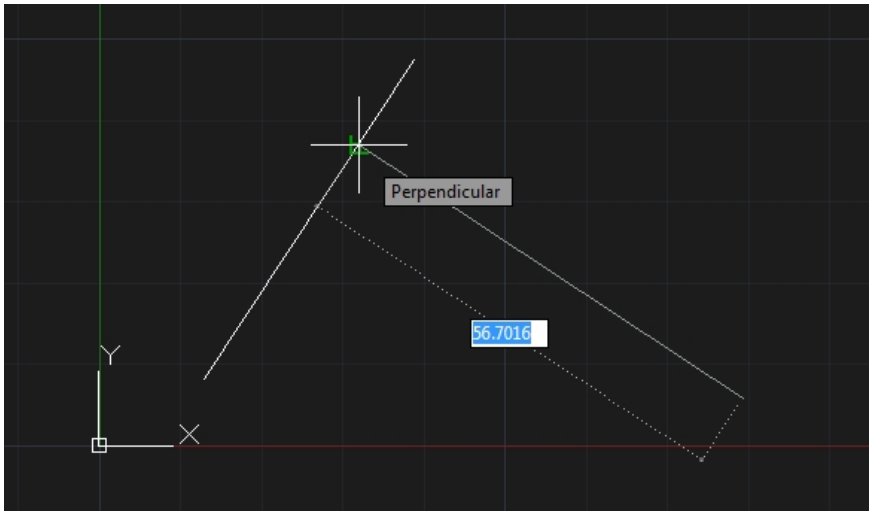


Figura 1.13. *Object snap Perpendicular* en la herramienta *AutoCAD*

- **Paralelo**: Se utiliza para dibujar una línea paralela a cualquier otra línea en el dibujo *(Figura 1.14)*.

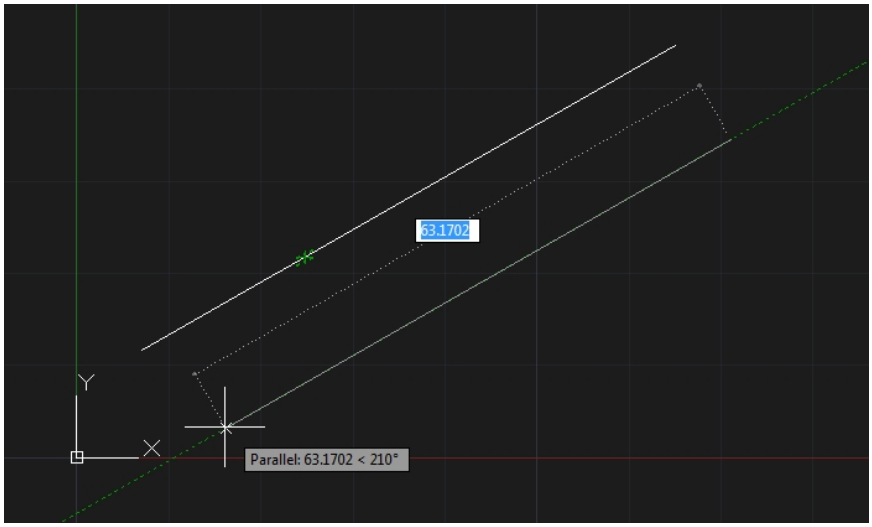


Figura 1.14. *Object snap Parallel* en la herramienta *AutoCAD*

- **Más cercano**: Encuentra el punto más cercano sobre un objeto relativo a donde se comenzó. Es útil para medir distancias y dibujar líneas rápidamente *(Figura 1.15)*.

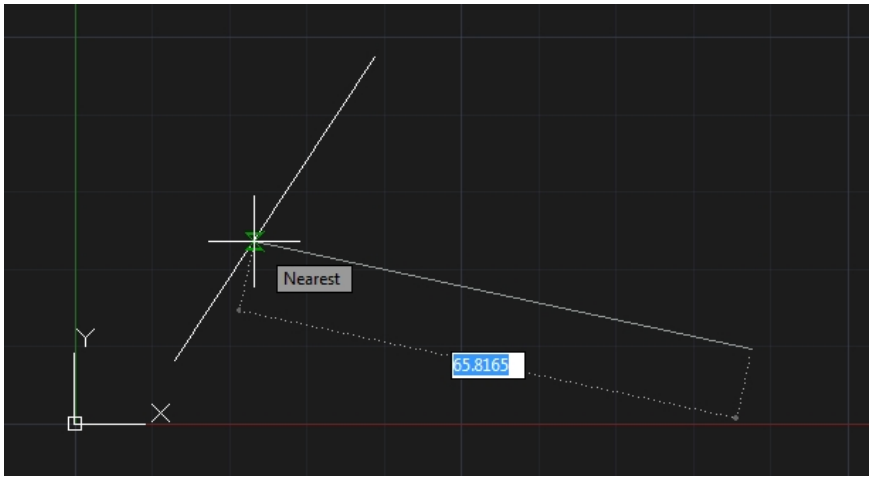


Figura 1.15. *Object snap Nearest* en la herramienta *AutoCAD*

1.4. Herramienta *AsiXMec*

AsiXMec es una herramienta *CAD* que se desarrolla en el proyecto DISEM del Centro de Entornos Interactivos 3D (VERTEX) de la UCI. Entre las funcionalidades que brinda el sistema se encuentra el soporte para *plugins*, además de contar con una interfaz *ribbon* en la cual el ingeniero mecánico puede seleccionar la opción que desee, para el diseño y modelado de una pieza. Cuenta con un *solver* de restricciones, permite el

diseño asistido por *snaps* e identificación automática de caras. Su interfaz gráfica es sencilla lo cual la hace de fácil manejo por los usuarios, además de brindar soporte de internacionalización para los lenguajes Inglés y Español. Entre los módulos que componen la herramienta se encuentra el módulo *Snap*, el cual contiene actualmente 15 *snaps* que ayudan al usuario a lograr mayor precisión en sus bocetos, y de los cuales solo se muestran al usuario 11 *snaps (Figura 1.16)*.

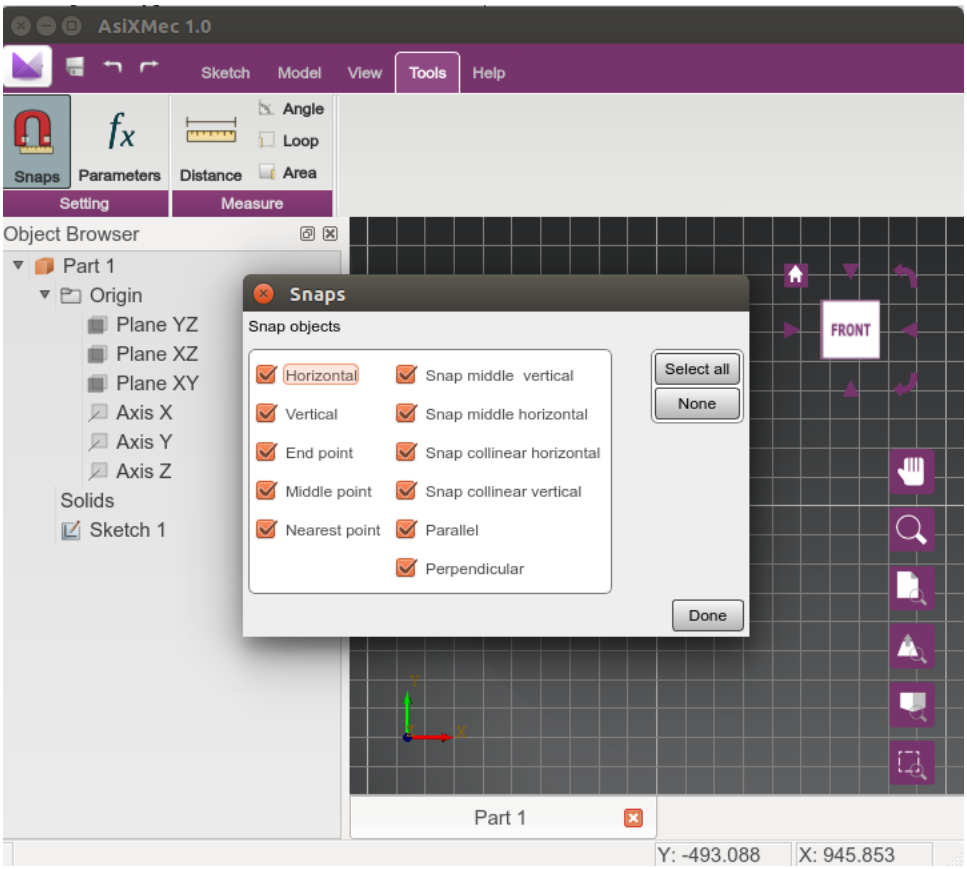


Figura 1.16. Módulo *Snap* de la herramienta *AsiXMec*

El módulo en cuestión presenta ciertas dificultades que atentan contra la calidad del software. Un ejemplo de ello consiste en que si el equipo de desarrollo desea adicionar nuevos *snaps* al módulo, o hacerle algunas mejoras a aquellos que ya contiene, es necesario modificar el núcleo de la aplicación (*cad-core*) y realizar allí los cambios que sean necesarios. Esto ocurre debido a que existe una gran relación entre este módulo y otros de la herramienta, como es el caso de los módulos de las entidades y las restricciones. Al pintarse en el *sketcher* un *snap*, este devuelve tres elementos con los cuales trabaja el módulo de las restricciones. Estos elementos son el tipo de entidad, el tipo de restricción que se puede usar y el punto que se proyecta sobre la entidad con la cual se trabaja.

Actualmente, todos los *snaps* se cargan al iniciar la aplicación, por lo que un usuario no puede decidir si los puede cargar o no, ya que esta funcionalidad se encuentra fusionada con el código del proyecto. Una

vez que sea concebida la licencia sobre la cual será comercializado *AsiXMec*, se pretende que la funcionalidades que brindan los *snaps* puedan ser comercializadas con una licencia diferente. Teniendo en cuenta las condiciones actuales del módulo *Snap*, es necesaria la búsqueda de una vía factible que permita reducir el acoplamiento en este *software* con el propósito de aumentar su calidad.

1.5. Buenas prácticas para mejorar la calidad de un *software*

Un *software* de calidad es el resultado de la unión de varios factores. Es necesario aplicar correctamente los principios y prácticas fundamentales de diseño, que el equipo de desarrollo utilice como guía una metodología que satisfaga cada una de las necesidades del producto, además de tener un control sistemático y exhaustivo sobre el progreso que se tiene en cada una de las fases de su creación. Una de las más importantes es la etapa de diseño, durante la cual se deben tener en cuenta ciertos términos que inciden en la calidad que pueda tener una aplicación informática, los cuales se muestran a continuación.

Modularidad

El *software* monolítico, es decir, un programa grande compuesto por un módulo sencillo, es difícil de entender para un ingeniero de *software* debido a la complejidad general que puede presentar (R. S. Pressman, 1997). Dentro de la programación orientada a objetos, la modularidad juega un papel muy importante. Una vez que se ha representado una situación del mundo real en un programa, se tiene regularmente como resultado, un conjunto de objetos de *software* que constituyen la aplicación. La modularidad, permite poder modificar las características de la clase que definen a un objeto, de forma independiente de las demás clases en la aplicación. En otras palabras, si la aplicación puede dividirse en módulos separados, normalmente clases, y estos módulos pueden compilarse y modificarse sin afectar a los demás, entonces dicha aplicación ha sido implementada en un lenguaje de programación que soporta la modularidad. La tecnología orientada a objetos brinda esta propiedad para hacer uso de ella en el *software* que se esté desarrollando (Joyanes Aguilar, 1996).

Independencia funcional

Se consigue al desarrollar módulos con una función determinante y que no posean una interacción excesiva con otros módulos. Se desea desarrollar el *software* de tal manera que cada módulo aborde una funcionalidad específica de los requisitos, y tenga una sola interfaz cuando se observe desde otras partes de la estructura del programa (Sommerville, 2005). Un *software* con módulos independientes es más fácil de desarrollar, debido a que las funciones se pueden fraccionar y las interfaces se simplifican. Cada módulo independiente es más fácil de mantener porque se limitan los efectos secundarios que originan modificaciones al diseño o al código, se reduce la propagación de errores y es posible emplear módulos reutilizables. En resumen, la independencia funcional es una clave para el buen diseño y este a su vez, es clave para lograr la

calidad del *software* (S. R. Pressman, 1992). La independencia se evalúa aplicando dos criterios cualitativos: cohesión y acoplamiento.

Cohesión

En informática, la cohesión hace referencia a la forma en que se agrupan las unidades de *software* (módulos) en una unidad mayor. Por ejemplo: la forma en que se agrupan funciones en una biblioteca de funciones o la forma en que se agrupan métodos en una clase. El consenso general para una buena programación o un buen diseño es que la cohesión debe ser alta, mientras más cohesionados estén los elementos, mejor. La cohesión de cualquier *software* determina la fortaleza de su código fuente (Reynoso et al., s.f.).

Acoplamiento

En un *software* indica el nivel de dependencia que existe entre los componentes que lo integran; es decir, el grado en que cada uno de ellos puede funcionar sin recurrir a otros. Un ejemplo simple de acoplamiento es cuando un componente accede directamente a un dato que pertenece a otro componente. En ese caso, el resultado del comportamiento del componente A dependerá del valor del componente B, por lo tanto, están acoplados (Sommerville, 2005).

El bajo acoplamiento entre las unidades de *software* es el estado ideal que siempre se intenta obtener para lograr un buen diseño. Cuanto menos dependiente sean las partes que constituyen un sistema informático, mejor será el resultado. Sin embargo, es difícil obtener un desacoplamiento total de las unidades, por lo cual el objetivo final del diseño de *software* es reducir al máximo el acoplamiento entre componentes. El bajo acoplamiento permite mejorar la mantenibilidad de los módulos del *software* así como aumentar su reutilización. Permite además evitar el efecto ola que se presenta cuando surgen problemas en un lugar y después, se propaga a través del sistema, lo cual hace más difícil detectar dónde está el error. Un bajo acoplamiento en un *software* minimiza el riesgo de tener que cambiar múltiples unidades del sistema, cuando se producen cambios en una de ellas (R. S. Pressman, 1997; Reynoso et al., s.f.).

1.5.1. Técnicas para reducir el acoplamiento en un *software*

Reducir el acoplamiento en un *software* significa saber definir las responsabilidades de cada componente para garantizar que la dependencia sea funcional o arquitectónica, no de implementación. En el caso del acoplamiento funcional, está bien que un componente de cálculo de probabilidades dependa de un componente de cálculo matemático básico, ya que es evidente que para calcular probabilidades será necesario aplicar fórmulas matemáticas y resolver operaciones aritméticas. Por su parte, el acoplamiento arquitectónico se puede ver como ejemplo en el esquema Modelo-Vista-Controlador (MVC). La capa del modelo realizará cambios de acuerdo con los parámetros que recibe del controlador, por lo tanto hay cierta dependencia, pero sólo como separación de capas de una arquitectura claramente definida (Sommerville, 2005). Los integrantes del equipo de desarrollo del *software* son los encargados de que los componentes que lo integran tengan entre

ellos el más bajo acoplamiento posible, existiendo así varias técnicas que estos pueden emplear para evitar que exista un bajo grado de acople.

Patrones arquitectónicos

Los patrones de arquitectura se pueden ver como la descripción de un problema en particular y recurrente de diseño, que aparece en contextos arquitectónicos específicos y que representa un esquema genérico. Este esquema se especifica mediante la descripción de los componentes que la constituyen, sus responsabilidades y desarrollos, así como también la forma en la que estos colaboran entre sí. Los patrones arquitectónicos se definen sobre aspectos fundamentales de la estructura del *software*, donde se especifican un conjunto de subsistemas con sus responsabilidades y una serie de recomendaciones para organizar los distintos componentes. A diferencia de los patrones de diseño de *software* que están orientados a objetos y clases (patrones creacionales, estructurales, de comportamiento y de interacción), los patrones de arquitectura están a un mayor nivel de abstracción (ibíd.).

Patrones de diseño

Los patrones de diseño son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de *software* y otros ámbitos referentes al diseño de interacción o interfaces. Un patrón de diseño provee un esquema para refinar los subsistemas o componentes de un *software*, así como las relaciones entre ellos. Son menores en la escala de abstracción que los patrones arquitectónicos, además de que tienden a ser independientes de los lenguajes y paradigmas de programación. Estableciendo patrones de diseño, se hace más fácil construir soluciones específicas que tienen determinadas estructuras o piezas en común, reutilizando al mismo tiempo experiencias adquiridas (Freeman et al., 2005; Sommerville, 2005). Un ejemplo de los patrones de diseño que se emplean con el objetivo de que exista un menor acople entre los módulos de un *software* es el MVC, muy empleado en el desarrollo de sistemas *web*.

Plugins

El término *plugin* no forma parte del diccionario de la Real Academia Española (RAE). Se trata de un concepto de la lengua inglesa que puede entenderse como inserción y que se emplea en el campo de la informática. Un *plugin* es un complemento el cual una vez adicionado a un *software*, le incorpora una nueva funcionalidad. Lo habitual es que el *plugin* se ejecute mediante el *software* principal, con el que interactúa a través de una interfaz (Definición.de, 2015).

En la actualidad, varios programas informáticos trabajan con *plugins*. En un gestor de contenidos, como *WordPress*, un *plugin* puede servir para incorporar ciertos botones a la publicación o para modificar su estética. Los navegadores *web Mozilla Firefox* y *Chrome*, por su parte, utilizan los *plugins* para la visualización de ciertas clases de contenidos. Algo similar ocurre con los reproductores de video o de audio, que pueden usar *plugins* para reproducir ciertos formatos que, en la versión original y básica del programa, no estaban

habilitados (ITESM, 2013). Las herramientas de diseño computarizado también emplean los *plugins* para agregar ciertas funcionalidades con el objetivo de ayudar en la gestión del modelado en 2D y 3D, ya sea de partículas, fluidos, renderizado o entorno de vegetación.

Algunas ventajas que ofrecen estos complementos es que facilitan la colaboración de desarrolladores externos con el *software*, además de que reducen su acoplamiento, pues eliminan y reducen relaciones innecesarias. Como resultado se obtiene un diseño donde se evidencia con mayor claridad las relaciones existentes entre los módulos. Sin embargo, una de sus desventajas es que en ocasiones puede surgir un conflicto entre un *plugin* y la aplicación principal, lo cual provoca diversos fallos en el sistema. En estos casos, por lo general, el *software* brinda la opción de desactivar el *plugin* de manera temporal o desinstalarlo.

1.6. Arquitectura de *plugins* que ofrece *QT Framework*

Como consecuencia de las ventajas que ofrecen los *plugins*, muchos *software* los utilizan para adicionar nuevas funcionalidades al sistema. Entre los *Integrated Develoment Environment* (*IDE*) que permiten la creación de *plugins* se encuentra *Qt Creator*, mediante el cual los desarrolladores pueden crear *plugins* de forma rápida e integrarlos a un sistema de forma segura, utilizando el *framework Qt*.

1.6.1. Creación de un *plugin* usando *Qt*

Qt provee dos API para la creación de *plugins*, una de alto nivel para extender *Qt* como *framework*; es decir, para adicionar funcionalidades al *framework Qt* y otra de bajo nivel, mediante la cual se pueden extender otras aplicaciones. Hacer una aplicación extensible utilizando *Qt Framework,* involucra varias etapas. En un primer momento se debe definir un conjunto de interfaces que permitirán la comunicación entre el sistema del cual se extiende y los *plugins* que se crean, para luego mediante la macro *Q_INTERFACES()*, a través de los metadatos informarle a la aplicación sobre las interfaces que fueron creadas (Corporation, 2011).

1.6.2. Leer un *plugin* desde *Qt*

En informática, una biblioteca (*library* en inglés) es un conjunto de funcionalidades, codificadas en un lenguaje de programación, la cual ofrece una interfaz bien definida para invocarlas. A diferencia de un programa ejecutable, una biblioteca no ejecuta automáticamente sus funcionalidades. Los servicios que esta proporciona pasan a formar parte del programa, o de otra biblioteca que los utilice. De esta manera se utiliza lo que ya está implementado en la biblioteca ahorrándose tiempo y recursos, además esto permite que el código y los datos se compartan y puedan modificarse de forma modular. Los *plugins* son bibliotecas, es por eso que heredan cada una de sus propiedades.

Los *plugins* se pueden guardar en una carpeta ya definida o en la carpeta de despliegue, donde se encuentra el *software* una vez ya compilado, listo para su uso. En el lugar donde estos se crean es donde se

establece su ubicación de salida, es por eso que la aplicación debe conocer la ubicación del *plugin*. El sistema puede cargar el *plugin* teniendo su dirección o la ruta de la carpeta donde se encuentra. Esta última forma es más factible cuando el proyecto que se desarrolla está pensado para utilizar varios *plugins*, ya que se encuentran organizados en una sola carpeta permitiendo su acceso de forma rápida. Una vez que se accedió al *plugin*, se intenta crear una instancia de este empleando su propia interfaz, con el objetivo de utilizar las funcionalidades que posee. *Qt* realiza el proceso de detección y carga de un *plugin* a través de la clase *QPluginLoader()*, quien verifica que el *plugin* está enlazado con la misma versión de *Qt* que la aplicación (Blanchette y Summerfield, 2006; Corporation, 2011).

1.7. Tecnologías para el desarrollo del módulo

En el proyecto DISEM ya están definidas las tecnologías con las cuales se desarrolla el *software* de diseño asistido por ordenador *AsiXMec*, es por eso que por cuestiones de compatibilidad, para dar solución al problema a resolver en el presente trabajo se hará uso de las mismas herramientas, las cuales se describen a continuación.

Lenguaje de programación C++

C++ fue diseñado a mediados de los años 1980 por Bjarne Stroustrup y fue creado para extender el lenguaje de programación C para que permitiera la manipulación de objetos. Posteriormente se añadieron facilidades de programación genérica, que se sumó a los otros dos paradigmas que ya estaban admitidos (programación estructurada y la programación orientada a objetos), por ello se suele decir que es multiparadigma. Está considerado por muchos como el lenguaje más potente, debido a que permite trabajar tanto a alto como a bajo nivel, sin embargo es a su vez uno de los que menos automatismos trae, con lo que obliga a usar bibliotecas de terceros. De los lenguajes de alto nivel, se puede decir que es uno de los más cercanos al lenguaje de máquina, lo que le proporciona mayor velocidad de ejecución con respecto a otros. Tanto C como C++ son lenguajes de programación de propósito general. Todo puede programarse con ellos, desde sistemas operativos y compiladores hasta aplicaciones de bases de datos, procesadores de texto o juegos (Stroustrup, 1999).

Entorno de desarrollo *Qt Creator*

En 1991, dos programadores noruegos (Eirik Eng y Haavard Nord de *Quasar Technologies*, más tarde *Trolltech*) crearon un *Framework* en C++ que permitía usar el mismo código en el desarrollo de una Interfaz Gráfica, tanto para *Windows* como para *UNIX/X11*. A finales de 2010, *Qt* salió en su versión 4.7 donde se añadieron nuevas características para la creación de diseños para dispositivos móviles y electrodomésticos, con la entrada de *QML* (*Qt Declarative*), un lenguaje declarativo para crear interfaces gráficas muy vistosas. *Qt* ya ha sido portado y soportado por *Nokia* para las siguientes plataformas: *Linux*/X11 (y otros sistemas

basados en *UNIX*, como *Solaris, AIX, HPUX, IRIX*), *Windows*, *Macintosh, Embedded Linux, Windows CE/-Mobile, Symbian* y *MeeGo*. Y con desarrollo externo se está portando a plataformas como *Open Solaris, Haiku* (*BeOS* libre), *OS/2* (Plataforma *eCS* de *IBM*), *iPhone* y *Android*. Como se puede apreciar, *Qt* es un *framework* que no para de crecer y el secreto de este éxito no es sólo su potencia y velocidad frente a otras alternativas como *Java*, si no porque es atractivo para el programador y rápido para desarrollar, desde la salida de *QtCreator* 2.0. Este incorpora como estilo de programación a C++, que le hace menos vulnerable a los fallos, sobre todo en el manejo de memoria (Corporation, 2011).

El *IDE* a emplear en la solución del problema planteado es *Qt Creator* en su versión 3.0.1, ya que es un entorno potente con una gran variedad de funcionalidades que facilitan su uso, además de proporcionar herramientas para diseñar y desarrollar aplicaciones con *Qt Framework*. Por otra parte, se hará uso de *Qt Framework* en su versión 5.2.1, ya que es multiplataforma y posee una gran comunidad, soporte y documentación. Es además un producto de código abierto con licencia comercial, que utiliza el lenguaje de programación C++ de forma nativa (Arcia Salazar, 2011). Actualmente la arquitectura del producto *AsiXMec* puede ser extendida a través de los *plugins* que brinda este *framework*.

Biblioteca *Open CASCADE Community Edition* (*OCE*)

OCE es un paquete de desarrollo de *software* robusto y avanzado, orientado a la visualización y modelado en 3D asistido por ordenador. Incluye un conjunto de bibliotecas implementadas en C++ que prestan servicios para la superficie 3D y modelado de sólidos, visualización, intercambio de datos y la creación rápida de aplicaciones (SAS, s.f.). Ofrece una gestión automatizada de la memoria acumulada, manejo de excepciones, clases de manipulación de agregados de datos, así como herramientas matemáticas. Igualmente suministra las estructuras de datos para representar modelos geométricos en dos y tres dimensiones; estas funcionalidades se encuentran localizadas en las clases geometría 2D, geometría 3D, utilidades geométricas y topologías (Arcia Salazar, 2011). La versión de *OCE* que se empleará es la 0.10.

1.8. Metodología de Ingeniería de *Software*

Para muchas personas un *software* es solo un programa de computadora; sin embargo son todos aquellos documentos asociados a la configuración de datos que se necesitan, para hacer que estos programas operen de manera adecuada. Estos productos de *software* se desarrollan para algún cliente en particular o para un mercado en general. Para el diseño y desarrollo de aplicaciones informáticas se aplican ciertas metodologías que permiten resolver los problemas que surgen. Las metodologías de desarrollo de *software* son un conjunto de procedimientos y técnicas que ayudan a que un *software* tenga la mejor calidad en el menor tiempo posible, con los menores costos de producción y, que a su vez, satisfaga las demandas del cliente, permitiendo además a los analistas documentar los aspectos más significativos en cada una de las etapas de su desarrollo. (Sommerville, 2005).

Existen diferentes metodologías que han servido de apoyo en los últimos años para el desarrollo de *software*, las cuales han evolucionado de manera significativa en las últimas décadas, permitiendo así el éxito o el fracaso de muchos de los sistemas desarrollados para distintas áreas. Estas pudieran tener un enfoque ágil o tradicional, y teniendo en cuenta las características que posee el producto es que son usadas.

1.8.1. Metodologías ágiles

Adaptarse a la agitada sociedad actual implica ser ágil, tener la capacidad de proveer respuestas rápidas y ser adaptables al cambio. Estas cualidades siempre han sido deseables, pero en el entorno de negocio actual resultan indispensables. Este requerimiento de agilidad en las empresas, gobiernos y cualquier otra organización provoca que el *software* también deba ser desarrollado de manera ágil (Dingsoyr et al., 2012). Las necesidades de un cliente pueden sufrir cambios importantes del momento de contratación de un *software* al momento de su entrega; y es mucho más importante satisfacer estas últimas que las primeras. Esto requiere procesos de *software* diferentes que en lugar de rechazar los cambios sean capaces de incorporarlos (Tiwari, Praveen y Manuj, 2013). Los procesos ágiles son una buena elección cuando se trabaja con requisitos desconocidos o variables. Si no existen requisitos estables, no existe una gran posibilidad de tener un diseño estable y de seguir un proceso totalmente planificado, que no vaya a variar ni en tiempo ni en dinero. En estas situaciones, un proceso adaptativo será mucho más efectivo que un proceso predictivo. Por otra parte, los procesos de desarrollo adaptativos también facilitan la generación rápida de prototipos y de versiones previas a la entrega final, lo cual agradará al cliente. Las metodologías ágiles proporcionan una serie de pautas y principios junto a técnicas pragmáticas que puede que no curen todos los males pero harán la entrega del proyecto menos complicada y más satisfactoria tanto para los clientes como para los equipos de entrega (Cendejas Valdéz, 2010).

Scrum

Scrum es un modelo de referencia que define un conjunto de prácticas y roles, que puede tomarse como punto de partida para definir el proceso de desarrollo que se ejecutará durante un proyecto. *Scrum* es el nombre con el que se denomina a los marcos de desarrollo ágiles caracterizados por adoptar una estrategia de desarrollo incremental, en lugar de la planificación y ejecución completa del producto, y en el solapamiento de las diferentes fases del desarrollo, en lugar de realizar una tras otra en un ciclo secuencial o de cascada (Canós, Letelier y Penadés, 2010). Esta metodología basa la calidad del resultado más en el conocimiento de las personas que conforman los equipos, que en la calidad de los procesos empleados. Un principio clave de *Scrum* es el reconocimiento de que durante un proyecto los clientes pueden cambiar de idea sobre lo que quieren y necesitan, y que los desafíos impredecibles no pueden ser fácilmente enfrentados de una forma predictiva y planificada. Por lo tanto, *Scrum* adopta una aproximación pragmática, aceptando que el problema no puede ser completamente entendido o definido, centrándose en maximizar la capacidad del equipo de entregar rápidamente y responder a requisitos emergentes (Dingsoyr et al., 2012).

Agile Unified Process (AUP)

Esta metodología es una versión simplificada del *Rational Unified Process* (*RUP*) usando técnicas ágiles y conceptos que aún se mantienen válidos en esta última. El proceso unificado es un marco de desarrollo de *software* iterativo e incremental, que a menudo es considerado como un proceso altamente ceremonioso porque especifica muchas actividades y artefactos involucrados en el desarrollo de un *software*. Dado que es un marco de procesos, puede ser adaptado. *AUP* se preocupa especialmente de la gestión de riesgos, proponiendo que aquellos elementos con alto riesgo obtengan prioridad en el proceso de desarrollo y se aborden en etapas iniciales. Para ello, se crean y mantienen listas identificando los riesgos desde etapas iniciales del proyecto. Especialmente relevante en este sentido es el desarrollo de prototipos ejecutables durante la base de elaboración del producto, donde se demuestre la validez de la arquitectura para los requisitos clave del producto y que determinan los riesgos técnicos (Cordero, 2010; Dingsoyr et al., 2012).

Extreme Programming (XP)

En la metodología extrema, todos los requerimientos se expresan como escenarios llamados historias de usuario, los cuales se implementan directamente como una serie de tareas. Todas las pruebas se deben ejecutar satisfactoriamente cuando el código nuevo se integra al sistema, existiendo un pequeño espacio de tiempo entre cada entrega del sistema. La programación extrema se diferencia de las metodologías tradicionales principalmente en que pone más énfasis en la adaptabilidad que en la previsibilidad (Letelier, 2006). Una de sus ventajas sobre otras metodologías, es que se centra en potenciar las relaciones entre el equipo de desarrollo, el cual debe estar integrado por cuatro personas como máximo, y el cliente; propiciando así la existencia de un entorno de trabajo agradable, lo cual es a su vez uno de los requisitos para lograr el éxito de un proyecto (Tiwari, Praveen y Manuj, 2013). Entre las características que posee se destaca el desarrollo iterativo e incremental, permitiendo pequeñas mejoras del producto; realización de pruebas unitarias y pruebas de aceptación continuas, así como la corrección de errores antes de añadir una nueva funcionalidad, permitiendo al equipo hacer entregas frecuentes (Bashir y Qureshi, 2012; Dingsoyr et al., 2012).

Consideraciones finales del capítulo

A medida que avanza el desarrollo en las diferentes esferas sociales, es evidente que las tecnologías informáticas aumentarán a la par, propiciando mejores escenarios de trabajo para la sociedad en general. Un ejemplo de estas tecnologías son los *software CAD*, encargados de brindar una mejor herramienta de diseño computarizado a profesionales y aficionados. Estas, a su vez, emplean en común varias técnicas para posicionar un objeto de forma precisa, como las guías, las restricciones, la cuadrícula o los *snap*; estos últimos muy empleados por los usuarios por las facilidades que provee. Sin embargo, no solo estas características complacen a los usuarios finales, sino otros atributos de calidad como son el rendimiento de la aplicación, aspecto que se puede medir entre otras cosas, por el acoplamiento que puedan tener las unidades que la conforman.

En el presente trabajo se desarrollará la segunda versión del módulo *Snap* empleando la metodología *XP*. De igual forma luego de haber identificado varias de las técnicas que permiten un menor grado de acople en un *software*, se hará uso de los *plugins* para dar respuesta al problema planteado inicialmente, todo esto con el objetivo de mejorar la calidad de la herramienta *AsiXMec*.

CAPÍTULO 2

Propuesta de solución

Introducción

En este capítulo se describen los elementos necesarios para la construcción de la solución empleando la técnica para reducir el acoplamiento en un *software* basada en los *plugins*. De igual forma se documenta los detalles más significativos de la Ingeniería de *Software* utilizando la metodología *XP*, como son la captura de requisitos, el tiempo estimado que debe durar el proyecto basado en las tareas de desarrollo y la arquitectura que sostendrá al *software* en cuestión.

2.1. Propuesta de solución

Para dar solución al problema de investigación planteado se propone desarrollar una segunda versión del módulo *Snap* basada en *plugins*, una técnica muy empleada en la actualidad para reducir el acoplamiento en un *software* que permite además adicionar nuevas funcionalidades a un sistema informático. En el desarrollo de dicha solución se hará uso de la arquitectura de *plugins* que brinda *Qt Framework*, donde mediante las macros que emplea para el desarrollo de interfaces, se realizará la conversión a *plugins* de cada uno de los *snap* que contiene actualmente el módulo, proceso muy importante en el desarrollo de la aplicación ya que de ello dependen las funcionalidades que debe permitir el sistema posteriormente. Para ello se empleará la API de alto nivel que provee el *framework*, ya que permite extender de otras aplicaciones.

Una vez culminado este proceso se deben integrar con el *software AsiXMec*, teniendo en cuenta los restantes módulos con los cuales está estrechamente vinculado, permitiendo cargar los complementos dinámicamente cuando el *software* inicialice. El sistema debe mostrar una interfaz gráfica con la cual el usuario puede interactuar, compuesta por varios *checkbox* que contienen solo aquellos *plugins* que se pudieron cargar una vez ejecutada la aplicación por un usuario. Este último debe tener la posibilidad de activar o desactivar, uno o varios *plugins* a la misma vez, dependiendo de los que desee que estén habilitados cuando se encuentre diseñando una estructura mecánica.

Con esta propuesta de solución se espera que en un futuro si se agregan al sistema nuevos *snaps* o se realizan modificaciones en los *snaps* existentes, no sea necesario recompilar la aplicación completa, solo el *snap* en cuestión, de esta forma ya no se afectarán aquellos módulos con los cuales se relaciona. Además, le brinda la opción al usuario de controlar la activación de los *plugins* al ejecutar el *software*, ya que puede decidir cuáles son los que desea activar o desactivar en un momento dado. Todo lo anterior permite que los módulos que conforman la aplicación tengan un menor acoplamiento y un mayor rendimiento del *software* en tiempo de ejecución.

2.2. Ingeniería de *Software*

La nueva versión del módulo debe contar con un conjunto de requerimientos funcionales y no funcionales que lo conviertan en el producto esperado, los cuales se enmarcan en los procesos que se describieron con anterioridad. Para la captura de estos requisitos es imprescindible hacer uso de la Ingeniería de *Software*, aplicando cada una de las etapas por las cuales transcurre la metodología seleccionada *XP*.

2.2.1. Etapa de planificación

XP plantea la planificación como un permanente diálogo entre la parte empresarial y técnica del proyecto, en la que los primeros decidirán el alcance que tiene el producto, los requisitos funcionales y no funcionales, la prioridad que tiene cada uno de los procesos que intervienen en su desarrollo así como la velocidad a la cual debieran ir, las fechas para las cuales se pueden entregar las posibles versiones que tendrá el *software*, y los posibles riesgos que pudieran ocurrir durante su creación (Rodríguez Corbea y Ordoñez Pérez, 2007).

Una de las primeras tareas del equipo de desarrollo es realizar la captura de las funcionalidades y atributos de calidad que tendrá el sistema, cuya comprensión está entre las tareas más complicadas que pude enfrentar un ingeniero informático; sin embargo les ayuda a entender el problema en cuya solución se trabajará. Esto permitirá que el proceso de desarrollo del *software* vaya en una dirección ascendente, hasta llegar al producto esperado (S. R. Pressman, 1992).

Requisitos funcionales

Los requerimientos funcionales son declaraciones formales de las prestaciones que proporciona el sistema. Estas deben argumentar la manera en que la aplicación puede reaccionar a las entradas y cómo se debe comportar en situaciones particulares (Sommerville, 2005). El proceso de levantamiento de requisitos concluyó con un total de 5 requisitos, los cuales se enumeran a continuación:

RF 1. Convertir los *snap* a *plugins*.

RF 2. Integrar los *plugins* con el *software AsiXMec*.

RF 3. Cargar los *plugins*.

RF 4. Mostrar *plugins* cargados.

RF 5. Controlar activación de los *plugins.*

Requisitos no funcionales

Los requisitos no funcionales representan restricciones a los servicios que debe brindar el *software* (Sommerville, 2005). A continuación se establecen los requisitos no funcionales establecidos para el desarrollo de la aplicación.

RnF 1. Usabilidad

- El sistema debe ser desarrollado para plataformas de escritorio, basándose en la plataforma DISEM, sobre la cual trabaja el centro de desarrollo VERTEX.
- Interfaz de usuario amigable, que siga las pautas de diseño de *AsiXMec*.

RnF 2. Soporte

- Debe seguirse el estándar de codificación definido en el proyecto DISEM para un fácil entendimiento del código de la aplicación, guiado al desarrollo de futuras versiones.

RnF 3. *Software*

- Sistema operativo: Linux 14.04
- Biblioteca: *OCE* 0.10

2.2.2. Descripción de historias de usuario

Las historias de usuarios describen las características y las funcionalidades requeridas para la nueva versión del módulo que se construye. Cada historia la describe el cliente y su formato es muy similar a una carta índice. Las historias de usuario son dinámicas, ya que durante el desarrollo de la aplicación pueden cambiar constantemente el valor de prioridad de cada historia; así como su fecha de entrega. Las estimaciones de esfuerzo asociado a la implementación de las historias la establecen los programadores utilizando como medida el punto, el cual equivale a una semana ideal de programación. Las historias generalmente valen de 1 a 3 puntos (Rodríguez Corbea y Ordoñez Pérez, 2007).

Tabla 2.1. Historia de usuario # 1

Historia de usuario	
Número: 1	**Nombre:** Convertir los *snap* a *plugins*
Usuario: Desarrollador	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta

Continúa en la próxima página

Tabla 2.1. Continuación de la página anterior

Puntos estimados: 2	**Iteración asignada:** 1
Programador responsable: Midiala Baños Artigas	
Descripción: Cada uno de los *snap* que contiene el módulo deben ser llevados a *plugins* haciendo uso de las macros que ofrece *Qt Framework.*	
Observaciones:	

Tabla 2.2. Historia de usuario # 2

Historia de usuario	
Número: 2	**Nombre:** Integrar los *plugins* con el *software AsiXMec*
Usuario: Desarrollador	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta
Puntos estimados: 2	**Iteración asignada:** 1
Programador responsable: Midiala Baños Artigas	
Descripción: Culminado el desarrollo de los *plugins*, deben integrarse con el *software AsiXMec* para un correcto funcionamiento.	
Observaciones:	

Tabla 2.3. Historia de usuario # 3

Historia de usuario	
Número: 3	**Nombre:** Cargar los *plugin*
Usuario: Desarrollador	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta
Puntos estimados: 2	**Iteración asignada:** 2
Programador responsable: Midiala Baños Artigas	
Descripción: Debe permitir que una vez que el *software AsiXMec* sea ejecutado por el usuario, se carguen dinámicamente cada uno de los *plugins* que contiene el módulo.	
Observaciones:	

Tabla 2.4. Historia de usuario # 4

Historia de usuario	
Número: 4	**Nombre:** Mostrar *plugins* cargados
Usuario: Desarrollador	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Media
Puntos estimados: 1.5	**Iteración asignada:** 2
Programador responsable: Midiala Baños Artigas	

Continúa en la próxima página

Tabla 2.4. Continuación de la página anterior

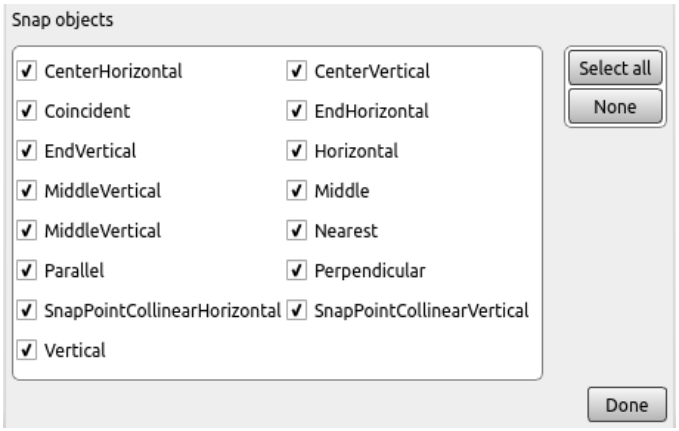
Descripción: Una vez que se han cargado los *plugins* al inicializar el sistema, estos deben ser mostrados en un menú, en el que cada *plugin* estará contenido en un *checkbox*.
Observaciones: Aquellos *plugins* que no se han podido cargar no se podrán mostrar en la interfaz.
Interfaz: 

Tabla 2.5. Historia de usuario # 5

Historia de usuario	
Número: 5	**Nombre:** Controlar la activación de los *plugins*
Usuario: Desarrollador	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Media
Puntos estimados: 1.5	**Iteración asignada:** 3
Programador responsable: Midiala Baños Artigas	
Descripción: Cuando el sistema muestre el menú de *plugins*, el usuario podrá habilitar o deshabilitar aquellos con los cuales necesite trabajar.	
Observaciones: Aquellos *plugins* que no se han podido cargar no podrán ser seleccionados por un usuario, ya que no aparecen en el menú.	

2.2.3. Tareas de desarrollo

Cada historia de usuario se transformará en tareas de desarrollo, las cuales corresponderán a un período ideal de una a tres semanas de desarrollo (Rodríguez Corbea y Ordoñez Pérez, 2007).

Tabla 2.6. Tarea de ingeniería # 1

Tarea	
Número de tarea: 1	**Número de historia de usuario:** 1
Nombre de la tarea: Convertir los *snap* a *plugin*	
Tipo de tarea: Desarrollo	**Puntos estimados:** 2
Fecha de inicio: 2 de febrero de 2015	**Fecha de fin:** 13 de febrero de 2015
Programador responsable: Midiala Baños Artigas	
Descripción: Se debe llevar a *plugin* cada *snap* que contiene el módulo, haciendo uso de las macros *Q_DECLARE_INTERFACE(ClassName, Identifier), Q_IMPORT_PLUGIN(PluginName) y Q_PLUGIN_METADATA()*, que ofrece *Qt Framework* para el desarrollo de *plugins*.	

Tabla 2.7. Tarea de ingeniería # 2

Tarea	
Número de tarea: 2	**Número de historia de usuario:** 2
Nombre de la tarea: Integrar los *plugins* con el *software AsiXMec*	
Tipo de tarea: Desarrollo	**Puntos estimados:** 2
Fecha de inicio: 16 de febrero de 2015	**Fecha de fin:** 27 de febrero de 2015
Programador responsable: Midiala Baños Artigas	
Descripción: Una vez desarrollados los *plugins*, es necesario modificar el funcionamiento del módulo e integrarlo a la herramienta. Con tal propósito se deben incluir algunas funciones en la clase *plugin-manager*, para que así pueda cargarlos.	

Tabla 2.8. Tarea de ingeniería # 3

Tarea	
Número de tarea: 3	**Número de historia de usuario:** 3
Nombre de la tarea: Cargar los *plugin*	
Tipo de tarea: Desarrollo	**Puntos estimados:** 2
Fecha de inicio: 2 de febrero de 2015	**Fecha de fin:** 20 de marzo de 2015
Programador responsable: Midiala Baños Artigas	
Descripción: Empleando la clase *QPluginLoader()* que provee *Qt Framework*, se cargan automáticamente los *plugins* desarrollados.	

Tabla 2.9. Tarea de ingeniería # 4

Tarea	
Número de tarea: 4	**Número de historia de usuario:** 4
Nombre de la tarea: Mostrar los *plugins*	
Tipo de tarea: Desarrollo	**Puntos estimados:** 2
Fecha de inicio: 23 de marzo de 2015	**Fecha de fin:** 2 de abril de 2015
Programador responsable: Midiala Baños Artigas	
Descripción: Los *plugins* que se pudieron cargar, deben ser visualizados en una interfaz que los contiene.	

Tabla 2.10. Tarea de ingeniería # 5

Tarea	
Número de tarea: 5	**Número de historia de usuario:** 5
Nombre de la tarea: Activar o desactivar los *plugins*	
Tipo de tarea: Desarrollo	**Puntos estimados:** 2
Fecha de inicio: 13 de abril de 2015	**Fecha de fin:** 24 de abril de 2015
Programador responsable: Midiala Baños Artigas	
Descripción: Debe permitir que un usuario pueda seleccionar del menú de *plugins* que se muestra en la interfaz, aquellos que desee que estén habilitados, y de igual forma poder deshabilitarlos si así lo desea.	

2.2.4. Estimación de esfuerzo por historias de usuarios

Tabla 2.11. Estimación de esfuerzo por historia de usuario

Iteración	**Historias de usuario**		**Puntos estimados (semanas)**
1	1	Convertir los snap a plugins	2.0
	2	Integrar los plugins con el software AsiXMec	2.0
2	3	Cargar los plugins	2.0
	4	Mostrar plugins cargados	1.5
3	5	Controlar la activación de los plugins	1.5
Total			**9.0**

2.2.5. Plan estimado de entregas

Las historias de usuario servirán para crear el plan estimado de entregas. El plan de entregas se usará para crear los planes de iteración para cada una de las iteraciones del proyecto. Con cada historia de usuario previamente evaluada en tiempo de desarrollo ideal, el cliente las agrupará en orden de importancia. Una semana ideal es cuánto tiempo costaría implementar dicha historia si no se tiene nada más que hacer, incluyendo la parte de los *tests* correspondientes. De esta forma se puede trazar el plan de entregas en función de estos dos

parámetros: tiempo de desarrollo ideal y grado de importancia para el cliente. Las iteraciones individuales son planificadas en detalle justo antes de que comience cada iteración (Rodríguez Corbea y Ordoñez Pérez, 2007).

Una entrega debería obtenerse en no más de tres meses. Esta fase dura unos pocos días. El equipo de desarrollo mantiene un registro de la velocidad de desarrollo, establecida en puntos por iteración, basándose principalmente en la suma de puntos correspondientes a las historias de usuario que fueron terminadas en la última iteración (Echeverry, Miguel y Delgado Carmona, 2007).

Tabla 2.12. Plan de entregas

Iteración	Entrega	Fecha
1	Versión 0.2	27 de febrero del 2015
2	Versión 0.8	2 de abril del 2015
3	Versión 1.0	24 de abril del 2015

2.2.6. Plan de duración de las iteraciones

Cada iteración corresponde a un período de tiempo de desarrollo del proyecto de entre una y tres semanas. De esta forma, un proyecto se divide en una docena de iteraciones aproximadamente. Se usará la velocidad del proyecto para determinar si una iteración está sobrecargada. La suma de los días que costará desarrollar todas las tareas de la iteración no debería sobrepasar la velocidad del proyecto de la iteración anterior. Si la iteración está sobrecargada, el cliente deberá decidir que historias de usuario retrasar a una iteración posterior. Si, por el contrario, la iteración tiene huecos se rellenará con otras historias de usuario (Rodríguez Corbea y Ordoñez Pérez, 2007).

- **Iteración 1:** Tiene como objetivo darle cumplimiento a las historias de usuario 1 y 2, consideradas de gran importancia para el desarrollo de la aplicación, ya que permiten la creación de los *plugins* y su integración con el sistema *AsiXMec* respectivamente.

- **Iteración 2:** Tiene como objetivo darle cumplimiento a las historias de usuario 3 y 4, la cuales son dos importantes funcionalidades que debe permitir el módulo una vez culminado. Con su culminación el sistema debe ser capaz de cargar los *plugins* que contiene el módulo, y poder mostrarlos a través de una interfaz.

- **Iteración 3:** Tiene como objetivo darle cumplimiento a la historia de usuario 5, la cual permite que un usuario pueda activar o desactivar uno a varios de los *plugins* que se muestran en la interfaz, siendo este uno de los resultados finales más importantes que se espera de la aplicación en cuestión.

Tabla 2.13. Plan de duración de las iteraciones

Iteración		Historias de usuario	Duración (semanas)
1	1	Convertir los snap a plugins	4.0
	2	Integrar los plugins con el software AsiXMec	
2	3	Cargar los plugins	3.5
	4	Mostrar plugins cargados	
3	5	Controlar la activación de los plugins	1.5
Total			**9.0**

2.3. Etapa de diseño

XP establece unas recomendaciones o premisas a la hora de abordar esta etapa, como es el caso de la simplicidad. Siempre cuesta menos tiempo implementar un diseño sencillo que uno complejo, por lo que hay que tratar siempre de realizar las cosas de la manera más sencilla posible. Si alguna parte de la implementación resulta especialmente compleja, esta debe replantearse; así, cualquier modificación será mucho más sencilla de realizar (Rodríguez Corbea y Ordoñez Pérez, 2007).

2.3.1. Tarjetas CRC

Para poder diseñar el sistema como un equipo se debe cumplir con tres principios: Cargo o Clase, Responsabilidad y Colaboración. Las tarjetas CRC permiten desprenderse del método de trabajo basado en procedimientos y trabajar con una metodología basada en objetos. Posibilitan además que el equipo completo contribuya en la tarea del diseño (Dingsoyr et al., 2012).

Tabla 2.14. Tarjeta CRC # 1

Tarjeta CRC	
Clase: SnapPluginInterface	
Responsabilidad	**Colaboración**
• Es la clase encargada de crear la interfaz de la cual heredará cada *plugin*.	PluginManager PluginManifest PluginInterface

Tabla 2.15. Tarjeta CRC # 2

Tarjeta CRC	
Clase: SnapManager	
Responsabilidad	**Colaboración**
• Es la clase encargada de cargar, mostrar y ejecutar cada *plugin*.	PluginManager SnapCenterHorizontalPlugin SnapCenterVerticalPlugin SnapCoincidentPlugin SnapEndHorizontalPlugin SnapEndVerticalPlugin SnapHorizontalPlugin SnapMiddlePlugin SnapMiddleHorizontalPlugin SnapMiddleVerticalPlugin SnapNearestPlugin SnapParallelPlugin SnapPerpendicularPlugin SnapPointCollinearHorizontalPlugin SnapPointCollinearVerticalPlugin SnapVerticalPlugin

Tabla 2.16. Tarjeta CRC # 3

Tarjeta CRC	
Clase: SnapSettingDialog	
Responsabilidad	**Colaboración**
• Es la clase encargada de mostrar una interfaz donde se muestran todos los *plugins* cargados, para que el usuario seleccione aquellos que desea que estén activados o desactivados.	SnapManager SnapSetting

2.3.2. Patrones de diseño

Un patrón de diseño es una descripción de clases y objetos comunicándose entre sí adaptada para resolver un problema de diseño general en un contexto particular. Estos se caracterizan por su efectividad, la cual ha sido comprobada resolviendo problemas similares en ocasiones anteriores, así como por su capacidad para ser reutilizables, ya que pueden ser aplicados a diferentes problemas de diseño en distintas circunstancias (Gamma et al., 2008). A continuación se exponen los patrones utilizados en el módulo desarrollado.

Los patrones GRASP representan los principios básicos de la asignación de responsabilidades a objetos, expresados en forma de patrones. Estos tipifican buenos principios de programación, cuya eficacia ha sido demostrada con anterioridad en escenarios similares.

- **Creador:**

  La creación de objetos es una de las actividades más frecuentes en un sistema orientado a objetos. Este patrón se utiliza cuando la clase contiene los datos del objeto que desea crear (Larman, 1999). Un ejemplo en el módulo donde se evidencie es la clase *PluginManager*, la cual contiene los datos de los *plugins* que se van a cargar.

- **Experto:**

  Este patrón se utiliza con el objetivo de asignar una responsabilidad a la clase que posee toda la información necesaria para realizarla (ibíd.). Es el patrón más utilizado debido a su esencia, por lo cual es empleado en todas las clases que contiene el módulo.

- **Alta cohesión:**

  La cohesión es una medida de cuan relacionadas y enfocadas están las responsabilidades de una clase. Por tanto, una alta cohesión promueve la creación de clases con responsabilidades bien definidas y que estén estrechamente relacionadas (ibíd.). Este patrón se evidencia en las relaciones entre las clases implementadas para la realización del módulo, ya que están organizadas de forma tal, que cada clase solo tiene conocimiento de las funciones que esta realiza.

- **Bajo acoplamiento:**

  Este patrón se encarga de que existan las menores relaciones posibles entre las clases que integran a un sistema informático, evitando así que existan dependencias innecesarias entre ellas (ibíd.). Este patrón se evidencia en las relaciones entre las clases implementadas para la realización del módulo, ya que cada clase se relaciona solo con aquellas de las cuales se necesita información. Así mismo, el módulo desarrollado tiene un menor grado de acoplamiento con los restantes módulos de la herramienta *AsiXMec*.

Los patrones GoF se descubren como una forma indispensable de enfrentarse a la programación a raíz del libro *Design Patterns Elements of Reusable Software* de Erich Gamma, Richard Helm, Ralph Jonson y John Vlissides. A partir de entonces estos patrones son conocidos como los patrones de la pandilla de los cuatro. El patrón Gand of Four (GoF) empleado en el desarrollo del sistema, es el que se describe a continuación.

- **Fachada:**

  Este patrón es empleado para facilitar la comunicación entre objetos, proporcionando una interfaz de la cual heredan otras clases (Freeman et al., 2005). Un ejemplo de ello en el módulo es la clase *SnapPluginInterface*, de la cual heredan los *snaps*.

2.3.3. Arquitectura

La arquitectura de *software* forma la columna vertebral para construir un sistema informático, y es en gran medida responsable de permitir o no ciertos atributos de calidad del sistema entre los que se destacan la confiabilidad y el rendimiento del *software*. Además, es un modelo abstracto reutilizable que puede transferirse de un sistema a otro y que representa un medio de comunicación y discusión entre participantes del proyecto, permitiendo así la interacción e intercambio entre los desarrolladores con el objetivo final de establecer el intercambio de conocimientos y puntos de vista entre ellos (R. S. Pressman, 1997).

Para el desarrollo de la aplicación se escogió la arquitectura basada en capas, a través de la cual se crean 2 capas donde cada una realiza operaciones que progresivamente se aproximan más al cuadro de instrucciones de la máquina. En la capa de presentación se encuentran las clases que sirven a las operaciones de interfaz de usuario, mientras que en la capa lógica se encuentran las clases que realizan operaciones de interfaz del sistema.

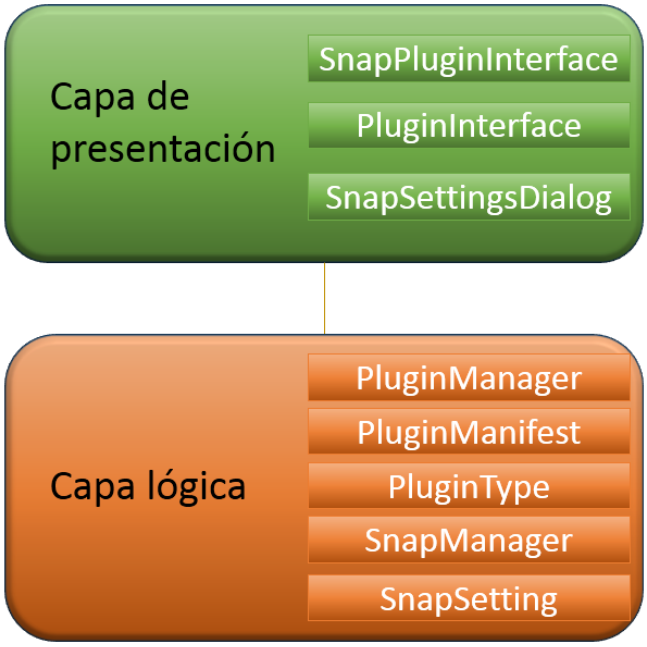


Figura 2.1. Arquitectura en capas del módulo *Snap*.

Capa de presentación

- *SnapPluginInterface*: Esta clase brinda una interfaz de la cual heredan todos los *snaps*, ofreciendo las funciones necesarias para calcular si es posible realizar el *snap*, cómo se debe mostrar y cómo finalmente se hará.
- *PluginInterface*: Esta clase ofrece una interfaz de la cual heredan todas las interfaces de los *plugins*, brindándole a la aplicación *AsiXMec* un mecanismo estándar para el manejo de sus *plugins*.

- *SnapSettingsDialog*: Esta clase muestra una interfaz que permite la activación y desactivación de los snaps que fueron cargados.

Capa lógica

- *PluginManifest*: Es la clase encargada de capturar los datos que se encuentran en el archivo xml de cada *plugin*, los cuales le permiten que se carguen. Estos datos son el tipo de *plugin*, el nombre y una descripción de cada uno.
- *PluginManager*: Es la clase encargada de cargar cada *plugin*, una vez que haya identificado su tipo. Asigna a cada uno la interfaz mediante la cual se podrá interactuar con él.
- *PluginType*: Esta clase contiene los tipos de *plugins*.
- *SnapSetting*: Es la clase encargada de gestionar la activación y desactivación de los *plugins*.
- *SnapManager*: Esta clase es la encargada de obtener todos los *snaps* que fueron cargados, con el objetivo de gestionarlos. Se ejecuta cuando se crea una entidad en el *sketcher*.

Se emplea esta arquitectura ya que facilita la migración, pues el acoplamiento con el entorno está localizado en las capas inferiores. Estas son las únicas a reimplementar en caso de migrar a un entorno diferente. Permite además la reutilización, pues como cada nivel implementa interfaces claras y lógicas, pueden intercambiarse; además del mantenimiento del sistema, pues los cambios en una capa apenas afectan a la superior e inferior. Para implementar los niveles superiores no se requiere conocer el entorno subyacente, basta con las interfaces que proporcionan los niveles inferiores.

Consideraciones finales del capítulo

Una vez analizados en el marco teórico los principales elementos que intervienen en el problema de investigación, se propuso una solución basada en *plugins* para disminuir el acoplamiento del software *AsiX-Mec*. Se realizó un levantamiento de las historias de usuario que intervienen además de la duración de cada una de ellas. Concluidos los procesos de la etapa de planificación, se procedió a la etapa de diseño, donde se estableció la arquitectura que tendría el *software* y las tarjetas CRC, todo esto con el objetivo de tener un mayor control y conocimiento sobre el desarrollo del sistema.

CAPÍTULO 3

Resultados

Introducción

En el presente capítulo se documentarán los principales elementos de la etapa de implementación y pruebas del *software*, siguiendo las pautas que establece la metodología seleccionada *XP*. Se describen los estándares por los cuales se debe guiar el desarrollador en cuestión durante el desarrollo del *software*, así como las pruebas de aceptación realizadas con el objetivo de verificar el cumplimiento eficaz de cada requisito funcional del sistema.

3.1. Estándares de implementación

Los estándares permiten que el código que se desarrolló sea de una mejor calidad, que el mantenimiento de los programas se vuelva menos complejo, y provoca que baje la tasa de errores ingenuos. A continuación se muestra el estándar de implementación utilizado para la realización del módulo.

- **Estilos de comentarios**

  El estilo de los comentarios debe ser como el estilo de comentarios para C (/* */ o //), no debe de utilizarse el estilo de comentarios de Perl (#).

- **Indentación**

  La indentación será de cuatro espacios, utilizándose la tabulación. No poseerán indentación las llaves asociadas a los espacios de nombre, definición de clases o implementación de métodos.

- **Declaración de clases**

  Los nombres de las clases deben de iniciar con letra mayúscula, y se debe evitar el uso del guion bajo en los casos que un nombre sea compuesto.

- **Declaración de variables**

  Se declarará una variable por línea. Además se evitará nombrar a las variables con abreviatura siempre que sea posible. Todos los caracteres serán escritos en minúscula, excepto en caso de que existan nombres compuestos. En este caso, se escribirá con minúscula el primer nombre y los restantes con letra inicial mayúscula, evitando usar el guión bajo.

- **Declaración de métodos**

  Siguen el mismo convenio de las variables.

Código fuente 3.1. Ejemplo del estándar de codificación

```
SnapSettingDialog::SnapSettingDialog(QWidget *parent) :
    QDialog(parent)

{
    ui.setupUi(this);
    vector<string> snapIds = settings.getSnapIds();
    int currentRow = 0;                                    //para saber en qué
        fila se pondrá el checkBox
    for(unsigned i = 0; i < snapIds.size(); ++i) {
        QString snapId = QString().fromStdString(snapIds[i]);
        QCheckBox * checkBox = new QCheckBox(snapId);
                              //el check box se muestra con el estado
                                 actual de los snap (activo o no)
        bool state = settings.isEnabledSnap(snapId.toStdString());
        checkBox->setChecked(state);


        if(i %2 == 0) {                                    //para saber la
           columna, si es par a la izquierda, sino a la derecha
            ((QGridLayout*)ui.groupBoxSnap->layout())->addWidget(
                checkBox, currentRow, 0);
        } else {                                           //si va a la
           derecha el próximo debe estar en otra fila
            ((QGridLayout*)ui.groupBoxSnap->layout())->addWidget(
                checkBox, currentRow++, 1);
        }
    }
}
```

3.2. Etapa de pruebas

Uno de los pilares de esta metodología es el proceso de pruebas. *XP* anima a probar constantemente tanto como sea posible. Esto permite aumentar la calidad de los sistemas reduciendo el número de errores no detectados y disminuyendo el tiempo transcurrido entre la aparición de un error y su detección. También permite aumentar la seguridad de evitar efectos colaterales no deseados a la hora de realizar modificaciones y refactorizaciones.

Las pruebas son un conjunto de actividades que se pueden planificar por adelantado y llevar a cabo sistemáticamente. Por esta razón se debe definir en el proceso de la ingeniería del *software*. Todo esto contribuye a elevar la calidad de los productos desarrollados y a la seguridad de los programadores a la hora de introducir cambios o modificaciones. Las pruebas que se realizan a la aplicación son las pruebas de aceptación o pruebas funcionales, destinadas a evaluar si al final de una iteración se consiguió la funcionalidad requerida diseñadas por el cliente final (Loret de Mola, 2013).

3.2.1. Pruebas de aceptación o pruebas del sistema

Las pruebas del sistema tienen como objetivo verificar la funcionalidad del sistema a través de sus interfaces externas comprobando que dicha funcionalidad sea la esperada en función de los requisitos del sistema. Generalmente las pruebas del sistema son desarrolladas por los programadores para verificar que su sistema se comporta de la manera esperada, por lo que podrían encajar dentro de la definición de pruebas unitarias que propone *XP*. Sin embargo, las pruebas del sistema tienen como objetivo verificar que el sistema cumple los requisitos establecidos por el usuario por lo que también pueden encajar dentro de la categoría de pruebas de aceptación (Rodríguez Corbea y Ordoñez Pérez, 2007).

Las pruebas realizadas al *software* constituyen su única garantía de calidad, las mismas pueden estar dirigidas a los componentes de forma individual o al sistema como un conjunto. Siguiendo la línea de la metodología, estas pruebas están dirigidas por cada historia de usuario, que se traducen a su vez en casos de prueba. Durante una iteración la historia de usuario seleccionada en la planificación de iteraciones se convertirá en una prueba de aceptación (Bashir y Qureshi, 2012).

Tabla 3.1. Prueba de aceptación # 1

Caso de prueba de aceptación	
Código: HU1_P1	**Historia de usuario:** 1
Nombre: Convertir los *snaps* a *plugins*	
Descripción: Con el objetivo de probar que cada uno de los *snap* ha sido convertido a *plugin* de manera satisfactoria, se debe compilar el proyecto *AsiXMec*, para determinar si existen o no posibles errores en los *plugins* creados que se encuentran dentro del módulo *snap-integration*.	
Condiciones de ejecución: Debe existir al menos un *plugin* creado.	
Pasos de ejecución: • Abrir el proyecto *AsiXMec*. • Compilar el proyecto.	
Resultados esperados: Debe mostrarse la herramienta *AsiXMec* una vez ejecutada, sin haberse mostrado errores durante el proceso de compilación.	

Tabla 3.2. Prueba de aceptación # 2

Caso de prueba de aceptación	
Código: HU2_P2	**Historia de usuario:** 2
Nombre: Integrar el módulo con el proyecto *AsiXMec*	
Descripción: Con el objetivo de probar si el módulo en desarrollo se integra al proyecto *AsiXMec*, es necesario compilar el *software* para determinar si existen errores en los módulos que se relacionan con el módulo en cuestión.	
Condiciones de ejecución: Deben estar creados todos los *plugins* que contendrá el módulo.	
Pasos de ejecución: • Abrir el proyecto *AsiXMec*. • Compilar el proyecto.	
Resultados esperados: Debe mostrarse la herramienta *AsiXMec* una vez ejecutada, sin haberse mostrado errores durante el proceso de compilación.	

Tabla 3.3. Prueba de aceptación # 3

Caso de prueba de aceptación	
Código: HU3_P3	**Historia de usuario:** 3
Nombre: Cargar los *plugins*.	
Descripción: Con el objetivo de probar si el proyecto carga los *plugins* creados, una vez compilado el sistema es necesario ir a la carpeta donde estos se guardan y comprobar si los *plugins* que contiene el módulo se encuentran allí.	
Condiciones de ejecución: Los *plugins* deben existir en el directorio **../../asixmec/snap-integration/snap-plugins**.	
Pasos de ejecución: • Abrir el proyecto *AsiXMec*. • Compilar el proyecto. • Verificar la existencia de los *plugins* en el directorio **../../builds-cdsem/build/plugins/**.	
Resultados esperados: Todos los *plugins* que posee el módulo deben estar contenidos una vez compilada la aplicación en el directorio esperado. De igual manera deben mostrarse en la pestaña ***Plugins*** de la herramienta.	

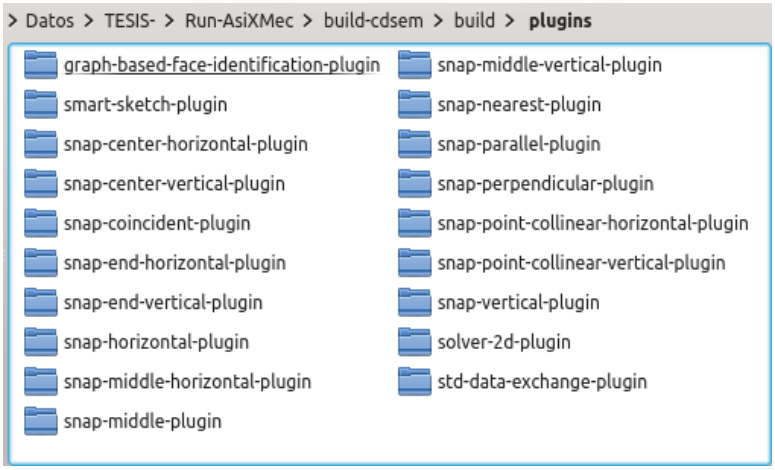


Figura 3.1. Directorio donde se encuentran los *plugins*.

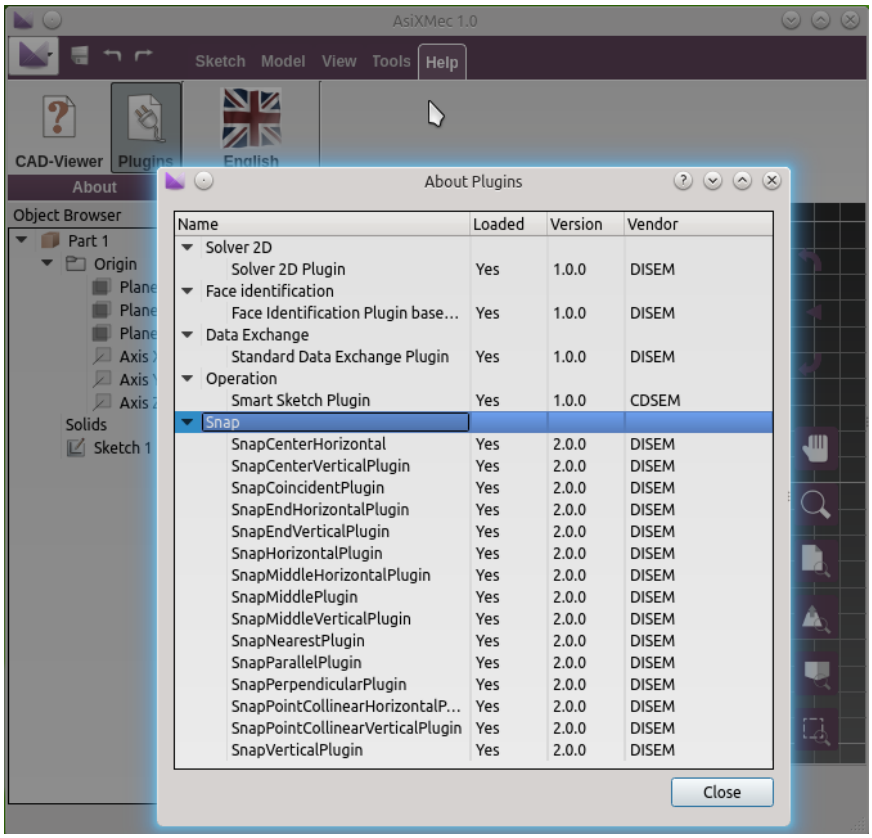


Figura 3.2. *Plugins* cargados.

Tabla 3.4. Prueba de aceptación # 4

Caso de prueba de aceptación	
Código: HU4_P4	**Historia de usuario:** 4
Nombre: Mostrar todos los *plugins* que contiene el módulo.	
Descripción: Una vez que el proyecto haya sido compilado, todos los *plugins* que han sido cargados deben mostrarse en una interfaz.	
Condiciones de ejecución: • Abrir el proyecto *AsiXMec*. • Compilar el proyecto *AsiXMec*. • Ir a la ventana de los *snaps*. • Verificar que se muestran todos los *plugins*.	
Pasos de ejecución: Debe existir al menos un *plugin* cargado	
Resultados esperados: Deben mostrarse en la pestaña ***Snaps*** todos los *plugins* que han sido cargados.	

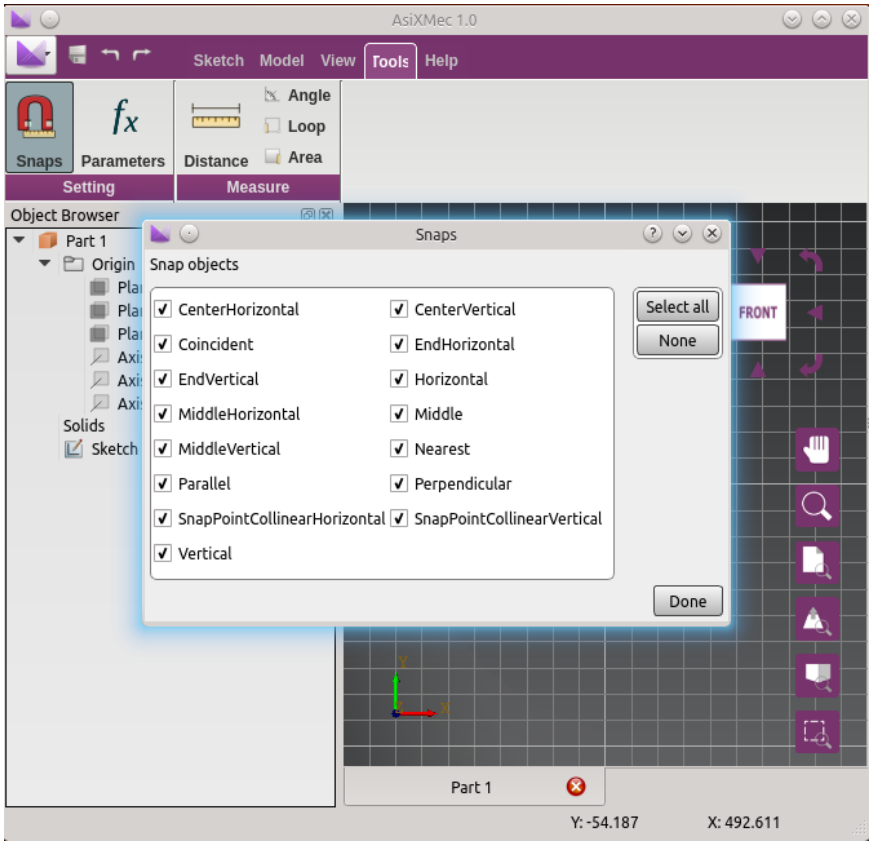


Figura 3.3. Interfaz visual donde se muestran los *plugins* una vez cargados.

Tabla 3.5. Prueba de aceptación # 5

Caso de prueba de aceptación	
Código: HU4_P5	**Historia de usuario:** 5
Nombre: Mostrar solo algunos *plugins*.	
Descripción: Se extraen de la ruta */media/Datos/TESIS/Run-AsiXMec/build-cdsem/build/plugins/* varios *plugins* aleatoriamente, y se ubican en una carpeta de prueba.	
Condiciones de ejecución:	

Continúa en la próxima página

Tabla 3.5. Continuación de la página anterior

Pasos de ejecución: • Ir al directorio *../../build-cdsem/build/plugins/*. • Ubicar los *plugins* seleccionados en una carpeta de prueba. • Eliminar los *plugins* seleccionados. • Recompilar el proyecto *AsiXMec*. • Ir a la ventana de los *snaps*. • Verificar que solo se muestran los *plugins* que no fueron extraídos.
Resultados esperados: Deben mostrarse solo los *plugins* que quedaron en el directorio antes mencionado.

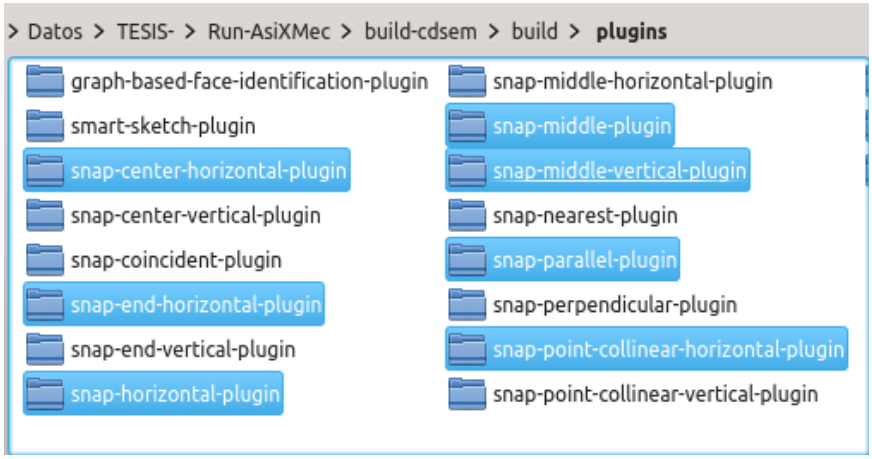


Figura 3.4. *Plugins* extraídos para realizar la prueba.

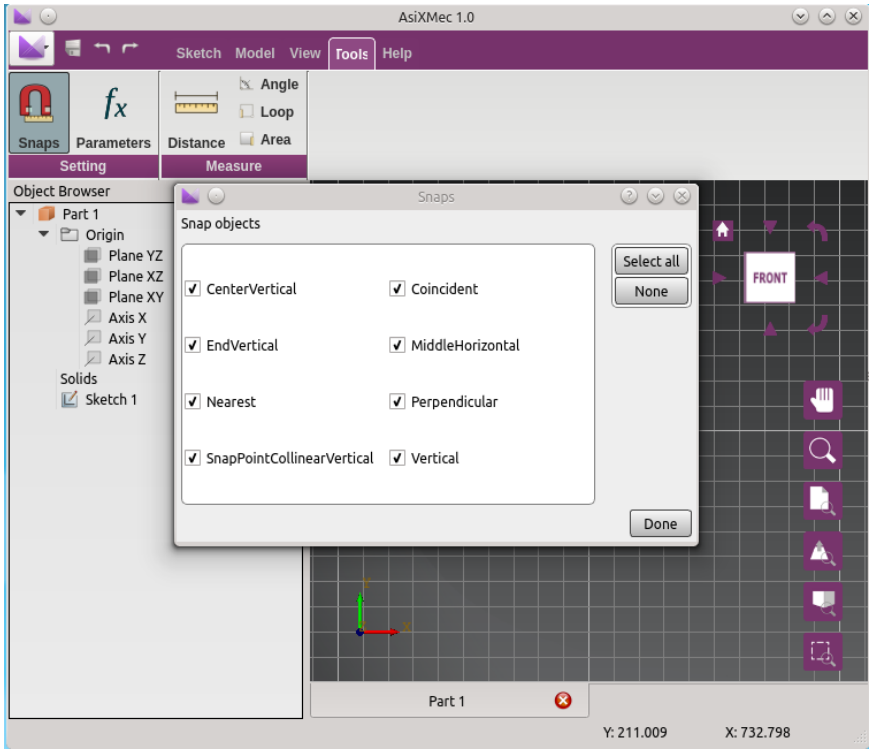


Figura 3.5. *Plugins* que fueron cargados.

Tabla 3.6. Prueba de aceptación # 6

Caso de prueba de aceptación	
Código: HU5_P6	**Historia de usuario:** 6
Nombre: Activar y desactivar los *plugins*.	
Descripción: Una vez que se muestre en la interfaz aquellos *plugins* que han sido cargados, se seleccionarán aleatoriamente solo algunos de ellos para comprobar si se pueden activar o desactivar.	
Condiciones de ejecución: Deben existir *plugins* cargados.	
Pasos de ejecución: • Abrir el proyecto *AsiXMec*. • Compilar el proyecto. • Ir a la ventana de los *snaps*. • Activar o desactivar varios *plugins*.	
Resultados esperados: Debe poder marcarse o desmarcarse cualquiera de los *plugins*.	

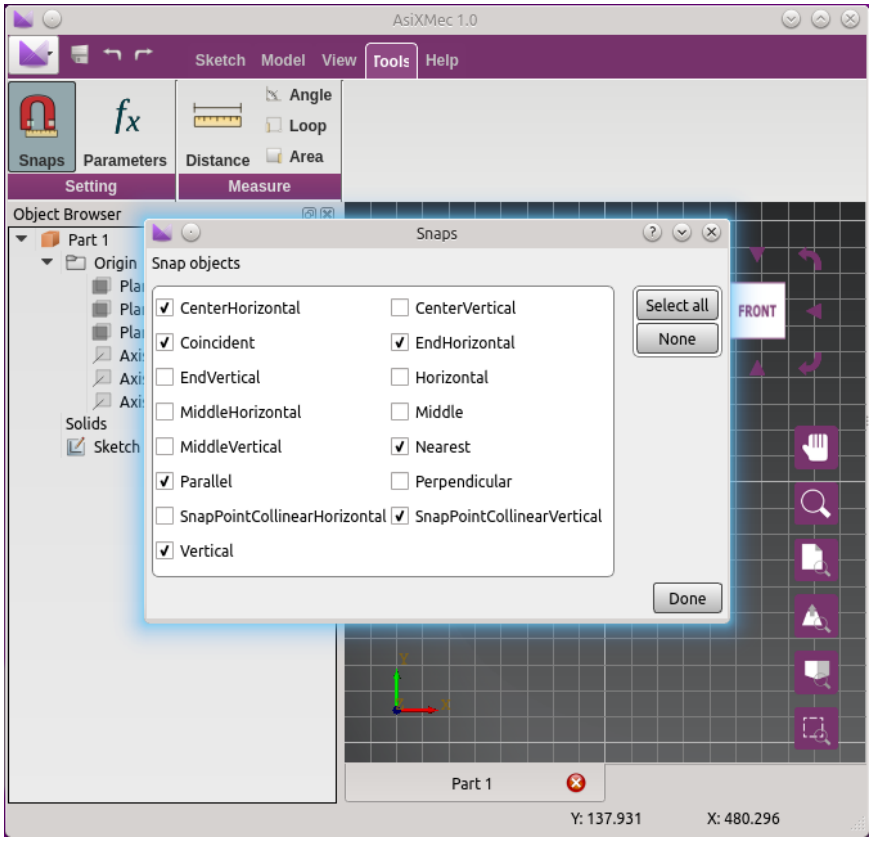


Figura 3.6. Activación y desactivación de los *plugins*.

3.2.2. Validación del módulo

La validación es el proceso en el cual se verifica que el sistema cumple con todos los requerimientos que el cliente pide. Una vez terminadas las pruebas de aceptación, las cuales arrojaron resultados satisfactorios, se verificó que la nueva versión se integra sin errores al sistema *AsiXMec*, además de que pueden ser adicionados al sistema en forma de *plugins* cada uno de los *snaps* que contiene. Una vez que el *software* sea ejecutado, estos se cargarán automáticamente, mostrándose en una interfaz visual donde el cliente podrá controlar su activación o desactivación.

Consideraciones finales del capítulo

Tras la implementación y pruebas del sistema ha quedado satisfechos todos los requerimientos pedidos por el cliente. Para ello fue de gran utilidad cada uno de los artefactos generados en capítulos anteriores. Una vez concluidas las pruebas, las cuales arrojaron resultados satisfactorios, se dieron por concluidas las iteraciones del proceso de desarrollo.

Conclusiones

Al término de la presente investigación se concluye que la nueva versión del módulo *Snap* del *software AsiXMec*:

- Permite al equipo de desarrollo incorporar nuevos *snaps* o modificar los que ya existen sin la necesidad de compilar todo el proyecto, reduciendo así el acoplamiento en el *software AsiXMec*.
- Brinda a los usuarios una vez inicializado el *software AsiXMec*, cargar, mostrar y seleccionar los *snaps* en forma de *plugins*, los cuales puede habilitar o deshabilitar teniendo en cuenta las necesidades del diseño.

Recomendaciones

Se recomienda para dar continuidad al presente trabajo:

- Incorporar al sistema nuevos *snaps* que asistan al usuario en el proceso de creación de entidades 2D, como por ejemplo el *snap* tangente y el *snap* intersección.
- Crear un modelo matemático mediante el cual se pudiera desarrollar otros *snaps* sin hacer uso de las funcionalidades que brinda la biblioteca *OCE*.

Acrónimos

AUP *Agile Unified Process*. 24

CAD *Computer Aided Design*. 1, 2, 4–7, 10, 15, 24

CATIA *Computer Aided Three-dimensional Interactive Application*. 6

IDE *Integrated Develoment Environment*. 20, 22

OCE *Open CASCADE Community Edition*. 22, 28, 50

PTC *Parametric Technology Corporation*. 5

RUP *Rational Unified Process*. 24

XP *Extreme Programming*. 24–27, 34, 39, 41

CRC Clase, Responsabilidad y Colaboración. 34, 38

DISEM Diseño y Simulación de Estructuras Mecánicas. 1, 15, 21, 28

GoF Gand of Four. 36

MVC Modelo-Vista-Controlador. 18, 19

RAE Real Academia Española. 19

TIC Tecnologías de la Informática y las Comunicaciones. 4

UCI Universidad de las Ciencias Informáticas. 1, 15

Referencias bibliográficas

Arcia Salazar, Miguel (2011). «Desarrollo de un componente visual para Qt utilizando el framework OpenCascade.» Tesis doct. Universidad de las Ciencias Informáticas (vid. pág. 22).

Autodesk (2015). *nXtRender - Rendering Plugin for AutoCAD* (vid. pág. 5).

Bashir, M Salman y M Rizwan Jameel Qureshi (2012). «Hybrid Software Development Approach For Small To Medium Scale Projects: Rup, Xp and Scrum». En: *Cell* 966, pág. 536474921 (vid. págs. 24, 41).

Blanchette, Jasmin y Mark Summerfield (2006). *C++ GUI programming with Qt 4*. Prentice Hall Professional (vid. pág. 21).

CadStock (2010). *Screenshot, plugin para capturar piezas o ensamblajes en Autodesk Inventor*. URL: http://cadstock.com/noticias/420-screenshot-plugin-para-capturar-piezas-o-ensamblajes-en-autodesk-inventor (vid. pág. 6).

Canós, José H., Patricio Letelier y M. Carmen Penadés (2010). «Métodologías Ágiles en el Desarrollo de Software». En: *DSIC -Universidad Politécnica de Valencia* (vid. pág. 23).

Caño, Alfredo del, Mª de la Cruz y Luis Solano (2007). «Diseño, ingeniería, fabricación y ejecución asistidos por ordenador en la construcción: evolución y desafíos a futuro». En: *Informes de la Construcción* 59.505, págs. 53-71 (vid. pág. 5).

Cendejas Valdéz, José Luis (2010). *2.11 Modelos y metodologías para el desarrollo de software*. http://www.eumed.net/tes doctorales/2014/jlcv/software.htm (vid. pág. 23).

Community, LibreCAD (2014). *LibreCAD, Open Source 2D-CAD*. http://librecad.org/cms/home.html. URL: http://librecad.org/cms/home.html (vid. pág. 5).

Cordero, Jorge Luis (2010). «METODOLOGIAS AGILES PROCESO UNIFICADO AGIL (AUP)». En: (vid. pág. 24).

Corporation, Nokia (2011). *Qt Creator Manual*. URL: http://doc.qt.digia.com/qtcreator-2.3/index.html (vid. págs. 20-22).

Definición.de (2015). *Definición de Plugin*. http://definicion.de/plugin/. URL: http://definicion.de/plugin/ (vid. pág. 19).

Dingsoyr, Torgeir et al., (2012). «A decade of agile methodologies: Towards explaining agile software development». En: *Journal of Systems and Software* 85.6, págs. 1213-1221 (vid. págs. 23, 24, 34).

Echeverry, Tobón, Luis Miguel y Luz Elena Delgado Carmona (2007). «Caso práctico de la metodología ágil XP al desarrollo de software». Tesis doct. UNIVERSIDAD TECNOLÓGICA DE PEREIRA FA-

CULTAD DE INGENIERÍA: ELÉCTRICA, ELECTRÓNICA, FÍSICA Y CIENCIAS DE LA COMPUTACIÓN INGENIERÍA DE SISTEMAS (vid. pág. 33).

Freeman, Eric et al., (2005). *Head First Design Patterns*. ISBN: 0-596-00712-4. O'Reilly Media, Inc (vid. págs. 19, 36).

Gamma, Erich et al., (2008). «Design Patterns: Elements of». En: (vid. pág. 35).

Hansen, Scott (2013). *Autodesk Inventor 2014: A Tutorial Introduction*. SDC Publications. URL: https://books.google.com.cu/books_id=w8JclpKGe6cC&lpg=PP2&ots=-PabpjO9ag&dq=tutorial%20of%20Inventor&lr&pg=PP2#v=onepage&q=tutorial%20of%20Inventor&f=false (vid. pág. 6).

ITESM Instituto Tecnológico y de Estudios Superiores de Monterrey, México (2013). *Qué es un Plug-in*. URL: http://itesm.custhelp.com/app/answers/detail/a_id/382 (vid. pág. 20).

Jackson, P. et al., (2013). «Method of creating a snap point in a computer-aided design system». US20130055125 A1. US Patent App. 13/214,962. URL: http://www.google.com/patents/US20130055125 (vid. págs. 4, 10, 11).

Joyanes Aguilar, Luis (1996). «Programación orientada a objetos». En: *Universidad Pontificia de Salamanca, Campus Madrid* (vid. pág. 17).

Larman, Craig (1999). *UML y Patrones*. Pearson (vid. pág. 36).

Letelier, Patricio (2006). «Métodologías ágiles para el desarrollo de software: eXtreme Programming (XP)». En: (vid. pág. 24).

Lockhart, Shawna (2013). *Tutorial Guide to AutoCAD 2014*. SDC Publications. URL: http://www.google.com.cu/books_hl=en&lr=&id=Sffb7LhMV-4C&oi=fnd&pg=PR11&dq=tutorial+of+AutoCad&ots=5yJ1-VY5XB&sig=SbqHyhd3_IKD99r67S2r63JEr3k&redir_esc=y#v=onepage&q=tutorial%20of%20AutoCad&f=false (vid. pág. 5).

Loret de Mola Pino, Lina de la Caridad (2013). «Portal web de la Comunidad Cubana de Realidad Virtual». Tesis doct. Universidad de las Ciencias Informáticas Facultad-5 (vid. pág. 41).

Pressman, Roger S (1997). *Ingeniería del Software: Un enfoque práctico*. Mikel Angoar (vid. págs. 17, 18, 37).

Pressman, S. Roger (1992). *Ingeniería del software: Un enfoque práctico*. Ed. por McGraw Hill. 5.ª ed. (vid. págs. 18, 27).

Reynoso, L et al., «Un Análisis Experimental sobre el Efecto del Acoplamiento en la Comprensibilidad y Facilidad de Modificación de Expresiones OCL». En: (vid. pág. 18).

Rodríguez Corbea, Maite y Meylin Ordoñez Pérez (2007). «La metodología XP aplicable al desarrollo del software educativo en Cuba.» Tesis doct. Universidad de las Ciencias Informáticas (vid. págs. 27, 28, 30, 33, 34, 41).

Saldaña Pomazunco, Jorge Luis (2012). «Creación de Estándares para planos mediante el software CAD Autodesk Inventor Professional 2011». Tesis doct. Universidad Nacional Mayor de San Marcos, Perú (vid. pág. 4).

SAS, Open CASCADE. *Open CASCADE Technology, 3D modeling and numerical simulation*. http://www.opencascade.or Consulta. URL: http://www.opencascade.org/ (vid. pág. 22).

Shih, Randy (2013). *AutoCAD 2014 Tutorial-First Level: 2D Fundamentals*. SDC Publications. URL: https://books.google.com.cu/books?id=7kBovUofxw0C&lpg=PR3&ots=4FPl3ZP-L3&dq=tutorial%20of%20AutoCad&lr&pg=PR3#v=onepage&q=tutorial%20of%20AutoCad&f=false (vid. pág. 8).

Shih, Randy H. (2014). *AutoCAD 2014 Tutorial - First Level: 2D Fundamentals*. Schroff Development Corporation (vid. pág. 10).

Software, Siemens PLM (2014). *Solid Edge*. URL: http://www.plm.automation.siemens.com/en%5C_us/products/solid-edge/index.shtml (vid. pág. 7).

Sommerville, Ian (2005). *Ingeniería del software*. Pearson Educación (vid. págs. 17-19, 22, 27, 28).

Stroustrup, Bjarne (1999). «An overview of the c++ programming language». En: *The Handbook of Object Technology* (vid. pág. 21).

Systèmes, Dassault (2014a). *CATIA is Dassault Systèmes Pioneer Brand. It is the World's Leading Solution for Product Design and Innovation.* http://www.3ds.com/products-services/catia/xtmc=catia&xtcr=1. URL: http://www.3ds.com/products-services/catia/xtmc=catia&xtcr=1 (vid. pág. 6).

— (2014b). *Disvover the SOLIDWORK 2015 Product Portfolio*. http://www.solidworks.com/. URL: http://www.solidworks.com/ (vid. pág. 6).

Tiwari, Anurag, Patel Praveen y Darbari Manuj (2013). «Global Prospect of Distributed Agile Software Development: A Review». En: *UACEE International Journal of Advances in Computer Science and its Applications* 3: Issue 2.ISSN 2250-3765 (vid. págs. 23, 24).

Trayak (2014). «EcoDesigner». En: URL: http://www.trayak.com/solutions/ecodesigner/ (vid. pág. 7).

Vértice, Editorial (2010). *Photoshop*. Editorial Vértice. URL: http://www.google.com.cu/books_hl=en&lr=&id=hSHLoLd4qIAC&oi=fnd&pg=PP2&dq=guias+en+photoshop&ots=JRGrG49aI-&sig=WcDc_9o9ZdjpS2Dc0JR-IiKz4vA&redir_esc=y#v=onepage&q=guias%20en%20photoshop&f=false (vid. págs. 7-9).

Watson, David (2015). *Object Snap*. URL: http://www.cadtutor.net/tutorials/autocad/object-snap.php (vid. pág. 11).