UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA

TRABAJO FIN DE MÁSTER EN INGENIERÍA INDUSTRIAL

DESARROLLO DE UN SOFTWARE DE ANÁLISIS PARA LA INSPECCIÓN AUTOMATIZADA DE LA CALIDAD DE LOS MODELOS CAD GEOMÉTRICOS.

AUTORA: INMACULADA POU SCHMIDT

TUTOR: NURIA ALEIXOS BORRÁS

COTUTOR: FRANCISCO ALBERT GIL

Curso Académico: 2019-2020

Agradecimientos

“Dedico mis agradecimientos, en primer lugar, a mis padres, siempre dispuestos a brindarme la oportunidad y el consejo más acertado y prudente.

A mis profesores, Nuria y Cisco, por su amabilidad y disposición.

Y por último a Juan Pablo y Gabriel por su apoyo y amistad.”

Resumen

El objetivo de este proyecto es programar un software que sirva como método de corrección de modelos Cad 3D utilizando una referencia. Las soluciones que existen actualmente para este problema no son del todo satisfactorias ya que no analizan en detalle los modelos y solo se obtiene una valoración respecto al resultado final del modelo, dejando de lado otras consideraciones de valor como el proceso seguido, las operaciones seleccionadas, …

El programa que se realice se utilizará en la asignatura de ingeniería gráfica para la corrección de exámenes por lo que se basará en gran medida en los criterios de evaluación de profesorado de esa asignatura.

Para la realización de software se utilizará la herramienta Ilogic dentro de Autodesk Inventor. Ilogic es una plataforma de programación que utiliza el lenguaje Visual Basic. Permite crear reglas de programación para automatizar procesos de diseño dentro de los modelos y crear formularios para facilitar la interacción con el usuario. Aunque el cometido para el que está diseñada la herramienta Ilogic no es específicamente el de realizar programas como el del proyecto, será de gran utilidad debido a su sencillez y a estar ya adaptado para trabajar con Inventor.

Finalmente, una vez completada la primera versión del programa se probará en una serie de modelos y se mostraran los resultados. También se plantearán algunas posibles mejoras o ampliaciones del mismo.

Resum

L'objectiu d'aquest projecte és programar un software que serveixi com a mètode de correcció per a models 3D CAD utilitzant una referència. Les solucions que existeixen actualment per a aquest problema no són del tot satisfactòries ja que no s'analitzen els models en detall i només s'obté una valoració respecte del resultat final del model, deixant de banda altres consideracions de valor com el procés seguit, les operacions seleccionades, ...

El programa que es dugui a terme s'utilitzarà en l'assignatura d'enginyeria gràfica per a la correcció d'exàmens per la qual cosa es basarà en gran mesura en els criteris d'avaluació del professorat d'aquesta assignatura.

Per a la realització del sofware s'utilitzarà l'eina iLogic dins d'Autodesk Inventor. ILogic és una plataforma de programació que utilitza el llenguatge Visual Bàsic. Permet crear regles de programació per automatitzar els processos de disseny dins dels models i crear formularis per facilitar la interacció de l'usuari. Si bé la finalitat per a la qual està dissenyada l'eina iLogic no és específicament realitzar programes com el del projecte, serà molt útil per la seva senzillesa i perquè ja està adaptada per a treballar amb Inventor.

Finalment, un cop finalitzada la primera versió del programa, es provarà en una sèrie de models i es mostraran els resultats. També es consideraran algunes possibles millores o extensions.

Abstract

The objective of this project is to program a software that serves as a correction method for 3D Cad models using a reference. The solutions that currently exist for this problem are not entirely satisfactory since they do not analyze the models in enough detail. Hence, only an assessment is obtained regarding the final result of the model, leaving aside other valuable considerations such as the process followed, the selected operations, ...

The program that is carried out will be used in the graphic engineering subject for the correction of exams so it will be based largely on the evaluation criteria of the teaching staff of this specific subject.

For the development of the software, the Ilogic tool will be used within Autodesk Inventor. Ilogic is a programming platform that uses the Visual Basic language. Moreover, allowing you to create programming rules to automate design processes within models and create forms to facilitate user interaction. Although the purpose for which the Ilogic tool is designed for is not specifically to carry out programs like the one in this project, it will be very useful due to its simplicity and since it is already adapted to work with Inventor.

Finally, once the first version of the program is completed, it will be tested on a series of models and the results will be displayed. Some possible improvements or extensions will also be considered for a more efficient development.

Índice de documentos

I.

Memoria

Índice de la Memoria

Índice de figuras

1.Introducción

Este proyecto consiste en la programación de un software de análisis para la inspección automatizada de la calidad de los modelos 3D

1.1 Motivación y objetivos

La motivación del proyecto es la elaboración de una herramienta que permita la corrección masiva de modelos industriales similares, en los que hay ciertas condiciones que deben cumplirse para su posterior incorporación a un conjunto. Esto sirve de ayuda en las industrias en las que en muchas ocasiones se realiza una revisión manual de los diseños que se realizan en la oficina técnica para asegurar están siguiendo las pautas de calidad y normalización del diseño. En este trabajo nos centraremos en la aplicación como herramienta de corrección de modelos tipo examen, que se puede utilizar tanto en procesos de selección de empresas como a nivel educativo.

Los objetivos principales del proyecto son:

1. Aprender a utilizar la herramienta *ilogic.*
2. Aprender programación en lenguaje *VisualBasic.net*
3. Elaborar un software utilizando *ilogic* que permita comparar todas las características de 2 modelos diseñados en *Autodesk Inventor .*

1.2 Alcance del proyecto

En este TFM se buscará comparar tanto el resultado final del modelado de la pieza como el procedimiento utilizado. Con toda la información de los modelos se sacará un informe de las similitudes y diferencias entre el modelo de referencia y el modelo a comparar. En el informe aparecerá una puntuación que reflejará el grado de validez del modelo en función de la similitud con la referencia. Como se ha mencionado anteriormente, se centrará en el caso de corrección para modelos tipo examen en los que se dispone de un modelo perfecto de referencia y un gran número de estos que comparar.

Es necesario establecer que criterios que se deben tener en cuenta para establecer la validez de un modelo. Son muchos los artículos y documentos que versan sobre este tema y que ayudarán a definir cuáles son las características que hacen que un modelo se considere válido o no, este tema se trata artículo del congreso Internacional de Ingeniería gráfica en el artículo" Metodología de modelado con herramientas CAD/CAM avanzadas". [1]

Como guía para detectar las necesidades que debe cubrir el software para la evaluación de exámenes se utilizará la rúbrica de la asignatura de ingeniería grafica del grado de Tecnologías industriales donde se reflejan los criterios de evaluación y sus pesos aproximados sobre la nota final.

[1] Contero, M., Company, P., Aleixos, N. y Vila, C. 2000. Metodología de modelado con herramientas CAD/CAM avanzadas. XII Congreso Internacional de Ingeniería Gráfica. ISBN 84-8448-008-9

Bloques	Parámetros	Escala valoración
Secuencia de modelado, orientación y colocación del modelo en el espacio (≈ 10-15 %)	-Sigue una secuencia lógica de modelado (geometría principal al inicio y detalles como redondeos y chaflanes al final) -Organiza bien las operaciones en el árbol del modelo (agrupa operaciones relacionadas y no actúa de manera arbitraria) -En el árbol del modelo no deben haber bocetos sueltos, no se han de realizar operaciones innecesarias, etc. -Orientación y colocación de la primera operación correctamente (pieza centrada en los planos de origen siempre que se pueda, según su posición de funcionamiento)	-No -Sí
Operaciones de modelado utilizadas, parámetros de las operaciones (≈ 30-35 %)	-Escoge correctamente las operaciones de modelado a realizar (revolución, agujero, nervio, rosca, chaflán, vaciado, etc.) -Utiliza correctamente las operaciones de trabajo (planos, ejes y puntos de trabajo) -Los parámetros de las operaciones capturan correctamente la intención de diseño (pasante todo, pasante hasta, simetría, etc.).	-No / Nunca -Casi nunca -A veces -Casi siempre -Sí / Siempre
Bocetos (≈ 50-60 %)	-Planos de los bocetos elegidos correctamente -Geometría correcta (contornos cerrados y sin cruces, que no hayan entidades superpuestas, que no hayan entidades troceadas o segmentadas, etc.) -Posición correcta de los bocetos en el plano (localiza correctamente el boceto en el plano y usa las referencias adecuadas) -Restricciones geométricas adecuadas (captura la intención de diseño en el boceto conservando tangencias, simetrías, concentricidades, etc.) -Restricciones dimensionales adecuadas (el esquema de acotación acorde con el plano/perspectiva acotada de la pieza)	

Ilustración 1Rúbrica para la evaluación de modelos 3D

La rúbrica permite establecer un criterio de evaluación del que tanto alumnos y profesores son conocedores. La evaluación mediante rúbricas puede ayudar a los alumnos a detectar los requisitos de diseño y ha interiorizar la necesidad de mostrar una intención de diseño en sus trabajos, como se explica en el artículo "A contribution to conveying quality criteria in mechanical CAD models and assemblies through rubrics and comprehensive design intent qualification. PhD Thesis, Submitted to the Doctoral School of Universitat Politècnica de València (2017)" [2] y también "Go/No Go Criteria in Formative E-Rubrics"[3]

Como puede observarse en la rúbrica se valora la colocación de la pieza, las operaciones utilizadas, la secuencia de modelado, las restricciones utilizadas, … Todos estos factores presentes en la rúbrica se tienen en cuenta como reflejo de la intención de diseño del alumno ya que el resultado final obtenido no es lo único que se debe valorar. La intención de diseño es igual o más importante que el resultado obtenido ya que marcará la integridad y la posible reutilización del diseño. Factores que son importes para los alumnos como futuros ingenieros ya que hay que ver más allá de los objetivos de la asignatura, el estudiante no está aprendiendo a modelar con un programa CAD para aprobar un examen sino para utilizar estos conocimientos en su futuro profesional por lo que es imprescindible concienciar a los alumnos sobre la importancia de un buen modelado. La intención de diseño y la reutilización de los modelos están estrechamente ligados. Este tema se discute ampliamente en el artículo "*Design reuse in a CAD environment – four case studies. Comput. Ind. " [4]*

Una vez finalizado el desarrollo del software se realizarán una serie de pruebas o ejemplos de aplicación con distintos modelos. Esto se utilizará como método para subsanar posibles carencias que tenga el código, para demostrar su validez a la hora de ser utilizado y para mejorar la interfaz intentando que sea lo más clara y simple posible para el usuario. También, tras poner a prueba el código surgirán nuevas áreas de crecimiento del mismo para aumentar sus posibilidades de aplicación y mejorar su rendimiento.

1.3 Antecedentes

Design Checker Autodesk Inventor

Design Checker es un complemento instalable en Autodesk Inventor que sirve para verificar algunos aspectos del modelo que las empresas pueden tener como estándares, diámetros de agujeros tipos de rosca. La herramienta permite obtener información en tiempo real sobre características del modelo.

Permite configurar las características que se desean revisar:

[2] Otey, J.: A contribution to conveying quality criteria in mechanical CAD models and assemblies through rubrics and comprehensive design intent qualification. PhD Thesis, Submitted to the Doctoral School of Universitat Politècnica de València (2017)

[3]Company, P., Otey, J., Agost, M.-J., Contero, M., Camba, J.D. 2018. Go/No Go Criteria in Formative E-Rubrics. Lecture Notes in Computer Science, 10925 LNCS, pp. 254-264

[4] Andrews, PTJ, Shahin, TMM, Sivaloganathan, S. Design reuse in a CAD environment – four case studies. Comput. Ind. Eng. 1999; 37(1): 105-9.

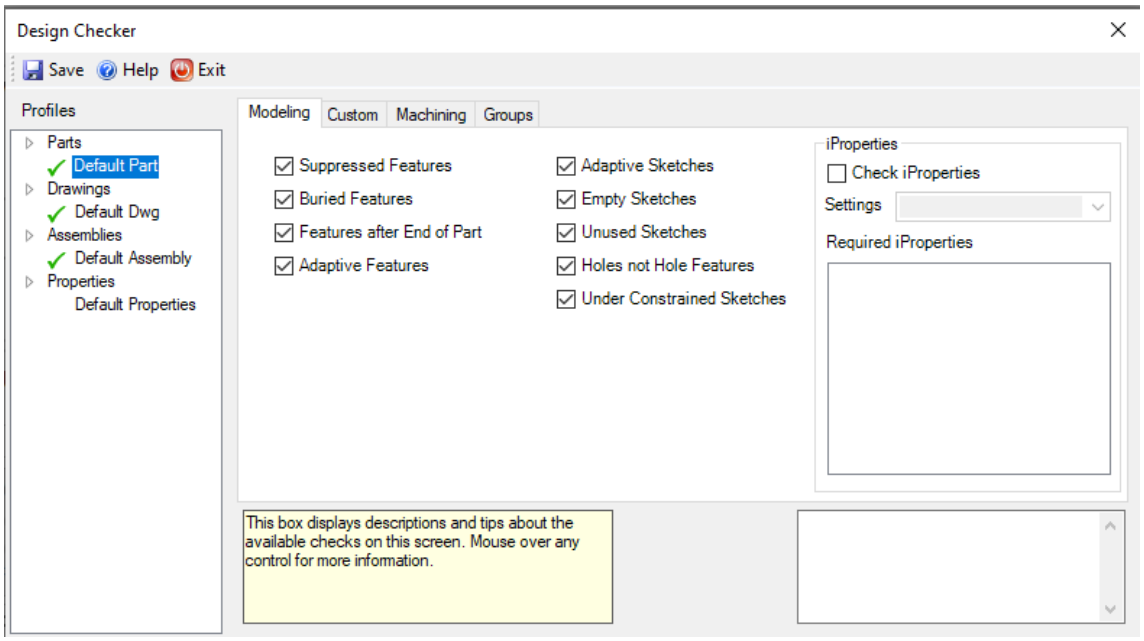

Ilustración 2 Captura de Design Checker

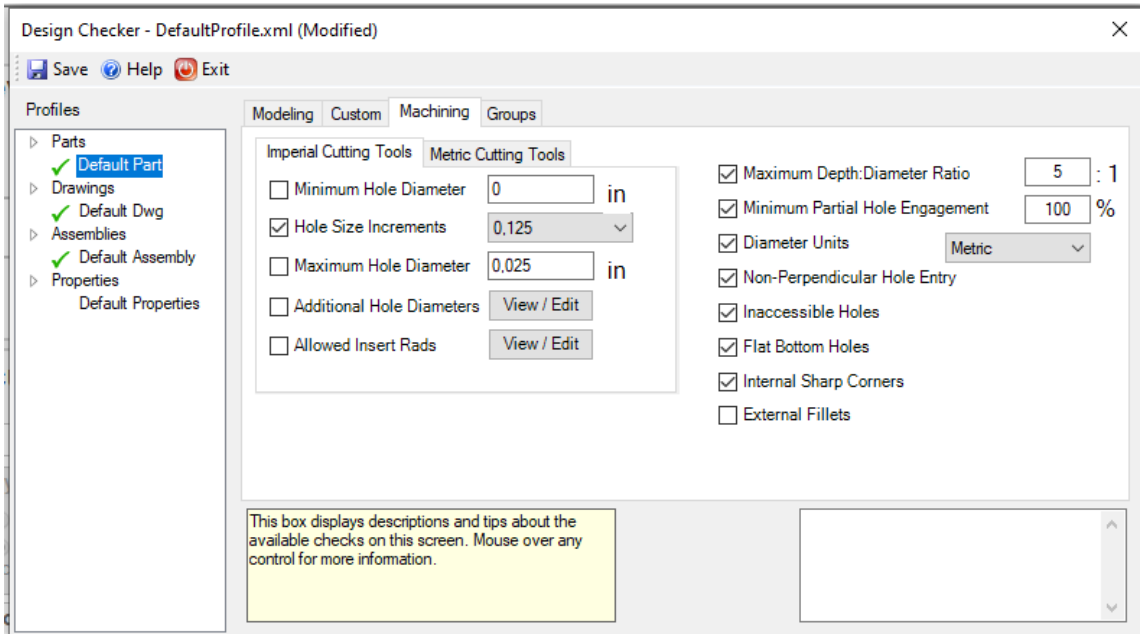

Ilustración 3 Captura de Design Checker

A medida que se realiza el modelo aparecen una serie de avisos en el árbol del modelo que indican si hay algún problema con esa operación. Design Checker tiene algunas características interesantes, pero no es exactamente lo que se busca en este proyecto ya que no solo se quiere comprobar características específicas sino comparar dos modelos en todas sus características.

Universidad de Purdue

La universidad de Purdue en EE. UU utiliza un software de corrección para sus modelos CAD. Esta comprueba el volumen final y el centro de gravedad de la pieza y según el resultado obtenido asigna un Apto o No Apto a sus alumnos. Este es un comienzo interesante, pero no suficiente para lo que se busca conseguir con este proyecto ya que se espera poder evaluar más aspectos del modelo y poder conseguir una nota para cada modelo.

1.4 Material y métodos

Las principales herramientas necesarias para la elaboración del proyecto son Autodesk Inventor y su módulo de personalización *Ilogic*, que permite programar pequeñas funciones tanto para la personalización del programa como para la automatización de la generación de modelos.

1.4.1 Metodología de programación

Para la elaboración del código se ha seguido un "desarrollo de prototipos ". El comienzo ha sido un planteamiento de los requisitos que debe cumplir el software y las funciones básicas que se necesitarán, para la posterior realización de un ciclo en el que se programa una parte pequeña de código, se corrige y - cuando esta funciona - se procede a la siguiente parte. Seguir esta metodología es de gran utilidad, debido a que permite ir aprendiendo y explorando nuevas características de *Ilogic* y de vb.net a medida que el código crece.

El ciclo utilizado durante toda la realización del proyecto es:

- **Programación de la función**. Se programa la función o una parte de ella. En este caso para el desarrollo se ha realizado cada función por separado en su propia regla para posteriormente incorporarlas a la regla externa.
- **Corrección de errores.** Se detectan y corrigen los errores de compilación
- **Prueba del prototipo**. Se comprueba que el código escrito realiza la función deseada. En este caso concreto se han utilizado pantallas de texto que iban informado de los valores que adquirían las distintas variables (algunas creadas expresamente para verificar el código) o del punto del código en el que se encuentra.
- **Vuelta al inicio del ciclo**. Una vez probado el prototipo, se vuelve a la fase de programación para corregir el código, en caso de ser necesario, incorporar nuevas funcionalidades o crear una nueva función
- Una vez completada la función se incorpora a la regla externa, que será la que ejecute el programa, se verifica que funciona correctamente y se pasa así al inicio del ciclo con una nueva función.

Otra característica a destacar con respecto a la metodología empleada es que en la medida de lo posible se ha procurado crear funciones que no realicen más de una operación, facilitando que el código sea fácil de entender y de seguir. Lo mismo pasa con los nombres que se les ha dado tanto a las funciones como a los parámetros, se ha procurado que sean fáciles de entender de forma que su nombre indique que contienen o que función realizan. También se ha intentado crear funciones de las subrutinas más repetitivas a lo largo del código.

El objetivo de la estructura de la programación es intentar mejorar la claridad. Se ha utilizado preferentemente la subrutina *if* para la selección y *for* para la iteración.

Asimismo, la programación está orientada a objetos, debido a que *Ilogic* y *Visual Basic* funcionan de esta forma, pero no se han creado nuevas clases ni objetos, utilizando las que vienen creadas desde *Inventor*.

1.5 Palabras clave

- **Calidad de los modelos CAD 3D;**
- **Features;**
- **Ilogic: Reglas y funciones (Visual Basic);**
- **Evaluación de modelos CAD;**

2.Marco teórico

En este apartado se explicarán brevemente las 3 herramientas principales que se utilizarán para el desarrollo del proyecto con el fin de exponer los conocimientos básicos necesarios para la comprensión del proyecto. Estas son:

El programa de diseño Autodesk Inventor. El software a desarrollar deberá revisar y evaluar modelos realizados en esta plataforma, por lo que resulta indispensable conocer profundamente el manejo de esta herramienta.

La herramienta de programación *Ilogic*: Ya que está será la plataforma sobre la que se trabajará para la programación del software.

VisualBasic.NET: Debido a que este es el lenguaje de programación utilizado en *Ilogic*, por lo que a pesar de que este dispone de funciones propias se hará uso de muchas funciones y estructuras habituales en VB.net

2.1 Sobre Autodesk Inventor

Autodesk inventor es un software de la compañía *Autodesk* utilizado para el modelado paramétrico de sólidos, que además de permitir la creación de modelos 3D permite generar planos de los modelo y generación de conjuntos. Es similar a otros softwares que podemos encontrar en el mercado como por ejemplo *Solid Works, Solid Edge* o *Catia*.

Inventor permite trabajar en un espacio 3D y generar diferentes modelos mediante la utilización de diversas operaciones, como pueden ser extrusión, revolución, redondeo, taladro… El usuario crea una serie de bocetos o "scketches" en 2D a partir de los cuales se podrán conformar figuras complejas mediante la aplicación sucesiva de funciones. Utiliza un método de trabajo CSG, combinación de solidos elementales mediante operadores booleanos.

Se trata de una herramienta muy completa con gran cantidad de utilidades. Dependiendo del trabajo a realizar utiliza distintos archivos: *.dwg* para planos de piezas, *.ipt* para piezas, *.iam* para ensamblajes (unión de varias piezas) y *.ipn* para presentaciones (se utiliza por ejemplo para crear vista explosionada). En este proyecto se trabajará sobre un archivo. *iam* ya que permite la apertura simultánea en un mismo archivo de dos o más archivos *.ipt,* que es el formato en el que estarán los archivos que se compararán.

Ilustración 4: Tipos de archivos en Inventor

2.1.1 Interfaz

Al abrir un archivo pieza (.ipt) nuevo se puede ver la siguiente pantalla.

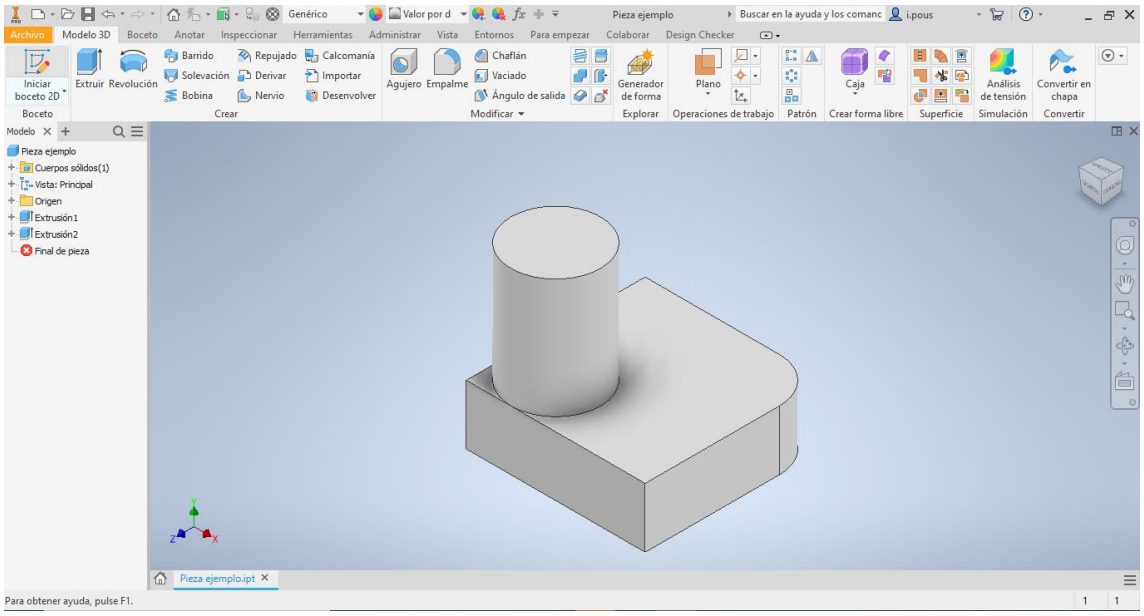

Ilustración 5: Pantalla de creación de pieza inventor 2020.

A continuación, se mostrarán las distintas zonas en las que se divide la pantalla haciendo hincapié en las que resultarán más útiles a lo largo del desarrollo del proyecto.

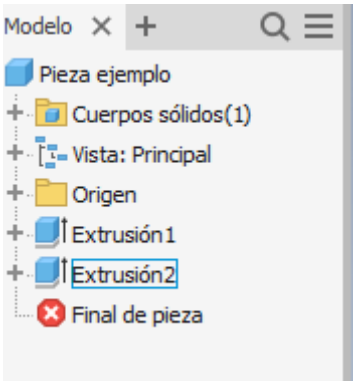

Ilustración 6: Navegador del modelo

A la izquierda de la pantalla se encuentra por defecto el navegador del modelo. Desde él se puede consultar y acceder a el árbol del modelo, es decir, ver y modificar todas las operaciones y bocetos realizados hasta el momento. En ese mismo espacio se pueden incluir otros navegadores, como por ejemplo el navegador de *Ilogic*.

En la parte superior se pueden encontrar distintas pestañas. Para la creación de piezas se utilizan principalmente la de modelado 3D y Boceto.

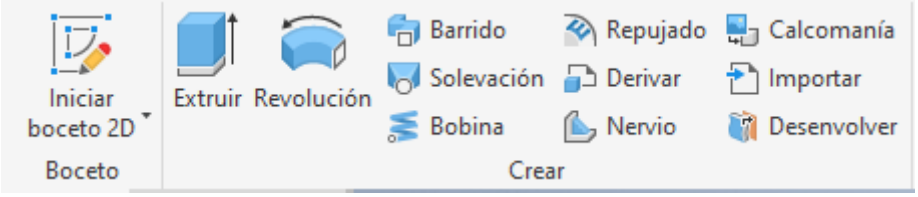

Ilustración 7 Principales operaciones de la pestaña de modelado 3D

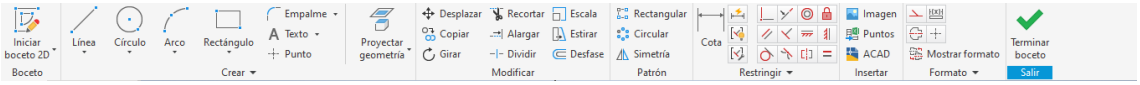

Ilustración 8 Pestaña de Boceto

A la hora de realizar bocetos es de interés para el proyecto se debe hacer hincapié en la restricción de los bocetos. Un boceto se puede restringir mediante cotas o restricciones geométricas. Es fundamental para la integridad del modelo que este esté completamente restringido. En la esquina inferior derecha de la pantalla un letrero informa de si esto es así o no.

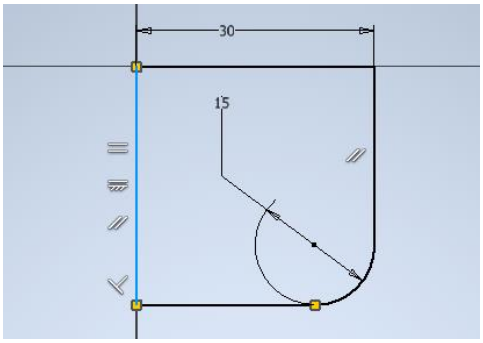

Ilustración 9:Ejemplo de boceto con dos tipos de restricciones

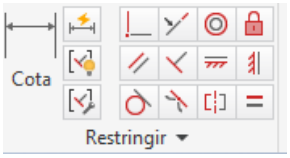

Ilustración 10 : Restricciones geométricas

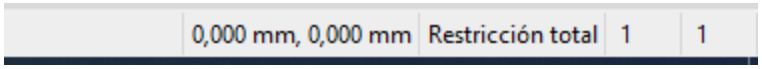

Ilustración 11: Letrero de estado de restricción del modelo

Por último, en el panel administrar encontramos el botón parámetros 

donde podemos consultar, modificar y crear nuevos parámetros para el modelo, y el panel de herramienta de Ilogic.

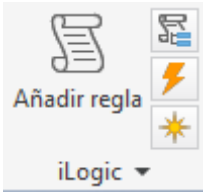

Ilustración 12: Panel de herramientas de Ilogic

2.1.2 Tipos de parámetros

Podemos encontrar distintos parámetros dentro del programa, con diferentes utilidades:

Parámetros del modelo: Son los parámetros utilizados por el programa para la generación del modelo, son las cotas asignadas en el boceto o las dimensiones indicadas para la realización de una operación.

Parámetros de usuario: son creados manualmente por el usuario y se pueden asignar a las características del modelo, por ejemplo, cotas en bocetos.

Se generan desde el cuadro de parámetros de inventor. Hay de tres tipos: numérico, de texto o verdadero y falso. Se deben asignar un nombre, valor y unidad, esto último solo en caso de ser numérico. Además, se pueden crear parámetros de valor múltiple de cualquiera de los tres.

Los parámetros también se pueden importar desde un archivo Excel vinculándolo al modelo. Esto es de gran utilidad a la hora de diseñar, debido a que permite relacionar matemáticamente los valores de distintos parámetros a través de una hoja Excel, una herramienta conocida por todos.

2.1.3 Tipos de operaciones

Es indispensable para la elaboración del software conocer las características de las operaciones que el modelo revisará. A continuación, se expondrán las distintas operaciones que se pueden realizar con el programa y sus características. Para mayor concreción, solamente se explicarán las operaciones que se utilizan en la asignatura.

- **Extrusión:** Crea un volumen a partir de un perfil cerrado, le añade profundidad. Necesita un boceto que contenga un perfil cerrado, una distancia y una dirección de extrusión. También se puede realizar una extrusión en dos direcciones.

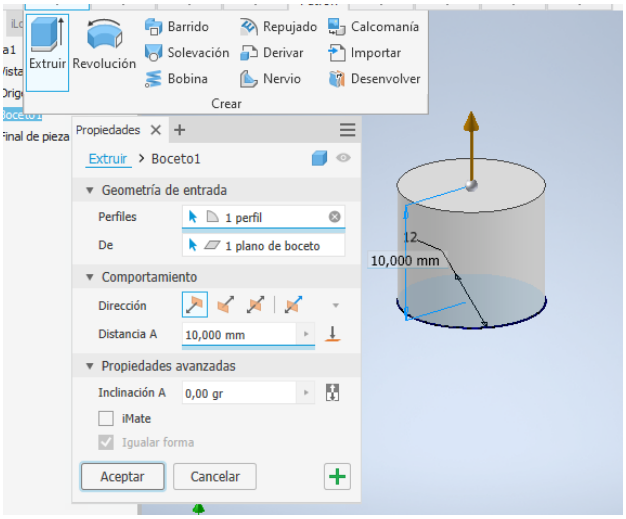

Ilustración 13 Captura Autodesk Inventor

Si se le indica una inclinación se genera un perfil recto de sección variable.

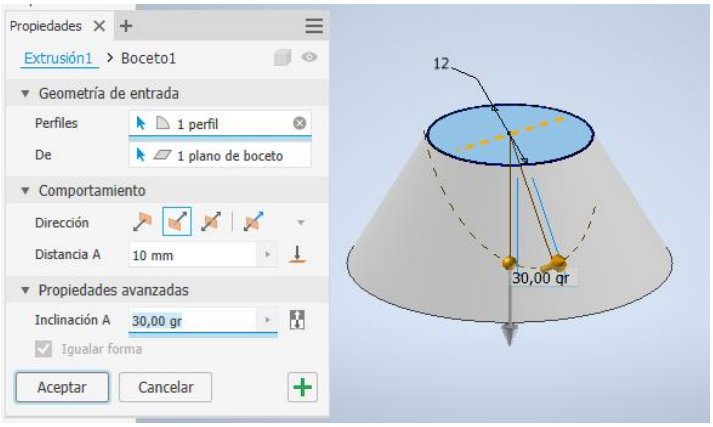

Ilustración 14 Captura Autodesk Inventor

Por lo tanto, la operación *extrusión* genera mínimo dos parámetros de modelo, una distancia y una inclinación, pero puede generar hasta cuatro parámetros en casos de indicar dos direcciones: dos distancias y dos inclinaciones.

Además, también se pueden realizar extrusiones negativas, en lugar de añadir material lo eliminan.

- **Agujeros:** Se pueden encontrar gran variedad de opciones en el menú de agujeros:

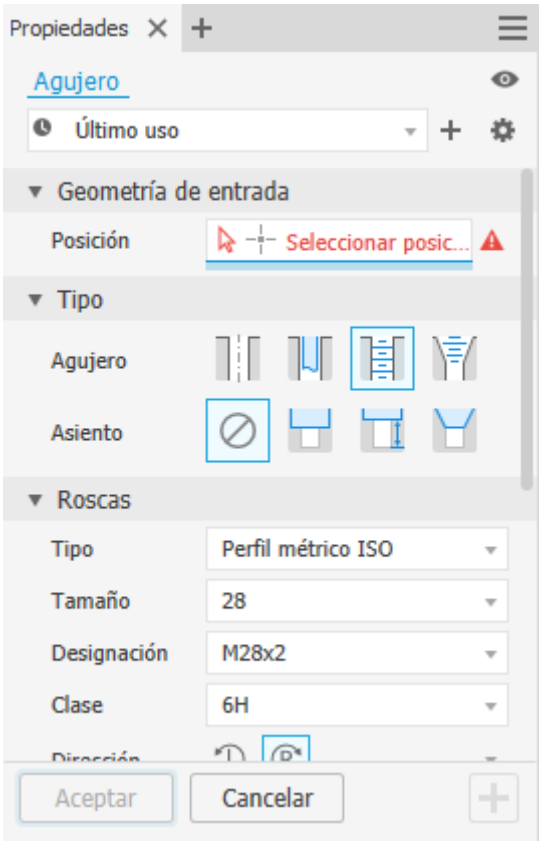

Ilustración 15 Captura Autodesk Inventor

La primera distinción que se hace del tipo de agujero es que se puede hacer a partir de boceto, indicando el centro del agujero o por concentricidad. Después se pueden encontrar distintos tipos de agujero sencillo, con juego, rosca o rosca inclinada. Se piden una serie de parámetros.

Dentro de cada tipo de agujero se puede elegir terminación (*por distancia, pasante* o *hasta*), dirección y tipo de punta (p*lana o en ángulo*). Otra variación que se puede añadir a los agujeros es el tipo de asiento: escariado, refrentado, avellanado o sin asiento.

Cada una de estas variantes supone unos parámetros y características distintas del modelo, lo que dificulta mucho su análisis mediante software.

- **Revolución:** Sirve para generar cuerpos de revolución. Esta operación crea un cuerpo sólido mediante el giro de un perfil entorno a un eje. Se utiliza para crear simetrías radiales, cuerpos cilíndricos, etc.

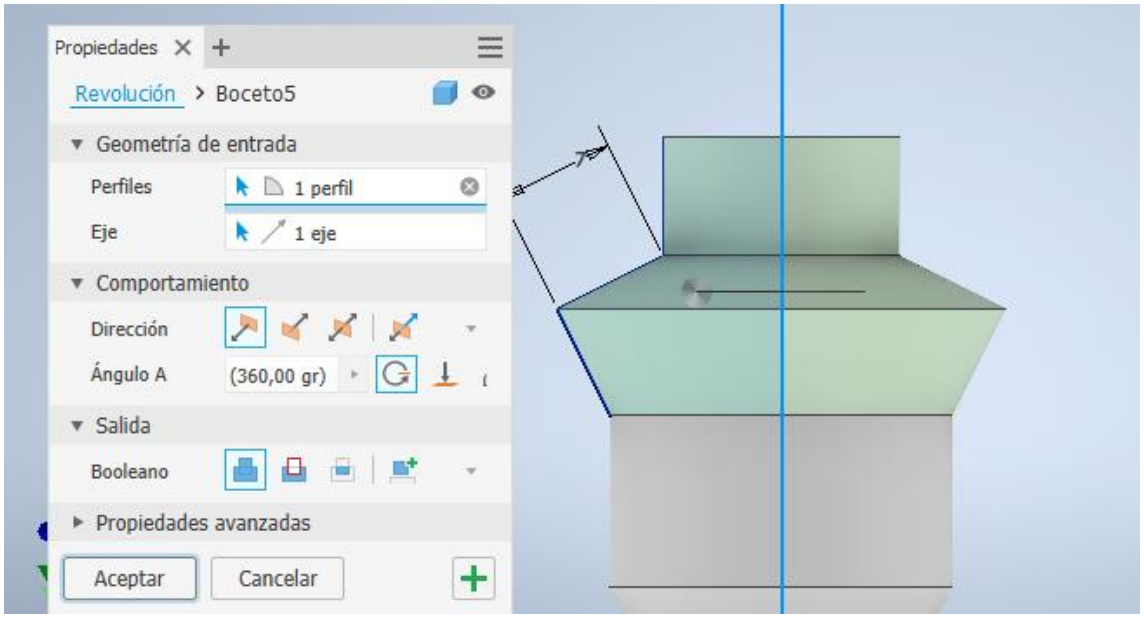

Ilustración 16 Captura Autodesk Inventor

Por lo tanto, para una operación de revolución se necesita como parámetro el ángulo de giro, además de los parámetros del boceto en el que se contiene el perfil. Al igual que en la extrusión, se puede eliminar material en lugar de añadirlo. También se le debe indicar la dirección y el eje de giro.

- **Empalme:** Redondea la arista seleccionada utiliza el radio de acuerdo que le indica el usuario. Existen multitud de opciones de esta herramienta, pero todas comparten que lo que necesitan como parámetro es el radio.

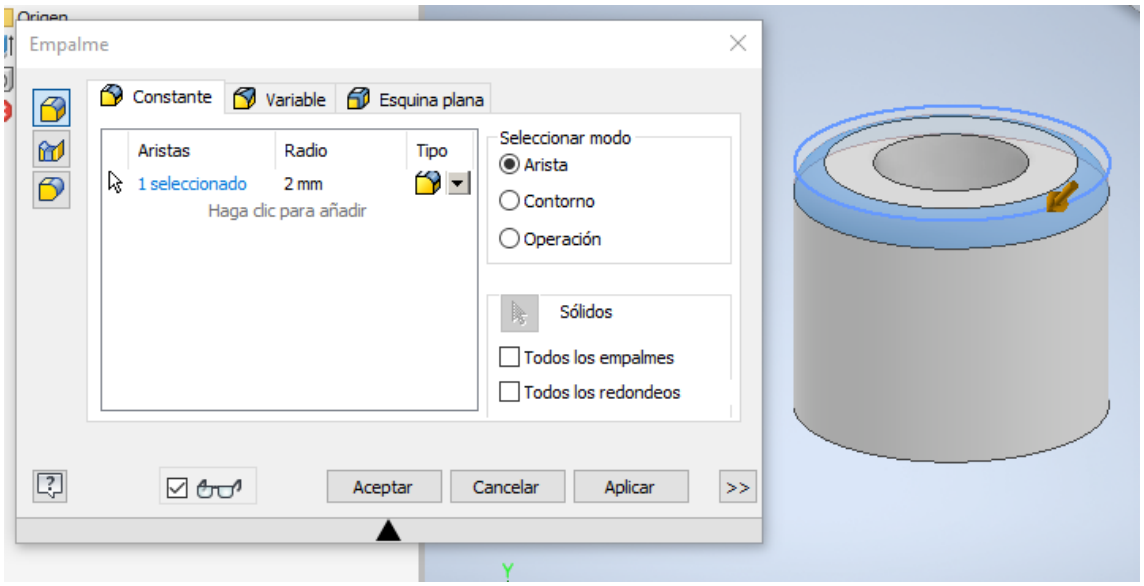

Ilustración 17 Captura Autodesk Inventor

- **Chaflan:** Realiza un corte plano en una arista. Hay distintas variaciones según los parámetros que se conocen, distancia y ángulo, dos distancias o la misma distancia en ambas direcciones.

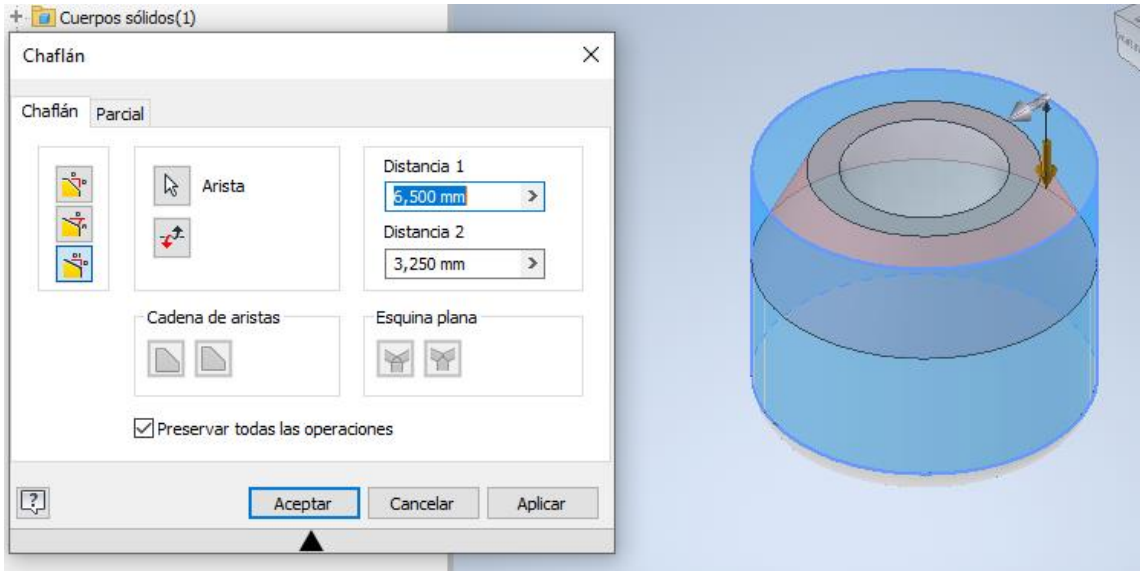

Ilustración 18 Captura Autodesk Inventor

- **Vaciado:** Crea una cavidad hueca a partir de una cara del sólido. Las paredes del sólido resultante tienen como espesor el indicado por el diseñador .

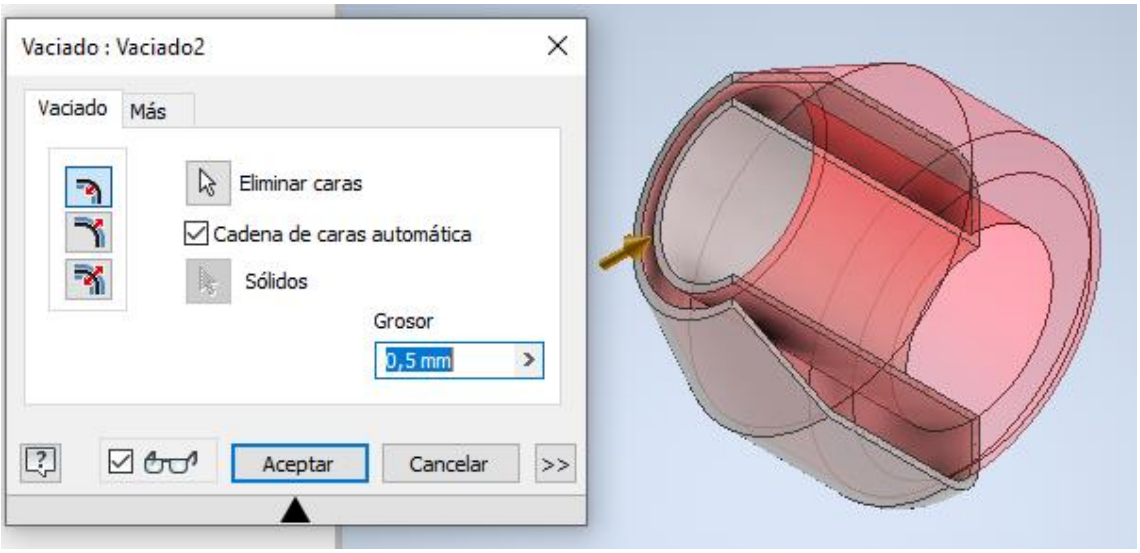

Ilustración 19 Captura Autodesk Inventor

La operación necesita como único parámetro el espesor que se desea que tenga la pared de la cavidad.

- **Simetría**: Crea una geometría simétrica a la seleccionada a partir de un plano de simetría. Permite realizar simetría de operaciones concretas de la pieza, no es necesario que sea el sólido completo.

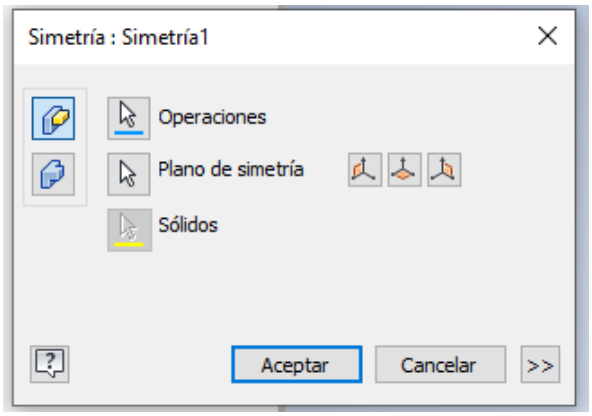

Ilustración 20 Captura Autodesk Inventor

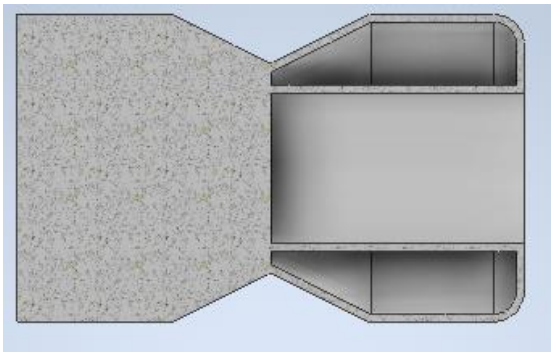

Ilustración 21 Captura Autodesk Inventor

- **Patrón**: Dada una geometría u operación seleccionada esta operación la repite tantas veces como se le indique. Hay dos tipos de patrones: circular y rectangular.

El patrón circular necesita que se le indique un ángulo y un número de repeticiones, además del eje de giro. El patrón rectangular necesita que se le indique el número de repeticiones y separación en ambas direcciones

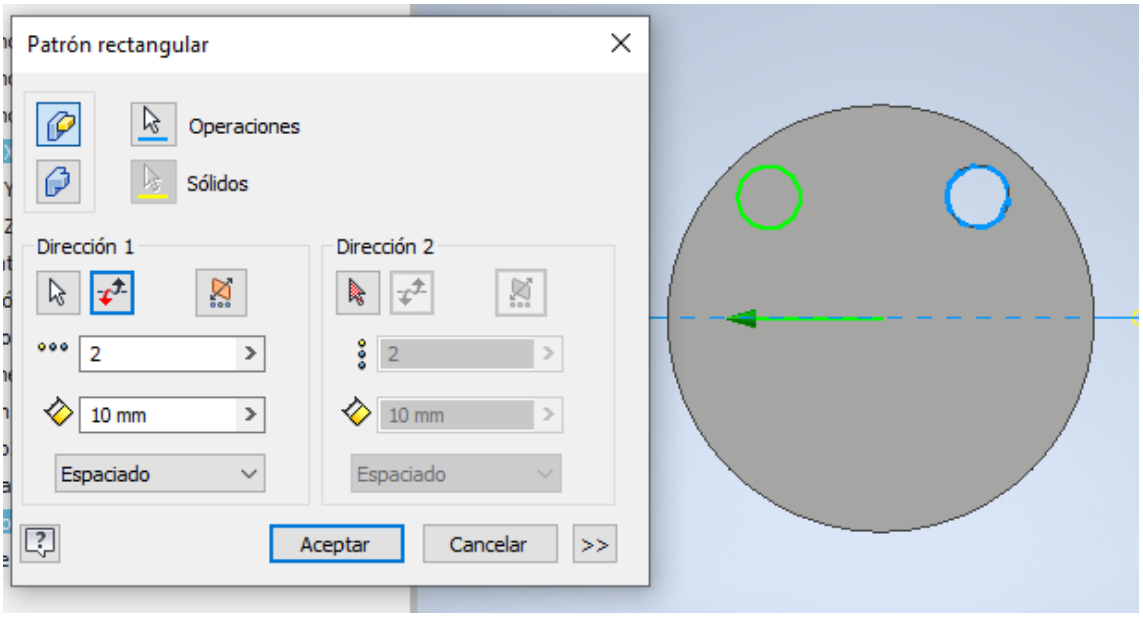

Ilustración 22 Captura Autodesk Inventor

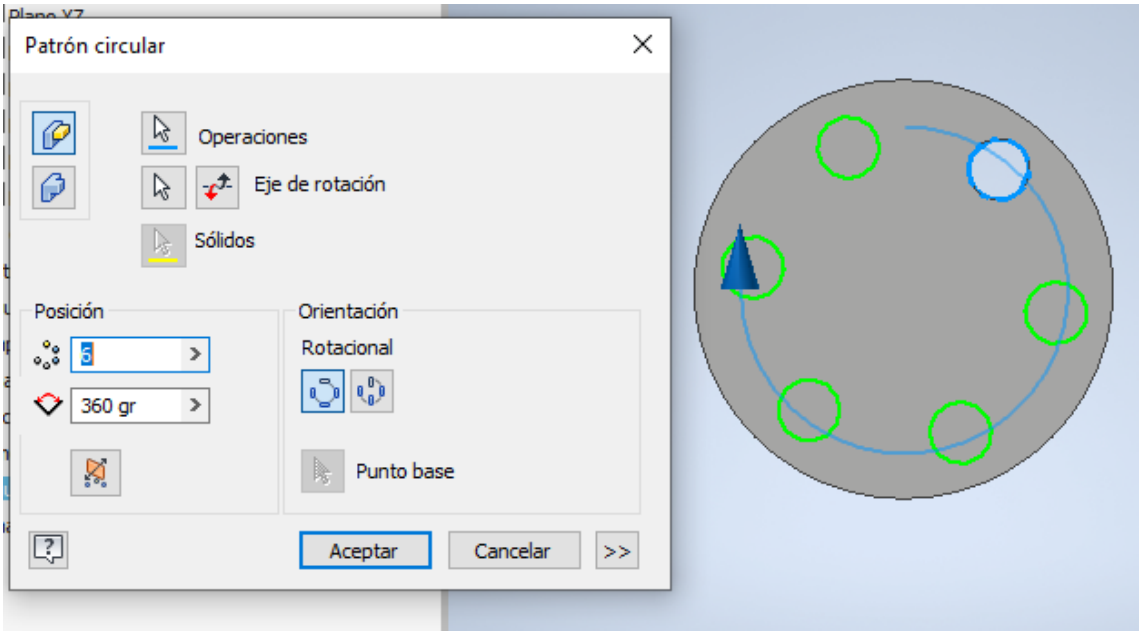

Ilustración 23 Captura Autodesk Inventor

2.2 Sobre Ilogic

La herramienta *Ilogic* que se incluye en *Autodesk Inventor* permite a los diseñadores crear modelos inteligentes y automatizados. Mediante el uso de reglas de programación sencilla permite automatizar los comportamientos del modelo introduciendo modificaciones en función de las variables que desee el usuario. Comúnmente es utilizada en casos en los que se desean generar varias piezas con características similares, aunque no iguales. Utilizando una regla, el diseñador puede introducir las variantes específicas de cada modelo sin necesidad de crear un documento desde 0. Un ejemplo típico de aplicación sería una pieza que tiene un número

determinado de agujeros dependiendo de la longitud de su lado o un auto escalado de los elementos dependiendo de las dimensiones generales del mismo. En ambos casos el ingeniero a cargo del diseño de esas piezas utilizando *Ilogic* podría crear una o varias reglas en las que utilizando una programación muy simple controlaría esas características. Y es que una de las principales características de *Ilogic* es que no exige un alto conocimiento de programación para poder utilizarlo. Con esta herramienta, también podemos crear formularios que faciliten el uso de la herramienta y la aplicación de las reglas.

Una de las ventajas a destacar de la utilización de *Ilogic* es la reutilización de modelos, permitiendo esto un gran ahorro de tiempo. Desde *Ilogic* puedes modificar de forma automática todas las características del modelo, material, dimensiones, componentes, eliminar operaciones…

Es una herramienta orientada a usuarios de *Inventor* de un nivel más avanzado, pero como ya se ha mencionado, no requiere conocimientos previos de programación, aunque la tenencia de unos fundamentos básicos si puede facilitar el trabajo con dicha herramienta.

Podemos acceder a la creación de dichas reglas desde el botón de *Ilogic* en la pestaña *administrar* o haciendo clic en el botón + del navegador del modelo.

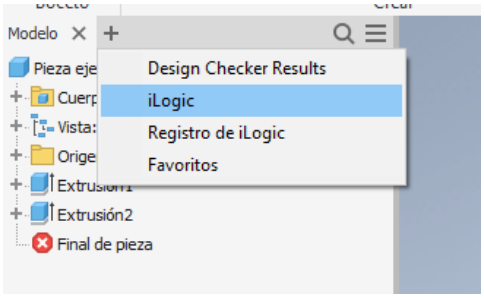

Ilustración 24:Entrar a Ilogic

Una vez entramos en el panel de Ilogic nos encontramos lo siguiente:

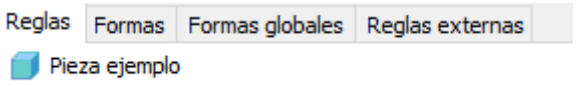

Ilustración 25 posibilidades de ilogic

Estos son los dos tipos de formularios y reglas que podemos crear con Ilogic. A continuación, pasaremos a explicar en qué consiste cada apartado y algunas de sus particularidades

2.2.1 Reglas

Como podemos observar en el apartado anterior podemos encontrar “Reglas” o “Reglas externas”. Ambos tipos de reglas se crean exactamente de la misma forma; la principal diferencia es que las reglas son propias de cada documento, mientras que las reglas externas son comunes a todos los documentos. Podemos crear una nueva regla haciendo clic derecho sobre el nombre de la pieza que se puede ver en la sección “Regla”. Aparecerá una pantalla como la de la siguiente imagen en la que se diferencian distintas zonas.

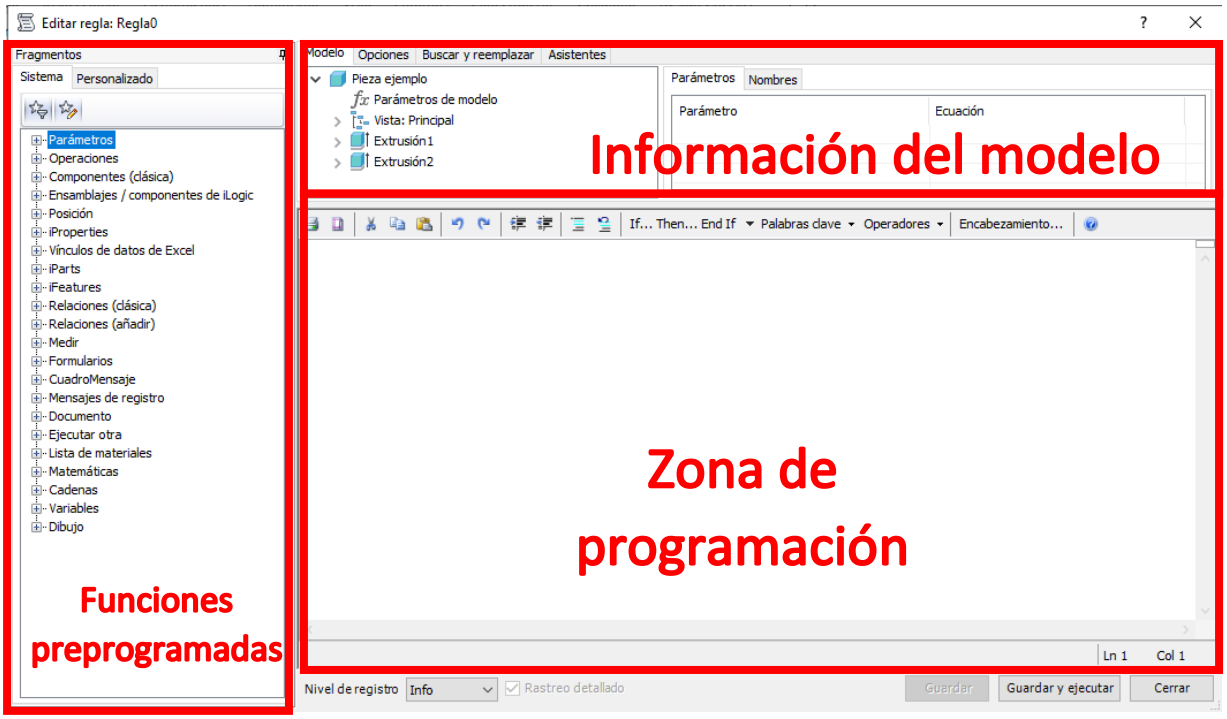

Ilustración 26: Pantalla de creación de regla y sus elementos

- *Zona de programación:* es, como su nombre indica, el espacio utilizado para escribir el código de la regla. Dispone de un acceso rápido a algunas estructuras o elementos más habituales, como por ejemplo los condicionales "if" y los operadores matemáticos
- *Zona de información del modelo:* muestra el árbol del modelo, sus operaciones y parámetros y nos da acceso rápido a los mismos.
- *Zona de funciones preprogramadas:* es la que se encuentra a la izquierda de la pantalla y contiene un gran número de funciones para facilitar la programación de la regla. Al situar el ratón sobre alguna de las funciones aparece una breve explicación de su función y de los parámetros que necesita que le introduzcan, y al hacer clic sobre ella coloca el fragmento de texto en la zona de programación.

Las más utilizadas son:

Parameter("Nombre del parametro"): Permite acceder o cambiar el valor de un parámetro del modelo.

`Feature.IsActive("featurename")`: Permite activar o desactivar operaciones.

`Components.Add(…)`: Permite añadir un componente a un ensamblaje

En el caso de las reglas, el procedimiento es exactamente el mismo. La única diferencia radica en que hay que indicar en que directorio del ordenador se encuentran las reglas que vamos a utilizar . Estas reglas se pueden crear desde inventor o simplemente crear y cargar un documento *txt* que contenga el código.

2.2.2 Formularios

Ilogic dispone de una herramienta para crear formularios. No obstante, esto no ofrece una gran variedad de herramientas, ya que *Ilogic* no está planteado para crear programas de gran complejidad.

Del mismo modo que la reglas y las reglas externas existen formularios y formularios globales. Los formularios son propios de cada documento, mientras que los formularios globales son compartidos entre todos.

Se puede crear un nuevo formulario haciendo clic derecho en el espacio en blanco de la pestaña formas y a continuación "Añadir forma". Aparecerá la siguiente pantalla en la que se pueden distinguir distintas zonas:

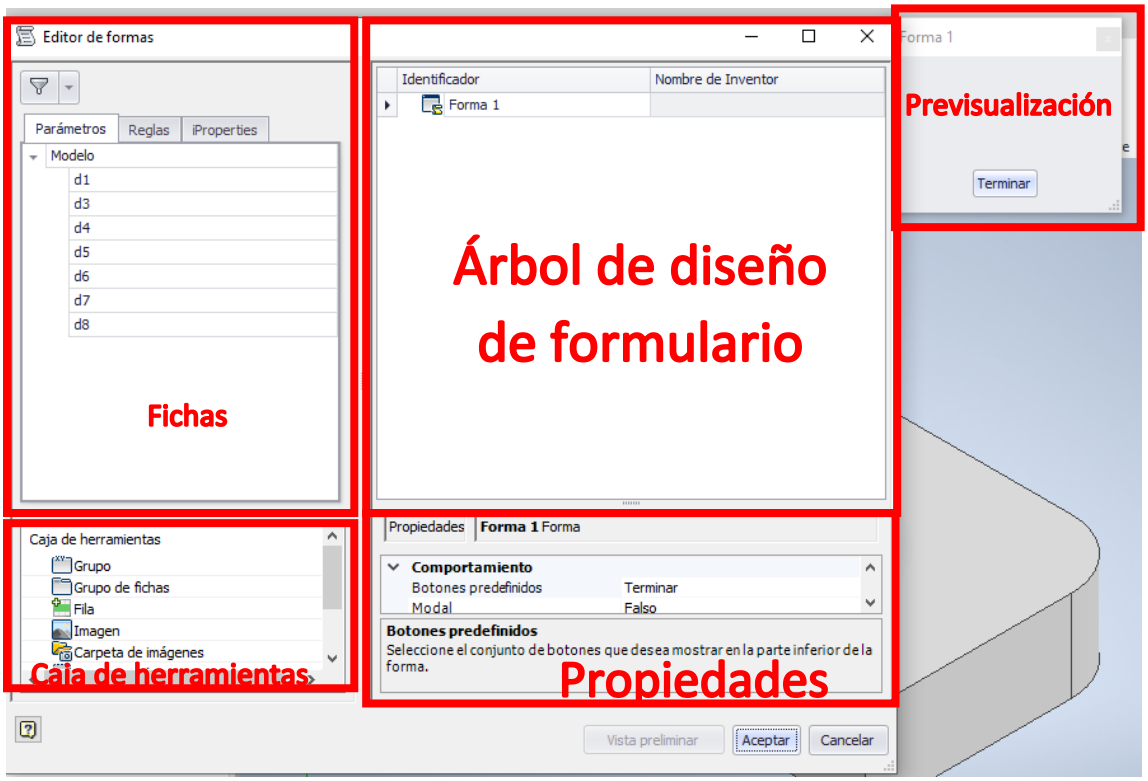

Ilustración 27: Pantalla de creación de formulario

- "*Fichas*": Da acceso a distintos elementos del modelo como parámetros, reglas o propiedades físicas. Al arrastrar uno de estos elementos sobre la zona de personalización permite interactuar con ellos. En el caso de incorporar un parámetro permite modificar su valor y en caso de incorporar una regla aparecerá en el formulario como un botón que al hacer clic sobre ejecutará la regla.
- "*Caja de Herramientas*": Permite incorporar al formulario elementos generales para modificar su apariencia y hacer más amigable para el usuario.
  - Grupo: Permite agrupar elementos en cuadros bajo un mismo nombre
  - Grupo de fichas: permite crear pestañas en el formulario
  - Fila : Permite agrupar componentes horizontalmente
  - Imagen: Añade una imagen al formulario
  - Carpeta de imágenes: permite disponer de una carpeta de imágenes alternativas
  - Espacio vacio: añade un espacio en blanco

- Identificador: Permite añadir un texto no editable al formulario como un título.
- Divisor: incorpora un separador en forma de barra entre los elementos del formulario

- *"Árbol de diseño del formulario":*: Zona en la que se trabaja para configurar la apariencia y elementos del formulario
- *Propiedades:* Permite introducir algunas modificaciones sobre los elementos incorporados al formulario, como por ejemplo tipografía, estilo y botones…
- Ajustar tamaño de los controles
- Editar el tipo de control
- Cambiar el nombre del parámetros
- Elegir el tipo de funete
- Elegir los botones del formulario
- Elegir las dimensiones de cada herramienta
- Seleccionar si es modal o no,solo lectura o no…
- Estilo visual
- Otras funciones de personalización del formulario

- Pantalla de Previsualización: Muestra el resultado final del formulario de forma simultánea con los cambios realizados.

2.3 Sobre vb.net

Visual basic, también conocido como *VB* es el lenguaje de programación orientado a objetos que se utiliza en la aplicación *Ilogic*. Se trata de un lenguaje dirigido por eventos, es decir, que está determinado por los sucesos del sistema que define o provoque el usuario. Es la base de lo conocido como interfaz de usuario. Un activador de evento puede ser por ejemplo un botón: el usuario lo pulsa y eso desencadena una serie de acciones del programa. El usuario es el que marca el flujo del programa.

Habitualmente se utiliza en el entorno *Microsoft Visual Studio* , pero es el caso que nos ocupa se utilizará *Ilogic* como entorno de programación, ya que facilita la interacción con *Autodesk Inventor.*

El uso de *VB* en *Ilogic* permite tener acceso a archivos externos de distintos formatos, acceder a aplicaciones de Windows u otras aplicaciones externas, pero también utilizar funciones *API* de *Inventor*

Las variables más habituales en *vb.net* son

Tipo	*Descripción*
Integer	Numeros enteros
Double	Valores reales de doble precisión
String	Cadena de texto
Char	Caracter
Float	Valores reales en punto flotante
void	Punteros genéricos

Ilustración 28 Tabla de tipos de parámetros

Además de los tipos de datos fundamentales, *Visual basic* permite crear otros tipos de datos compuestos a su vez por otros tipos de datos, lo que permite manejar gran variedad de datos relacionados. Al tratarse de un leguaje orientado a objetos, puedes definir un objeto con unas características específicas y atributos propios. Una vez creado pueden crearse varios objetos del mismo tipo. Además, los distintos tipos de objetos se pueden crear con relaciones de herencia entre ellos. Esto significa que las clases derivadas entienden las funciones y características propias del objeto base.

En el caso de *Ilogic* vienen ya creados una serie de objetos o tipos relacionados con *Inventor* (*partfeature, parameter, document*…) que serán los que se utilizarán en el proyecto.

3.Planteamiento inicial

En este apartado se explicarán las distintas características que el programa debe inspeccionar, los planteamientos de cómo debe ser, comportarse y las opciones de las que debe disponer.

Se desarrollarán también de forma detalla las dificultades que entraña y que se esperan encontrar en la realización del proyecto.

3.1 Consideraciones previas

3.1.1 Elementos a inspeccionar

Se han determinado una serie de requisitos que debe cumplir el programa a desarrollar y que serán en los que marcarán el resultado final del proyecto.

- El programa debe comparar un modelo considerado perfecto, el del profesor, con el modelo del alumno, el que se va a corregir.
- El nombre que se le asignará a cada operación es una exigencia de enunciado para poder identificar correctamente los distintos elementos del modelo.
- Uno de los aspectos a inspeccionar por el programa es si el modelo es geométricamente correcto.
- Además de esto, debe inspeccionar y comparar el procedimiento seguido por el alumno, ya que es de interés para la evaluación que el alumno siga una serie de pautas o buenas prácticas para el diseño de piezas en 3D .
- También es de gran relevancia para el profesor la inspección de los bocetos y el tipo de restricciones utilizadas por el alumno.
- Los resultados de la evaluación deberán registrarse en una hoja Excel.

La principal problemática que entraña esto es que el software a desarrollar debe funcionar en infinidad de modelos. Esto supone un reto, ya que no se conocen con certeza datos como el número de operaciones, los nombres de las mismas, parámetros del modelo…

3.1.2 Planteamiento de la interfaz de usuario

En el planteamiento inicial el formulario consistiría en una pantalla en la que el profesor podría elegir unas características principales a inspeccionar y una puntuación asignada a cada una de ellas. Además, debería permitir seleccionar una serie de operaciones secundarias de las que se indicaría el número que debe encontrar en el documento.

A lo largo del desarrollo de proyecto este planteamiento ha ido evolucionando hasta llegar al formulario final.

- Se han eliminado los selectores de operaciones secundarias, debido a que se ha considerado mejor solución comparar la cantidad de operaciones de cada tipo en lugar de centrar la atención en una.
- Se ha mantenido el selector de operaciones principales, pero se ha eliminado el parámetro de puntuación, ya que al existir una gran variabilidad entre modelos los criterios de corrección pueden variar mucho. Se ha considerado mejor establecer un sistema de puntuación más completo y personalizable por el profesor.

4.Desarrollo y resultado final

4.1 Formulario

El formulario es la parte del programa con la que el usuario interactuará, por lo que debe ser sencillo e intuitivo. Por ello, se ha creado un formulario organizado por pestañas.

En la primera pestaña se encuentran las opciones de selección de archivos; concretamente, se limita a dos botones, uno para la selección del archivo de referencia y otra para para la selección del modelo a evaluar. Además, se muestra un cuadro desplegable (con el título 'Help') en el que se dan algunas explicaciones e información sobre los pasos a seguir.

Ilustración 29 Pestaña 1 del formulario

Cuando el usuario selecciona cualquiera de las dos opciones se abre una ventana de navegador de Windows para seleccionar un archivo .ipt (archivo tipo pieza en inventor). Al seleccionar un archivo a evaluar automáticamente, el programa guarda los nombres de todos los archivos .ipt de la carpeta , con lo que se evita tener que seleccionar todos los archivos a evaluar para facilitar el proceso de corrección.

Una vez seleccionados los archivos necesarios, éstos se cargan en el ensamblaje y se pasa a la siguiente pestaña. En ella el profesor podrá seleccionar las operaciones más importantes del modelo y el peso que les quiere dar sobre la puntuación final. Para poder determinar la operación se utilizará un selector que se crea de forma automática al elegir el archivo de referencia. Podrán incluirse hasta 10 operaciones para revisar.

Esta puntuación se divide en partes iguales entre tres factores a revisar: la geometría conseguida, el tipo de operación utilizado y las restricciones del boceto consumido por la operación. Igual que en el caso anterior, hay un cuadro de texto explicativo.

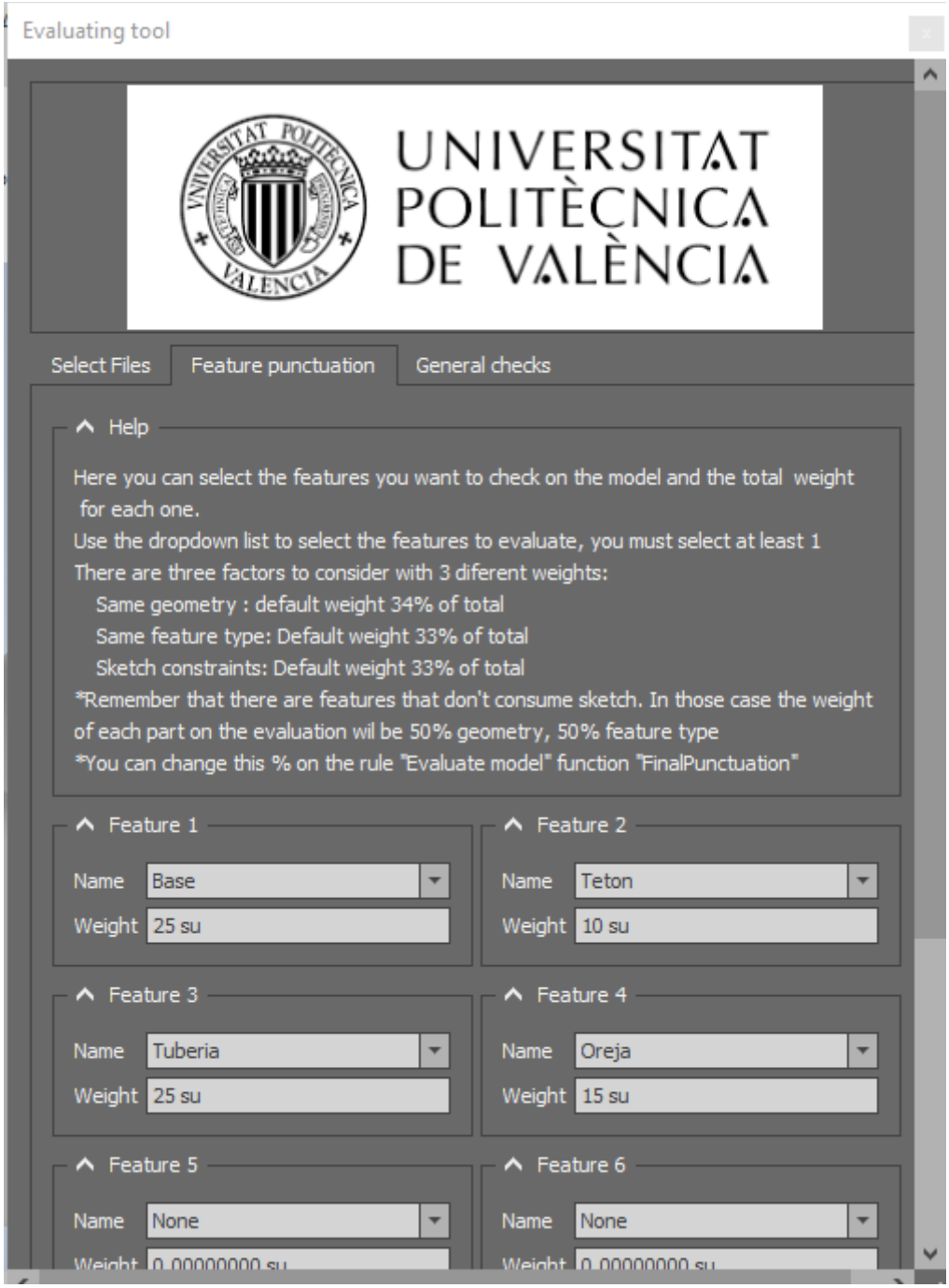

Ilustración 30 Pestaña 2 del formulario

Por último, la tercera pestaña permite al profesor dar una puntuación a otros aspectos más generales del modelo que inspeccionará el programa. Por un lado, se le asignará una puntuación a la similitud general del modelo, volumen total del modelo y volumen de interferencia entre piezas. También se le asignará una puntuación a la inspección de operaciones secundarias, como por ejemplo redondeos o chaflanes. En la parte inferior se puede ver el número de operaciones de cada tipo que se encuentran en la referencia.

Ilustración 31 Pestaña 3 del formulario

En la parte inferior de esta pestaña se encuentra el botón para iniciar la evaluación. Cuando se pulse, el programa realizará algunas comprobaciones iniciales que verifiquen que todo es correcto antes de comenzar la evaluación, y se mostrará un mensaje con las operaciones que se

van a revisar. En caso de encontrar algún error o que el profesor pulse el botón “no”, se volverá al formulario; si no, comenzará la ejecución del programa.

4.2 Funcionamiento del software

Este apartado está dedicado a la explicación del funcionamiento del programa: se desarrollarán las funciones de las que hace uso, las comprobaciones para la evaluación, los archivos que crea, etc.

El programa está organizado en tres reglas: dos de ellas destinadas a la selección de los archivos y una tercera en la que se realiza la evaluación. No se incluirá en este apartado el código completo del programa, ya que el objetivo de la sección es ayudar a comprender la lógica del conjunto y el código puede ser difícil de entender. El código completo se puede consultar en los anexos junto con un esquema para ayudar a clarificar la estructura del programa

4.2.1 Regla: “EvaluateModel”

Esta es la regla más importante del software, la encargada de realizar la evaluación. Primero se explicará la función “main” y después cada función específica.

La regla se activa cuando se pulsa el botón “Evaluate Model “del formulario. Lo primero que hace el programa es cargar la lista que contiene el nombre de todos los archivos de la carpeta que se evaluará, crear las variables necesarias para el correcto funcionamiento del programa e identificar los archivos como “ref” y “eval”.

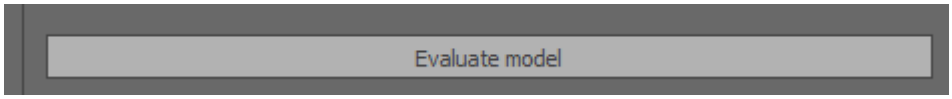

Ilustración 32 Botón del formulario

Llegados a este punto se empiezan las comprobaciones previas a la evaluación. Se utiliza la función “PreviousReviews(…)”, que hace algunas comprobaciones para asegurar el correcto funcionamiento de programa (como por ejemplo, que exista un documento de referencia y uno de evaluación, y no más). Al finalizar las comprobaciones se muestra un mensaje con las operaciones que el usuario ha escogido evaluar.

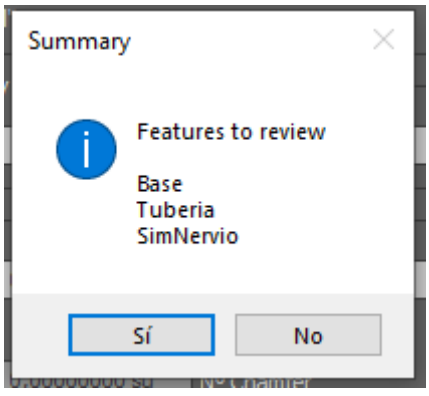

Ilustración 33 Mensaje resumen del programa

Comienzan las comparaciones generales del modelo. La primera comprobación que hace es la colocación de los ejes con la función “LocatePart(…)”. El siguiente paso es calcular el volumen de interferencia entre las piezas utilizando la función “interf(…)”. Con la función “CheckTotalVolume(…) se comprueba si las dos piezas coinciden en volumen. Por último con la

función “CountFeature(…)” y “CountFeatureByType(…) “ se cuenta el número total y de cada tipo de operaciones que contiene cada modelo.

La siguiente sección de código se dedica a comprobar las operaciones seleccionadas por el profesor. Primero se comprueba si la operación existe, ya que el alumno puede no haber creado la operación o no haberle puesto el nombre adecuado. Con cada operación se hacen tres comprobaciones básicas: con la función “EvaluateFeature(…)” se comprueba el tipo de operación utilizado; con la función “VolumeDiference(…)” se verifica que el volumen que añade dicha operación a la geometría total es la misma en los dos modelos; y por último, “SketchConstraints(…)” comprueba si los bocetos consumidos por la operación estaban correctamente restringidos.

Una vez realizadas todas las comprobaciones, el último paso es recoger todos los datos obtenidos para puntuar el modelo y pasar los resultados a un documento Excel. Utilizando la función “FinalPunctuation (…)” se asigna una nota a cada parte evaluada del modelo, y con la función “AddtoExcelFile(…)” se pasan al documento en el que el profesor podrá consultarlos.

Finalmente se prepara el siguiente documento a evaluar y se borra el elemento ya evaluado de la lista de piezas con la función “RemoveListElement (…)”. Se carga la siguiente con la función “deletepart(…)”. Repite la ejecución del programa hasta terminar todos los archivos de la lista. Al finalizar se muestra un aviso.

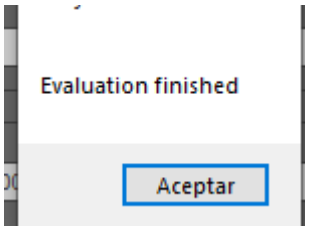

Ilustración 34 Mensaje final del programa

PreviousReviews(…) :

Esta función es la encargada de realizar la revisión antes de empezar a comparar los modelos. Para funcionar necesita como argumentos un parámetro que le indique el número de ficheros que hay en el documento y un booleano que le indica si es el primer archivo que se evalúa. Conocido el número de archivos del documento, la función podrá comprobar si el usuario ha cargado los archivos necesarios. La función llama a otra, “SummaryMessage()”, en la que se recogen toda la información de las operaciones seleccionas . De esa función se extrae una variable compartida llamada “FeatCounter”, que servirá para comprobar que el usuario ha seleccionado al menos una operación que revisar. En caso de que alguna de estas comprobaciones proporcione un valor de error o negativo, la función devolverá un valor de cero; lo que significará que cesará la ejecución del programa y volverá al formulario.

SummaryMessage(…)

Esta función es llamada por “PreviousReviews(…)” y requiere como argumento la variable booleana “firstrun” para cambiar el valor a falso después de la primera ejecución. En esta función se crean tres variables compartidas que serán de utilidad más tarde:

SFeaturesNames, una variable string que contiene los nombres de las operaciones.

FeatCounter, una variable tipo entero que contienen el número de operaciones a evaluar.

FeatureWeight, una variable tipo doble que guarda el peso que el usuario ha asignado a cada operación.

Por último, se llama a la función CreateNewExcel(…).

LocatePart()

Esta función tiene como objetivo asegurar que la pieza está correctamente orientada, ya que es posible que un alumno coloque mal los ejes y eso provocaría errores en futuras comprobaciones. Esta función hace uso de otras funciones sencillas:

Rotate() :Se le indica como argumento un eje y rota 90 grados entorno al mismo.

Interf(): Calcula y devuelve como valor la interferencia entre las dos piezas.

InertiaAsem(): Calcula los momentos de inercia del ensamblaje y los devuelve en un vector double.

Puede parecer que no tiene sentido crear una función independiente para cada uno de estos procesos debido a su simplicidad, pero dado que son procesos que se repiten una gran cantidad de veces a lo largo de la función, ayuda a conseguir un código más aseado y fácil de entender.

El funcionamiento de “LocatePart” se basa en comparar los momentos de inercia de las dos piezas y el momento de inercia total del ensamblaje. La función realiza una serie de iteraciones en las que compara el eje con mayor momento de inercia de la pieza de referencia con el eje de mayor momento de inercia del modelo a evaluar, y rota la pieza hasta hacerlos coincidir. Después repite el mismo proceso, pero con el mínimo. Antes de empezar esa correcciones comprueba que los ejes estén correctamente colocados y de no estarlo añade un mensaje a la variable de texto creada con ese fin.

A continuación, se muestra un extracto de código para ayudar a entender mejor el funcionamiento. Concretamente, este fragmento muestra el caso en el que el eje de máxima inercia de la referencia es X y el del modelo es Y.

```
While (xok = False And yok = False) Or (xok = False And zok = False) Or (yok = False And zok
'Search the max inertia axis of each part and make them coincide
'Star searching max on the ref
    If Ixxref = MaxOfMany(Ixxref, Iyyref, Izzref)
    'If the max is on X axis search the max on eval
        If Ixxeval = MaxOfMany(Ixxeval, Iyyeval, Izzeval)
    'If the maxim values is also on X this axis is ok
        xok = True

        Else If Iyyeval = MaxOfMany(Ixxeval, Iyyeval, Izzeval)
        'If max is on y rotate eval part arround z axis until they are on the same position,
            dif = IxxAsem - (Ixxref + Iyyeval)
            'The value must have an error < 0.005
            While dif>0.005
                Rotate("Z Axis")
                CenterPieces()
                'The value of inertia assembly must be actualiced on ech iteration.
                momentAsem=inertiaAsem()
                IxxAsem = momentAsem(0)
                IyyAsem = momentAsem(1)
                IzzAsem = momentAsem(2)
                IxxAsem = Round(IxxAsem,3)
                IyyAsem = Round(IyyAsem,3)
                IzzAsem = Round(IzzAsem, 3)
                dif= IxxAsem - (Ixxref + Iyyeval)

            End While
            xok = True
```

Ilustración 35 Fragmento de código

Partiendo de X como eje de mayor inercia de la referencia, comprueba cuál es el del modelo a evaluar; en caso de ser X, no es necesario realizar ningún giro. En cambio, si se tratara de Y o Z habría que rotar la pieza hasta hacer coincidir ambos ejes, el de máxima inercia de cada modelo. Para detectar que se cumple esta condición se calcula la diferencia entre la inercia en X del ensamblaje y la suma de los dos momentos máximos de cada pieza.

Esta diferencia debería ser cero en caso de estar alineados, pero puesto que el programa utiliza gran cantidad de decimales, se ha decidido contemplar un error de 0.005. Si esta condición no se cumple, se hace rotar la pieza y se hacen coincidir sus centros de masa, se actualizan los valores del momento del ensamblaje y se vuelve a realizar la comprobación de la diferencia.

Estas líneas de código se repiten de forma parecida para todos los casos. De forma similar se alinean los ejes de mínima inercia.

Llegados a este punto los ejes están alineados, pero podrían estar invertidos. Para eliminar esa posibilidad se utiliza el cálculo de la interferencia: se calcula el volumen de interferencia para cada posición después de aplicar un giro de 180 grados en torno a cada eje. Finalmente, la pieza se deja colocada en la posición en la que la interferencia ha sido mayor.

interf(...)

Calcula el volumen de interferencia entre ambas piezas y comprueba si coincide con el volumen de la pieza. Esta función lleva el mismo nombre que la llamada desde “LocatePart(…)” pero a diferencia de la anterior, ésta necesita un argumento string donde guarda un mensaje informando acerca del resultado . Devuelve un porcentaje de coincidencia entre modelos. Además, crea una variable compartida que será la que indique el grado de coincidencia y la que se utilizará para calcular la puntuación.

VolumeDiferenceForAll(…) y VolumeDiference(…)

Calcula el volumen que aporta cada operación en ambos modelos y lo compara; en caso de que no coincidan, suma uno al contador de operaciones erróneas. Esta función no afecta a la

puntuación final, sino que se utiliza solo de forma informativa; añade un comentario con las diferencias encontradas para que el profesor disponga de esa información.

ActiveAllFeatures(…)

Se utiliza esta función para activar operaciones que hemos desactivado previamente. Se utiliza en las funciones "VolumeDiferenceForAll" y "VolumeDiference" para volver a activar las operaciones antes de terminar la operación

DeactivateFeatures()

Esta función sirve para desactivar operaciones. Se utiliza en las funciones "VolumeDiferenceForAll(…)" y "VolumeDiference(…)" para desactivar las operaciones y poder calcular los volúmenes antes y después de la operación.

```
Function DeactivateFeatures(sName As String, sfeatu As String, bPosition As Boolean,Doc As Document)
    Dim oFeature As PartFeature
    Dim oFeatures As PartFeatures
    Dim Asem As AssemblyDocument
    Asem = ThisApplication.ActiveDocument
    oFeatures = Doc.ComponentDefinition.Features
        i = 0
        For Each oFeature In oFeatures
            Feature.IsActive(sName, oFeature.Name) = False
            If oFeature.Name.Equals(sfeatu)
                Feature.IsActive(sName, oFeature.Name) = bPosition
                 i=1
            Else If oFeature.Name<>sfeatu And i=0
                Feature.IsActive(sName, oFeature.Name) =True
            Else If i=1
                Feature.IsActive(sName, oFeature.Name) =False
            End If

        Next


End Function
```

Ilustración 36 Fragmento de código

CheckTotalVolume(…)

Esta función extrae el valor del volumen de ambas piezas y los comprara. Devuelve un porcentaje según el grado de coincidencia de los modelos. También crea una variable compartida que indica la diferencia de volumen en forma de porcentaje. Esta variable servirá para calcular la puntuación final.

CountFeature(…) y CountFeatureByType(…)

La primera cuenta el número total de operaciones, la segunda el número total de operaciones de un tipo específico.

FeatureEntitiesCount()

Está función calcula el número de entidades a las que se les ha aplicado un tipo de operación. Esta función es necesaria debido a que varias operaciones iguales se pueden aplicar todas a la vez con una sola operación o de forma individual, por lo que contar las operaciones aplicadas no nos da toda la información necesaria acerca de estas características del modelo.

Acceder a esta información es más complicado de lo que parece dado que para cada operación se deberá acceder a un elemento propio de la misma, por ejemplo, en el caso de agujeros serán los centros, en el caso de los redondeos serán las caras generadas.

CheckFeatureExist(…)

La misión de esta función es comprobar que las operaciones seleccionadas para la evaluación en detalle existen en el modelo a evaluar. Devuelve un valor de -1 en caso de que no exista. Está

implementada en el sub main, de forma que si no encuentra la función, automáticamente asigna un valor de -1 a las posteriores verificaciones.

EvaluateFeature(…)

Comprueba dos operaciones con el mismo nombre, una en cada modelo, que comparten el mismo tipo de operación. La función incluye una casuística de intercambios de operaciones no válidos (como por ejemplo hacer una extrusión negativa en lugar de un agujero). En caso de que el tipo de operación no sea válido devuelve un valor de -1; si el tipo coincide, devuelve valor de 0; y si es distinto, pero no está incluida en una de las excepciones, devuelve un valor de 1.

En el siguiente fragmento de código se muestran las excepciones tenidas en cuenta.

```
If sEvaltype = sReftype

    Return 0
'Some feature changes are 100% wrong, returns -1
Else If sReftype.Contains("Hole") And sEvaltype.Contains("Extru")
    sFinalText=sFinalText & sFeatureName & ":  The feature type is wrong;"
    Return -1
Else If sReftype.Contains("Chamfer") And sEvaltype.Contains("Extru")
    sFinalText=sFinalText & sFeatureName & ":  The feature type is wrong;"
    Return -1
Else If sReftype.Contains("Rib") And sEvaltype.Contains("Extru")
    sFinalText=sFinalText & sFeatureName & ":  The feature type is wrong;"
    Return -1
Else If sReftype.Contains("Stiffener") And sEvaltype.Contains("Extru")
    sFinalText=sFinalText & sFeatureName & ":  The feature type is wrong;"
Return -1
'For the rest of options returns 1; for example, it is correct to do a revolution or a extrusio
Else
    sFinalText=sFinalText & sFeatureName &":  The used feature is diferent from the reference;"
    Return 1
End If
```

Ilustración 37 Fragmento de código

SketchConstraints(…)

Con esta función se verifica que el boceto consumido por la operación que se está evaluando está completamente restringido. Además, también se comprueba si el tipo de restricción usado por el alumno es el mismo que las del profesor. Si el boceto está completamente restringido, pero no igual, el alumno no recibirá la nota completa del apartado. Si además solo ha utilizado restricciones de tipo dimensional (cotas), el apartado recibirá puntuación de 0 (a no ser que el profesor también haya utilizado solamente restricciones de este tipo). También recibirá puntuación de 0 en el apartado si el boceto no está completamente restringido.

CenterPieces()

El cometido de la función “CenterPieces()” es conseguir que la pieza de referencia y la pieza a evaluar estén situadas en el mismo punto, independientemente de si el centro de coordenadas de la pieza esta desplazado. Esto se soluciona haciendo coincidir el centro de gravedad de las piezas con el centro de coordenadas del ensamblaje. Para ello será necesario crear una matriz de transformación que permitirá trasladar las piezas de un punto a otro y un vector que indicará a la matriz la dirección del movimiento. Utilizando las funciones de iproperties, propias de Autodesk Inventor, se extrae el centro de gravedad de cada pieza del ensamblaje.

FinalPunctuation(…)

Recoge todos los resultados de las funciones anteriormente explicadas y calcula la nota del modelo y de cada parte evaluada. La evaluación de cada operación se realiza en función de tres factores: el boceto, el tipo de operación utilizado y la geometría conseguida. Por defecto se

reparte esta puntuación de forma equitativa entre los tres factores. Además, según el valor que devuelva la función el modelo recibirá la nota total del apartado, la mitad o nada. El profesor puede acceder a esta función y modificar estos porcentajes si lo desea.

Para la puntuación de la valoración general se multiplicarán los porcentajes de similitud entre los modelos extraídos de las funciones "interf(…)" y "CheckTotalVolume(…)" y a su vez estos por la puntuación asignada a esta parte de la evaluación.

```
Dim geo = 34
Dim sketch = 33
Dim type = 33
Dim medio = 0.5
Dim Ok = 1
Dim NOK = 0
```

Ilustración 38 Fragmento de código

TakeFeatureType()

Recoge el tipo de una operación y lo devuelve como string.

OccurenceName ()

Simplemente se trata de una función auxiliar, que extrae el nombre que tiene asignada la pieza en el árbol del modelo ya que algunas funciones necesitan este nombre en lugar de el documento para funcionar.

CreateNewExcel(…)

Crea un documento Excel rellena el cajetín, creando la estructura necesaria para después introducir los datos de la evaluación.

File name	Feature: Base	Feature: Teton	Feature: Tuberia	Feature: Oreja	General	Other Features	Calification
	Weight 25	Weight 10	Weight 25	Weight 15	Weight: 15	Weight: 10	100

Ilustración 39 Estructura del excel

Student: Se guardará el nombre del usuario que crea la pieza.

FileName: Nombre del archivo que se evalúa.

Feature: Nombre de las operaciones que se van a evaluar. Se crea una columna para cada operación y se indica el nombre de ésta.

Weight: Añade el peso de cada operación y lo coloca en la columna correspondiente.

General: En esta casilla se incluirá el resultado de los chequeos generales del modelo.

Other Features: La columna contendrá la puntuación asignada a otras operaciones

Calification: La columna servirá para colocar la nota final que le asigne el programa al modelo.

Failures found: Esta columna contendrá los errores que el programa ha detectado en el modelo, separados por";".

La creación del Excel se muestra en el siguiente fragmento de código

```
Dim colum As Char
Dim Box As String
'To creates the new document on Desktop with the name of the reference model uncomment
'spath = "C:\Users\Usuario\Desktop\" + spath
'If not the file will be saved on the same folder dan the reference
'Get the Inventor user name From the Inventor Options
spath = SharedVariable("Path")
iFeatCounter=SharedVariable("FeatCounter")
'define Excel Application object
excelApp = CreateObject("Excel.Application")
'set Excel to run visibly, change to false if you want to run it invisibly
excelApp.Visible = False
'suppress prompts (such as the compatibility checker)
excelApp.DisplayAlerts = False
'check for existing file
If Dir(spath) <> "" Then
'workbook exists, open it
excelWorkbook = excelApp.Workbooks.Open(spath)
'set the first sheet active
excelSheet = excelWorkbook.Worksheets(1).activate
Else
'workbook does NOT exist, so create a new one
excelWorkbook = excelApp.Workbooks.Add
```

Ilustración 40 Fragmento de código

AddtoExcelfile(…)

Recoge los resultados de toda la evaluación y los escribe en el Excel. El fragmento de código que se muestra a continuación es un ejemplo de cómo escribir en el Excel usando ilogic.

```
With excelApp
    Name = "A" & excelline.ToString
    Filename = "B" & excelline.ToString
    .Range(Name).Select
    .ActiveCell.Value = iProperties.Value(OccurenceName, "Summary", "Author" )
    .Range(Filename).Select
    .ActiveCell.Value = IO.Path.GetFileName(SharedVariable("PathList")(0))
    colum = "C"
    iFeatCounter=iFeatCounter-1
    For counter As Integer = 0 To iFeatCounter
        SBox = colum & excelline.ToString
        colum = Chr(Asc(colum )+ 1)
            .Range(SBox).Select
            .ActiveCell.Value = SharedVariable("Punctuations")(counter)
    Next
    SBox = colum & excelline.ToString
    .Range(SBox).Select
    .ActiveCell.Value = sharedVariable("General")
    colum = Chr(Asc(colum) + 1)
    SBox = colum & excelline.ToString
    SBox = colum & excelline.ToString
    .Range(SBox).Select
    .ActiveCell.Value = FinalText
    FinalText=""
End With
```

Ilustración 41 Fragmento de código

Deletepart()

Se utiliza para eliminar la pieza ya evaluada del ensamblaje. Elimina la pieza y carga la siguiente de la lista.

FileList()

Se utiliza para crear una lista que contendrá la ruta de todos los archivos de la carpeta a evaluar. Utiliza una variable compartida que se crea en la regla "Select file to evaluate", que contiene la ruta del primer archivo seleccionado. Así se consigue que el profesor no tenga que seleccionar los archivos uno por uno.

RemoveListElement ()

Elimina los elementos ya evaluados de la lista para poder acceder al siguiente archivo.

4.2.2 Regla: Select reference file

En esta regla se escogerá el archivo de referencia, es decir, el modelado por el profesor, y se crearán los parámetros y variables necesarios para ejecutar correctamente el programa y para hacer funcionar el formulario.

La regla se activará cuando el usuario seleccione pulse el botón "Select Reference file" en el formulario.

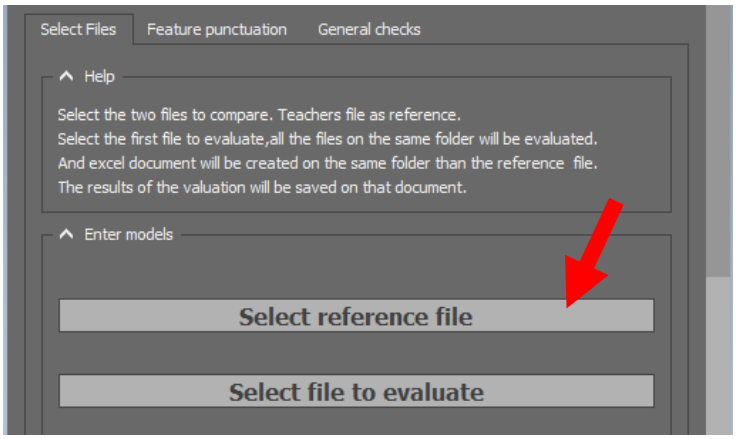

Ilustración 42 Captura del formulario

El formulario no se podrá completar antes de seleccionar este archivo, aparecerá de la siguiente forma:

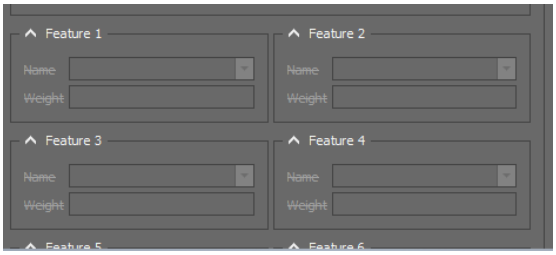

Ilustración 43 Captura del formulario

Ilustración 44 Captura del formulario

Igual que con la regla anterior, en este apartado primero se explicará la función Sub Main(), y después cada función utilizada en detalle.

Main()

Cuando se active la regla aparecerá un cuadro de diálogo de buscador de Windows, y se podrá seleccionar el archivo deseado.

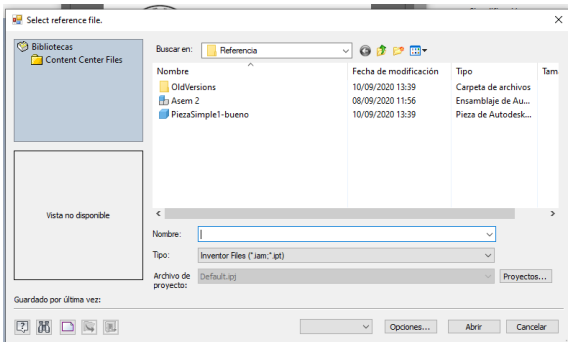

Ilustración 45 Captura del formulario

Cuando el archivo esté seleccionado, el programa cargará automáticamente la pieza en el ensamble y toda la información del modelo. El siguiente paso es crear los parámetros necesarios para el formulario ”Main()“; llama a la función ”CreateSelector()“ y termina la ejecución.

CreateSelector()

En este punto, lo primero que hace la función es guardar en una lista los nombres de todas las operaciones de la referencia, para luego crear diez parámetros multivalor que aparecerán en el formulario en forma de selector. La estructura para la creación de parámetros de usuario es la siguiente:

```
oParam = ThisDoc.Document.ComponentDefinition.Parameters("Feature1")
oNewParam = userParams.AddByValue("Feature1","None", UnitsTypeEnum.kTextUnits)
MultiValue.SetList("Feature1", adaptiveStringList.ToArray)
MultiValue.UpdateAfterChange = True
```

Ilustración 46 Fragmento de código

Después creará todos los parámetros numéricos del formulario, las puntuaciones y pesos. La estructura es similar a la anterior.

FeatureEntitiesCount()

Está función cuenta las entidades del archivo de referencia para que se muestren en el formulario y posteriormente poder compararlas con las del archivo del alumno.

4.2.3 Regla: Select file to evaluate

Esta regla es muy similar a la regla anterior. Cuando se pulsa el botón en el formulario se abre un cuadro de diálogo de Windows y el profesor selecciona el primer archivo que se evaluará. El archivo se carga automáticamente en el ensamblaje.

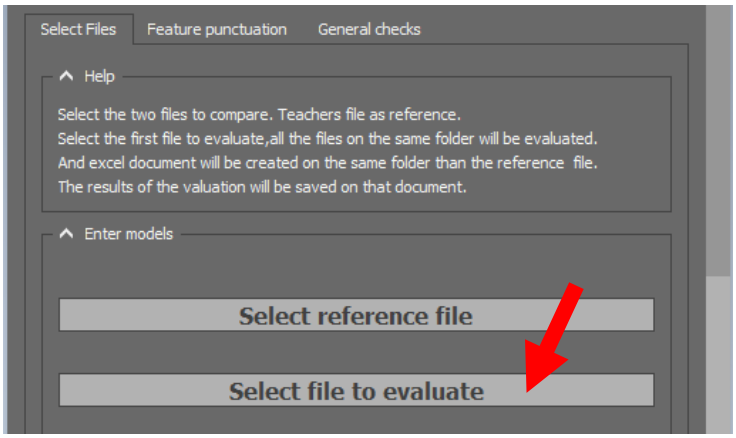

Ilustración 47 Captura del formulario

Antes de terminar la ejecución de la regla se crea una variable compartida que contiene la ruta del archivo. Esta variable se cargará en la regla “Evaluate Model” para poder crear la lista de archivos que se van a corregir.

5. Ejemplos de Aplicación

5.1 Pieza “Modelo prueba 1”

La primera prueba del programa se realiza con tres piezas modeladas por el profesor; una considerada solución perfecta, y dos con distintos fallos (una peor modelada que la otra).

A continuación, se hará una descripción detallada de las piezas que se evaluarán, de los fallos que debe detectar el software en dichas piezas y de la referencia. También se indicarán las condiciones que se le darán al programa para la evaluación, es decir, las puntuaciones y operaciones que se evaluarán.

5.1.1 Pieza de referencia

La pieza de referencia es la siguiente:

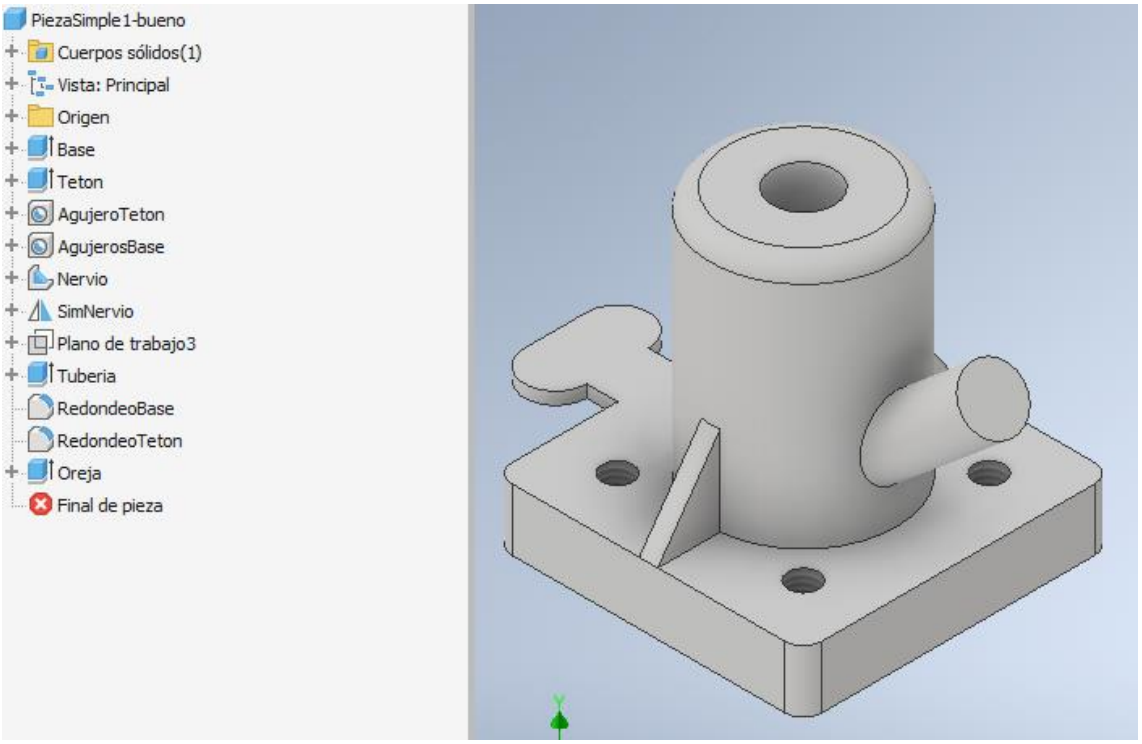

Ilustración 48 Pieza de referencia

Todas las operaciones tienen un nombre asignado que las identifica, todos los bocetos utilizados están completamente restringidos y las operaciones realizadas son las idóneas para la pieza.

Las operaciones seleccionadas para la evaluación serán aquellas que no son “básicas” , es decir, extrusiones, revoluciones… Serán Base, Tetón, Tubería y Oreja.

5.1.2Piezas modelo

Se han creado tres versiones de la pieza para realizar la evaluación. La primera versión es una copia exacta de la pieza, por lo que el programa no debería detectar ningún fallo en esta pieza. Las otras dos piezas contienen distintos fallos, tanto de dimensiones como de intención de diseño. La llamada “piezasimple-mal” tiene errores más graves, por lo que debe obtener peor

nota que las otras dos. A continuación, se mostrarán los errores que contiene cada pieza y que debe detectar el programa.

"Piezasimple-medio"

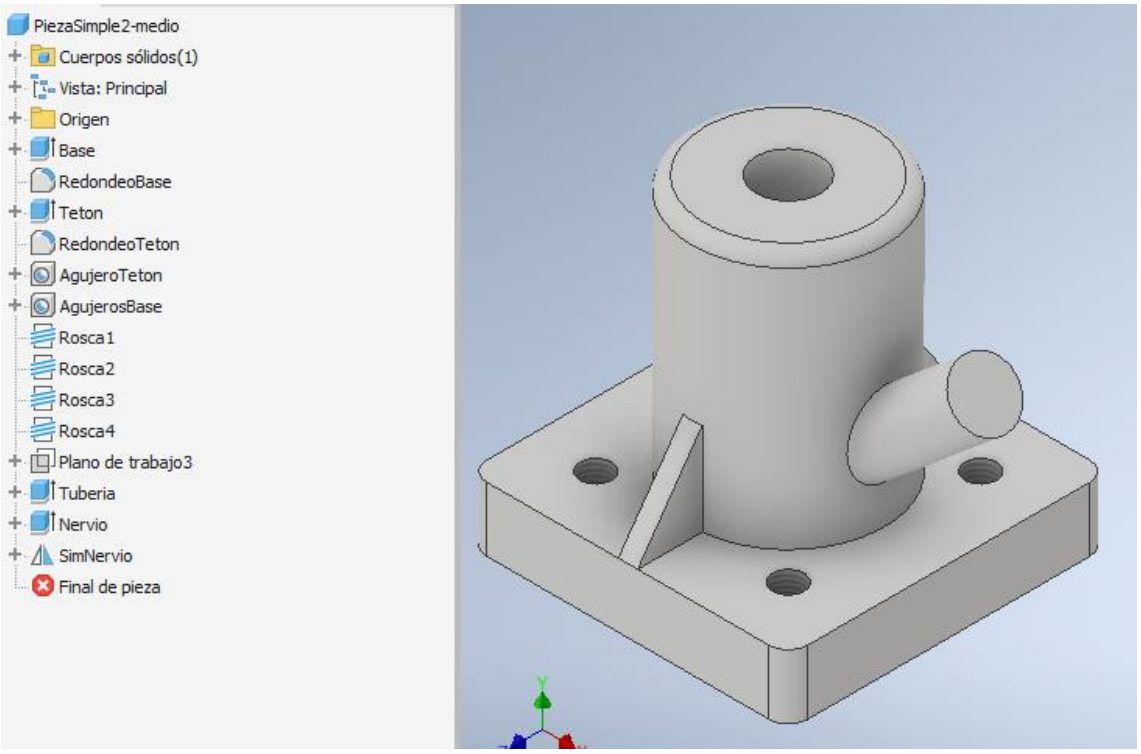

Ilustración 49 Pieza simple-medio

El primer fallo que se observa es el orden de las operaciones: las operaciones de redondeo y chaflan deben realizarse al final del modelo, ya que son operaciones sencillas. En caso de necesitar realizar alguna modificación sobre esta operación, todas las operaciones posteriores se pueden ver afectadas, teniendo así que rehacer operaciones complejas que consumen mucho tiempo de realización.

Otro de los fallos que se pueden encontrar a simple vista es que han realizado agujero y rosca por separado, en lugar de agujero roscado. También se ha realizado la operación nervio con una extrusión. Finalmente, se puede ver que no están todas las operaciones que contiene el documento de referencia.

"Piezasimple-mal"

El fallo más destacable que vemos en el modelo 'mal' es que los ejes no están colocados correctamente.

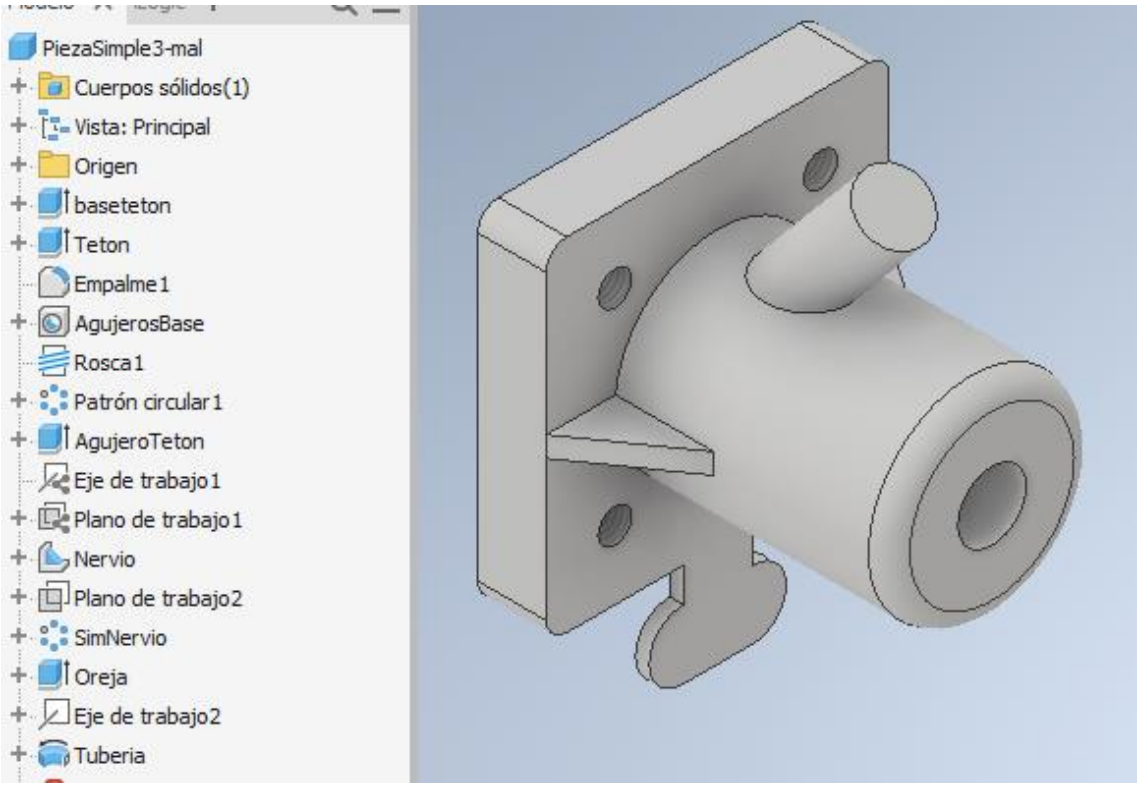

Ilustración 50 Pieza simple- mal

Hay que fijarse en el árbol del modelo para detectar otras diferencias. El error que llama más la atención es que ha realizado un patrón circular en lugar de una simetría, y un barrido en lugar de una extrusión para la tubería. Además de utilizar operaciones de rosca en lugar de agujeros roscados. También se puede observar que las operaciones de empalme no se han realizado al inicio del modelo. Igualmente, consultando cada operación más detalladamente se puede observar que algunos bocetos no están completamente restringidos.

5.1.3 Configuración del programa

Para la corrección de estas piezas el programa se ha configurado de la siguiente forma:

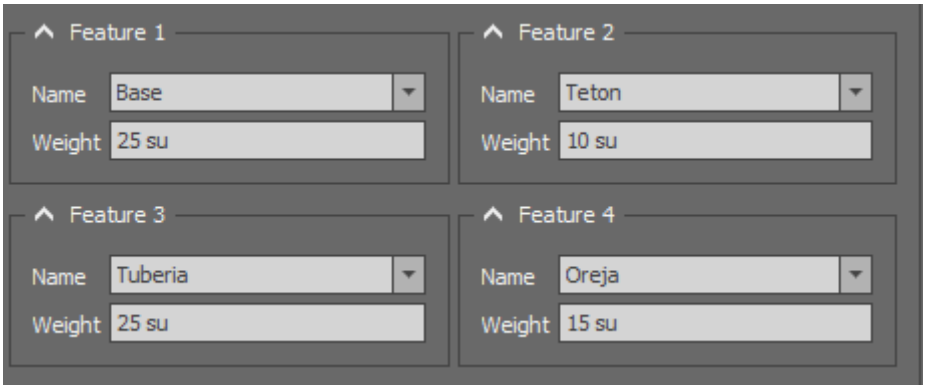

Ilustración 51 Configuración del formulario

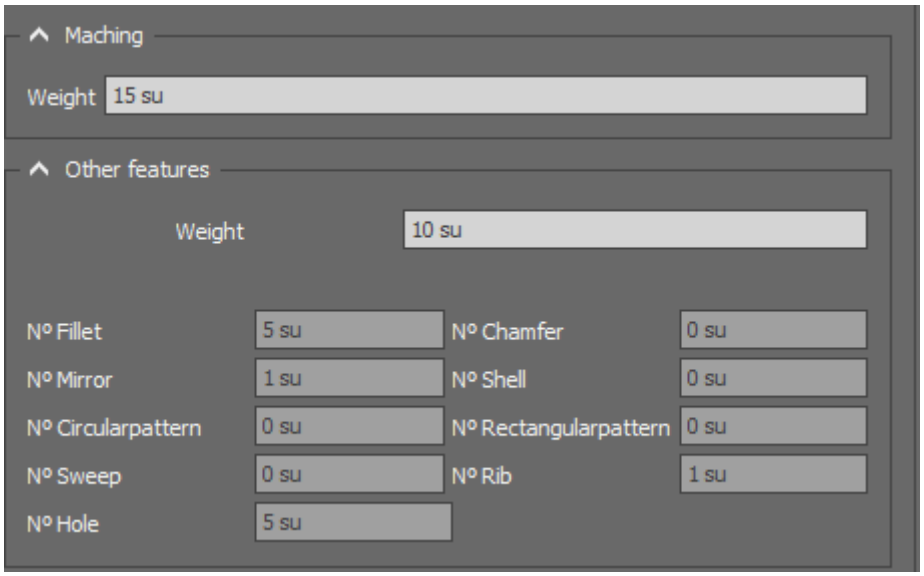

Ilustración 52 Configuración del formulario

5.1.4 Resultados obtenido

Se ha eliminado la primera columna para mantener el anonimato de los autores.

La primera pieza se trata de una pieza completamente distinta introducida en la carpeta para asegurar el comportamiento del programa en ese caso.

File name	Feature: Base	Feature: Teton	Feature: Tuberia	Feature: Oreja	General	Other Features	Calification
	Weight 25	Weight 10	Weight 25	Weight 15	Weight: 15	Weight: 10	100
Distinto.ipt	0	0	0	0	0	25	2,5
PiezaSimple1-Diez.ipt	100	100	100	100	100	100	100
PiezaSimple2-medio.ipt	83,5	100	100	0	96,45561903	91,66666667	79,51000952
PiezaSimple3-mal.ipt	0	83,5	16,5	33	99,99769716	41,66666667	36,59132124

Ilustración 53 Captura de Excel

Además el Excel consta de una última columna de cometarios en el que se guardan los resultados obtenidos.

"Modelo-Diez"

El modelo que coincide a la perfección por el realizado por el profesor ha obtenido un 100% sin ningún comentario adicional.

"Modelo-Medio"

Ha obtenido como calificación final casi un ocho sobre diez. Los comentarios de errores son los siguientes:

Resultados de las comprobaciones generales:

- La Interferencia de las piezas no es correcta;
- Redondeo Base: El volumen no coincide;
- Redondeo Tetón: El volumen no coincide;
- Oreja: La operación no existe;
- El volumen total no coincide;
- Ha utilizado mayor número de operaciones;
- Distinto número de nervios;
- Redondeo Base: Mala prioridad de operaciones;

- Redondeo Tetón: Mala prioridad de operaciones;

Resultados de las comprobaciones de las operaciones:

- Base: Las restricciones no son las mismas;
- Oreja: No existe;

"Modelo-Mal

El modelo mal ha recibido la peor nota que el modelo medio: concretamente un 36.59%.

Resultados de las comprobaciones generales:

- La interferencia de la pieza no coincide;
- Ejes mal orientados;
- Base: La operación no existe;
- Agujero Teton: El volumen es distinto;
- Agujeros Base: El volumen es distinto;
- Nervio: El volumen es distinto;
- Tubería: El volumen es distinto;
- Redondeo Base: La operación no existe;
- Redondeo Teton: La operación no existe;
- Oreja: El volumen es distinto;
- El volumen total es distinto;
- Mayor número de operaciones que en la referencia;
- Empalme1: Mala prioridad de operaciones;
- Distinto número de agujeros;
- Distinto número de patrones circulares;
- Distinto número de espejos;

Resultados de las comprobaciones de las operaciones:

- Base: No existe;
- Teton: Restricciones no coinciden;
- Tubería: La operación utilizada es distinta;
- Tubería: El volumen es distinto;
- Tubería: No está completamente restringido
- Oreja: El volumen no coincide
- Oreja: No está completamente restringido;

"Pieza Distinta"

La pieza distinta ha recibido un 2,5 %, es decir, un 0.25 sobre diez. El resultado es adecuado ya que la pieza no guarda apenas similitudes con el modelo de referencia. El único apartado en el que ha recibido puntuación es el que comprueba el número de operaciones secundarias, en el que no ha encontrado suficientes diferencias como para asignarle un 0 **(**y sí contienen el mismo número de alguna de estas ,0).

5.2 Pieza "Modelo Prueba 2"

Para esta pieza se les ha enviado un enunciado adaptado a una serie de exalumnos de la asignatura para que la modelen según su propio criterio. Se realiza una segunda comprobación

para asegurar que el programa se comporta correctamente al introducirle una pieza no modelada por una persona conocedora del programa, por lo que los fallos que puedan tener las piezas no están “preparados” para que el programa los detecte.

5.2.1 Referencia y modelos autores anónimos

Referencia

Como pieza de referencia se ha escogido un modelo de examen y se ha simplificado. A los exalumnos se les han dado algunas indicaciones adicionales a las habituales en el examen, como los nombres de las partes y la importancia de la posición de los ejes.

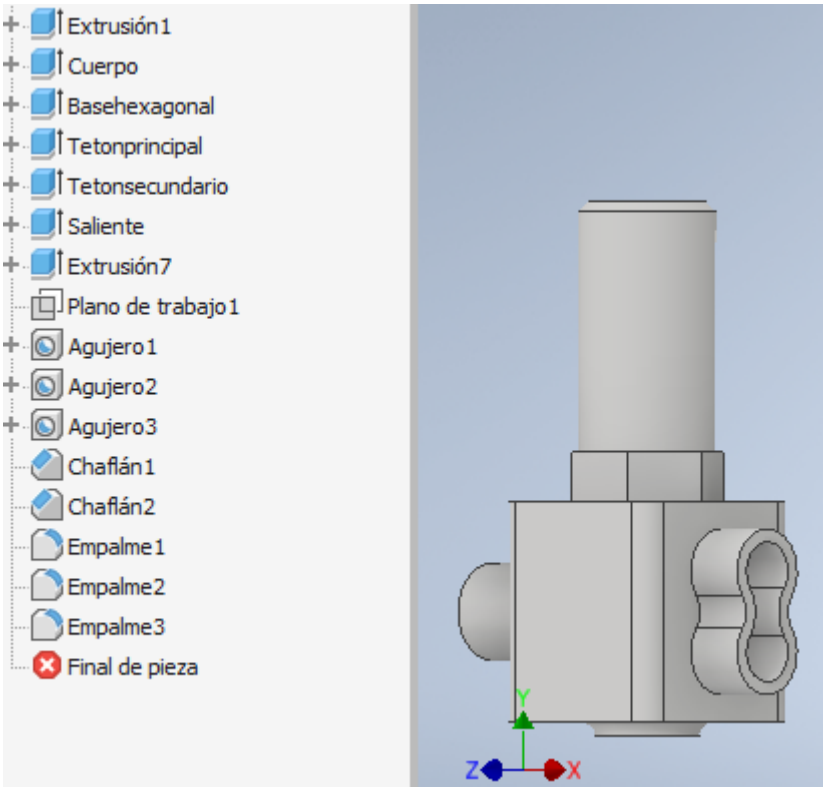

Ilustración 54 Pieza de referencia

Modelo prueba 1

La primera discrepancia que se encuentra respecto a la referencia es el nombre de las operaciones. Al no asignar el nombre correcto la nota de este modelo será muy baja ya que no localizará las operaciones que debe corregir.

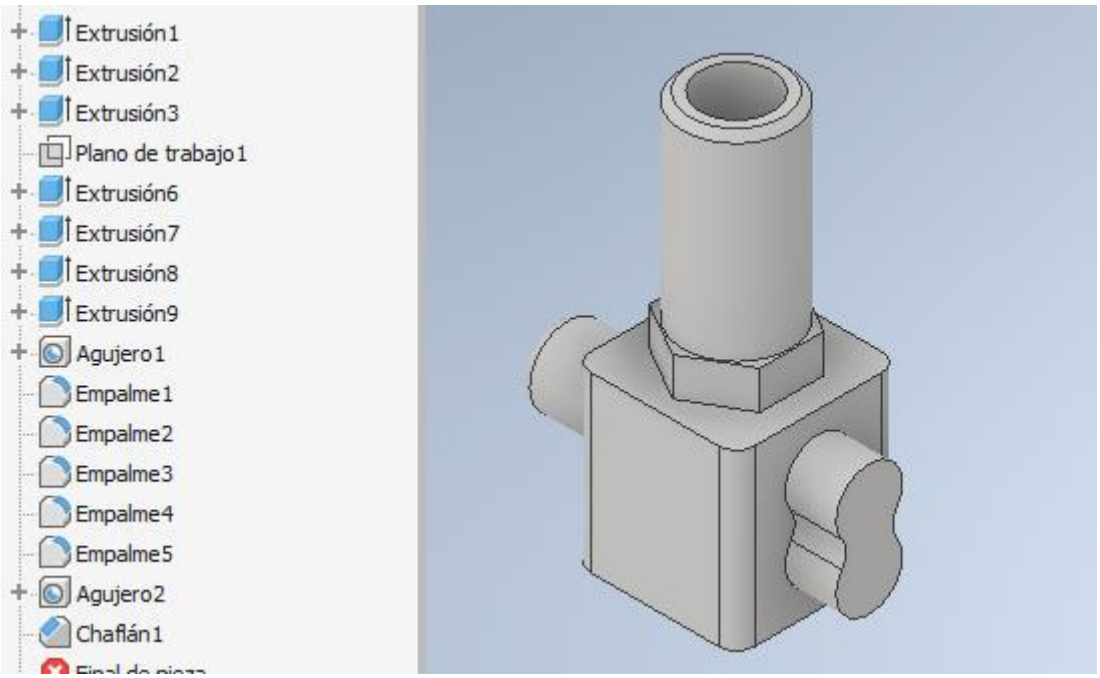

Ilustración 55 Pieza del alumno 1

Modelo prueba 2

Aparentemente el fallo más llamativo de este modelo es la orientación de los ejes. El programa indicará otros fallos que contiene el modelo.

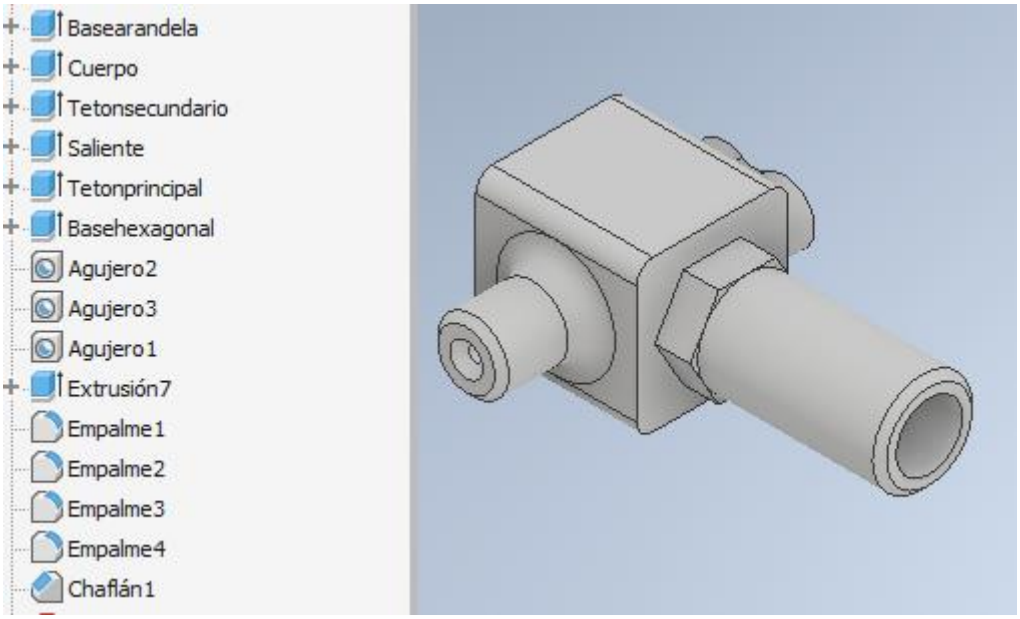

Ilustración 56 Pieza alumno 2

5.2.2 Configuración del programa

Se ha establecido la siguiente configuración:

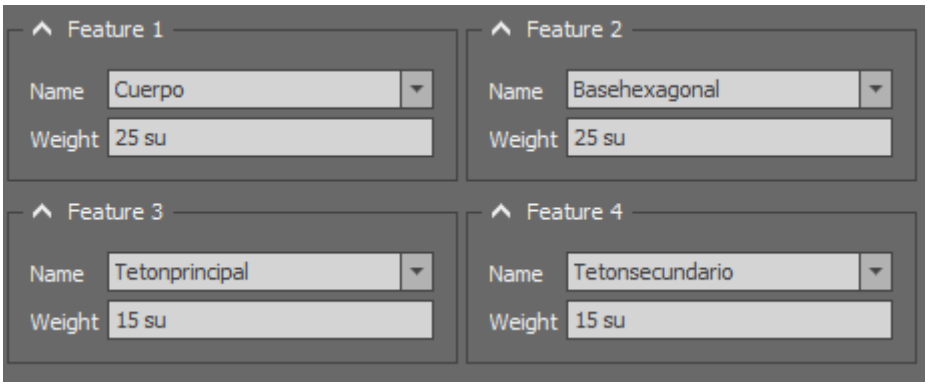

Ilustración 57 Configuración del formulario

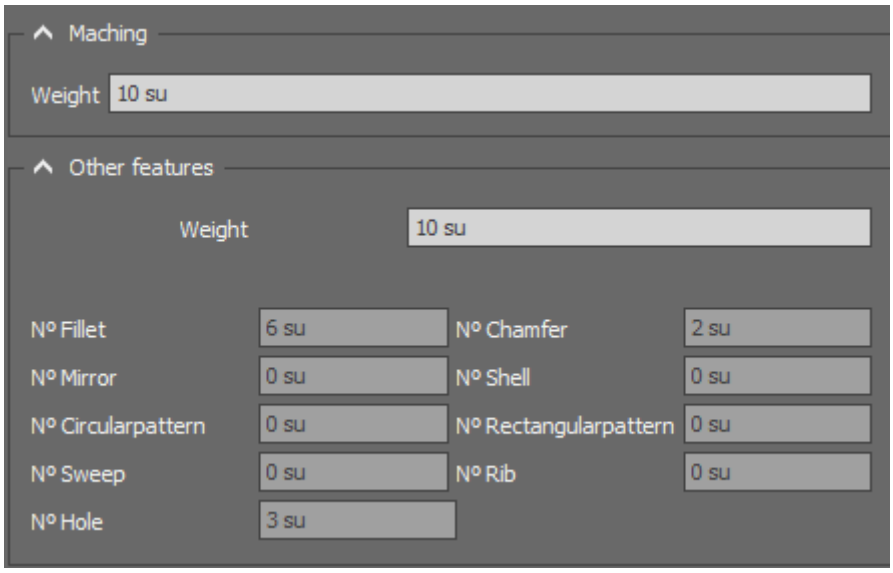

Ilustración 58 Configuración del formulario

5.2.3 Resultados obtenidos

Feature: Cuerpo	Feature: Basehexagonal	Feature: Tetonprincipal	Feature: Tetonsecundario	General	Other Features	Calification
Weight 25	Weight 25	Weight 15	Weight 15	Weight: 10	Weight: 10	
0	0	0	0	92,02068859	63,63636364	15,56570522
83,5	33	49,5	33	92,20343341	81,81818182	58,90216152
100	100	100	100	100	100	100

Ilustración 59 Captura del Excel

Como era de esperar la pieza perfecta ha obtenido un 10 (el último de la lista).En los otros dos modelos ha encontrado una serie de fallos.

Como era de esperar, la pieza perfecta ha obtenido un 10 (el último de la lista). En los otros dos modelos ha encontrado una serie de fallos.

Modelo prueba 1

Ha obtenido una nota de 1,55 sobre diez. Esto es debido principalmente a que no ha nombrado correctamente las operaciones.

Los fallos detectados por el programa son los siguientes:

Comprobaciones generales:

- La interferencia no es correcta;

- Cuerpo: La operación no existe;
- Basehexagonal: La operación no existe;
- Tetonprincipal: La operación no existe;
- Tetonsecundario : La operación no existe;
- Saliente: La operación no existe;
- El volumentotal es distinto;
- Distinto número de redondeos;
- Distinto número de agujeros;

Comprobación de operaciones:

- Basehexagonal: No existe;
- Cuerpo: No existe;
- Tetonprincipal: No existe;
- Tetonsecundario: No existe;

Se han retirado de la lista aquellas operaciones que en el modelo de referencia no tienen un nombre específico.

Modelo prueba 2

Este modelo ha obtenido una nota de 5.8 sobre diez. Los fallos que ha extraído el programa son:

General:

- Los ejes del modelo no son correctos
- La interferencia no coincide;
- Basehexagonal: El volumen es distinto;
- Tetonprincipal: El volumen es distinto;
- Tetonsecundario: El volumen es distinto;
- Saliente: El volumen es distinto;
- El volumen de la pieza es distinto;
- Distinto número de redondeos;

Inspección por operaciones:

- Basehexagonal: El volumen no coincide;
- Basehexagonal: No está completamente restringido;
- Cuerpo: Las restricciones no son las mismas;
- Tetonprincipal: El volumen no es el mismo;
- Tetonprincipal: Las restricciones no son las mismas
- TetonsecundarioEl volumen no es el mismo;
- Tetonsecundario: No está completamente restringido;

Se han retirado de la lista aquellas operaciones que en el modelo de referencia no tienen un nombre específico.

6. Conclusiones del TFM

6.1 Resultados obtenidos

Se ha puesto a prueba el programa con distintas piezas y se ha podido alcanzar un resultado satisfactorio, lo que implica que los objetivos planteados para este proyecto se han alcanzado. El programa permite al profesor seleccionar una pieza y unas características, y utilizar esta información como referencia para la corrección de otro modelo.

El software no sólo centra la pieza en el centro de masas y comprueba su volumen total, realiza una serie de comprobaciones más específicas de cada parte y con ello asigna una nota al modelo, más allá de apto y no apto; sino que ofrece un espectro más variado de calificaciones que los programas que existen en la actualidad. Permite cierto grado de personalización de los parámetros para que sirva para la evaluación de una gran variedad de piezas distintas.

Además, almacena los resultados en un Excel con la puntuación de cada parte y los comentarios para cada pieza, por lo que en caso de haber alguna discrepancia el profesor puede consultar los comentarios antes de realizar la revisión manual del modelo.

Otra ventaja que tiene el programa es que el profesor no necesita abrir cada archivo individualmente, sino que corrige todos los archivos dentro de una carpeta, reduciendo en gran medida el tiempo de corrección.

De hecho, el tiempo medio de corrección de una pieza con el programa es de un minuto, mucho más rápida que la corrección manual que se lleva a cabo actualmente.

Por lo tanto este primer acercamiento a la corrección de modelos CAD 3D permite reducir la variabilidad de notas en la corrección de exámenes debida a la diferencia de criterio entre profesores, reducir el tiempo de corrección y el tiempo consumido en completar manualmente el Excel de notas.

Aun así, el programa aún tiene algunas mejoras que podrían llevarse a cabo a modo de versión dos.

6.2 Actualizaciones y posibilidades de ampliación y mejora

Perfeccionamiento del programa

Todavía quedan posibilidades por explorar para la identificación de fallos del modelo. Se podría intentar incluir una segunda referencia para poder contemplar los casos en los que hay varias soluciones válidas.

Se podrían expandir las funciones que sólo se aplican a las operaciones seleccionadas a todas las funciones del modelo.

Respecto al formulario, podría ser interesante añadir una pestaña de ajustes avanzados que permitiera modificar los pesos de todas las partes evaluadas sin necesidad de modificar el código.

Cambio a Visual Basic

Una de las posibles líneas de mejora para el software es pasar a Visual Studio, ya que esto aportaría mayor flexibilidad en el formulario, permitiendo que se personalice para cada modelo. Un ejemplo de esta personalización sería que, en lugar de presentar una serie de selectores, aparecieran todas las operaciones del modelo para escoger seleccionarlas o no. Ilogic no permite

crear formularios dinámicos, por lo que no es posible hacer esta mejora sin cambiar de programa.

Incorporación de excepciones y ajuste de parámetros de evaluación:
Para realizar esta mejora habrá que realizar una serie de pruebas con el programa, en las que se introducirán una gran cantidad de modelos distintos y se estudiará la respuesta del software. Este estudio permitirá detectar carencias y solucionarlas. Además, se podrán ajustar los criterios de evaluación comparando la nota que le asigna el profesor y la que asigna el programa.

6.3 Resultados personales

Este TFM ha supuesto un gran reto de aprendizaje autónomo, ya que ha sido necesario familiarizarse con una herramienta nueva. También me ha permitido desarrollar una de las habilidades imprescindibles en un ingeniero, la capacidad de aplicar unos conocimientos básicos adquiridos durante la carrera como fundamento para la adquisición de nuevos conocimientos, más complejos y concretos.

Tanto en el grado como en el máster, las distintas asignaturas cursadas no sólo nos enseñan los contenidos de esa materia, sino también metodologías de trabajo y aprendizaje que nos servirán en un futuro para asumir nuevos retos profesionales y adaptarnos a los requerimientos de la industria. Específicamente, en este proyecto han sido de gran utilidad los conocimientos de programación en distintos leguajes adquiridos tanto en primero y cuatro de grado (C++ y Android), ya que ha servido de base para aprender VB.net; y por supuesto las asignaturas tanto de grado como de máster en las que se trabaja la confección de modelos 3D, ya que el proyecto está orientado a resolver un problema en este campo de la ingeniería.

El proyecto también me ha permitido adquirir mayor sensibilización en cuanto a la importancia de todos los parámetros y detalles que incurren en la realización y la excelencia de un diseño.

Por último, cabe mencionar la posibilidad de compenetración de habilidades técnicas y creativas en este proyecto, ya que la programación, a pesar de tener unas normas muy estrictas en cuanto al lenguaje y estructuras, tiene infinitas posibilidades. Permite crear gran variedad de soluciones a un mismo problema de formas distintas, lo que convierte solucionar un problema en un reto creativo. Dispones de una serie de herramientas sencillas y un objetivo que alcanzar, y utilizando tu ingenio debes combinar esas herramientas de forma creativa para diseñar tu propia solución al problema.

Bibliografía

[1] Contero, M., Company, P., Aleixos, N. y Vila, C. 2000. Metodología de modelado con herramientas CAD/CAM avanzadas. XII Congreso Internacional de Ingeniería Gráfica. ISBN 84-8448-008-9

[2] Otey, J.: A contribution to conveying quality criteria in mechanical CAD models and assemblies through rubrics and comprehensive design intent qualification. PhD Thesis, Submitted to the Doctoral School of Universitat Politècnica de València (2017)

[3]Company, P., Otey, J., Agost, M.-J., Contero, M., Camba, J.D. 2018. Go/No Go Criteria in Formative E-Rubrics. Lecture Notes in Computer Science, 10925 LNCS, pp. 254-264

[4] Andrews, PTJ, Shahin, TMM, Sivaloganathan, S. Design reuse in a CAD environment – four case studies. Comput. Ind. Eng. 1999; 37(1): 105-9.

Libro: Microsoft visual basic. net. lenguaje y aplicaciones (3ª ed.) de Francisco Javier Ceballos

Apuntes de la asignatura Ingeniería gráfica

https://forums.autodesk.com

https://knowledge.autodesk.com

http://help.autodesk.com

https://appsource.microsoft.com

https://universidadeuropea.es

https://forum.solidworks.com

http://inventortrenches.blogspot.com/2012/01/creating-basic-ilogic-rule-with-event.html

https://social.msdn.microsoft.com/Forums

https://docs.microsoft.com

https://issuu.com/acunar/docs/gbp_completo

https://stackoverflow.com

https://www.cadlinecommunity.co.uk

http://vb.net-informations.com

https://www.2acad.es

http://decsai.ugr.es

https://www.tutorialspoint.com

https://modthemachine.typepad.com

http://forums.codeguru.com

https://github.com

https://www.youtube.com/watch?v=i2lxyG9vXVM

https://www.youtube.com/watch?v=FKDohvqtG9o

https://www.youtube.com/watch?v=K0ZGkdNLyVA

https://www.youtube.com/watch?v=GatSovRjFvE&t=370s

https://www.youtube.com/watch?v=K0ZGkdNLyVA&t=116s

https://www.youtube.com/watch?v=zt-0XxChuco

II.
Presupuesto

Indice del presupuesto

Presupuesto

Partida 1: Mano de obra

Se ha estimado el coste por hora de contratar un ingeniero industrial en 15.625 €/h

Programación del software: Tiempo dedicado a la programación del software incluyendo las pruebas de funcionamiento correcto de las funciones

Realización de pruebas: Tiempo dedicado a la puesta a prueba final del programa con las piezas del modelo y al análisis de los resultados

Redacción de la memoria: Tiempo dedicado a la redacción y revisión de la memoria del proyecto.

Partida 1:Mano de obra					
Código	Concepto	Unidad Básica	Cantidad (h)	Precio por Ud.Básica (l/h)	Precio total(l)
O2	Ingeniero:Programación del sofware	h	194	15,625	3031,25
O3	Ingeniero: Realización de pruebas	h	52	15,625	812,5
O4	Ingeniero: Redacción de la memoria	h	54	15,625	843,75
			300	Total(l):	4687,5
				Costes indirectos	2%
				Subtotal(l)	4781,25

Partida 2: Materiales

Licencia de Autodesk Inventor: El programa utilizado para la realización del software. El precio de un mes de licencia es de 345€ y se ha utilizado durante 4 meses.

Licencia de Microsoft Office: Programa utilizado para la redacción de la memoria y realización de este presupuesto. El coste de la licencia mensual es de 7€.

Coste de Material de oficina: Esto incluye coste de impresión de material de apoyo y otros materiales utilizados.

Ordenador Personal: El coste del ordenador fue de 900€, además de 200€ en reparaciones del equipo para alargar su vida útil. El ordenador lleva 5 años en funcionamiento por lo que el coste mensual sería de 18,33 €

Partida 2:Materiales					
Código	Concepto	Unidad Básica	Cantidad	Precio por Ud.Básica	Precio total(l)
M2	Licencia Autodesk Inventor	l/mes	4 meses	345	1380
M3	Licencia Microsoft Office	l/mes	4 meses	7	28
M4	Coste Material oficina	l	1ud	30	30
M5	Ordenador personal	l	4 meses	18,33	73,32
	MICROSOFT VISUAL BASIC. NET. LENGUAJE Y APLICACIONES (3ª ED.)	l	1ud	33,15	33,15
				Total(l):	1544,47
				Costes indirectos	2%
				Subtotal(l)	1575,3594

Presupuesto final

Partida 1:Mano de obra					
Código	Concepto	Unidad Básica	Cantidad (h)	Precio por Ud.Básica (l/h)	Precio total(l)
O2	Ingeniero:Programación del sofware	h	194	15,625	3031,25
O3	Ingeniero: Realización de pruebas	h	52	15,625	812,5
O4	Ingeniero: Redacción de la memoria	h	54	15,625	843,75
			300	Total(l):	4687,5
				Costes indirectos	2%
				Subtotal(l)	4781,25

Partida 2:Materiales					
Código	Concepto	Unidad Básica	Cantidad	Precio por Ud.Básica	Precio total(l)
M2	Licencia Autodesk Inventor	l/mes	4 meses	345	1380
M3	Licencia Microsoft Office	l/mes	4 meses	7	28
M4	Coste Material oficina	l	1ud	30	30
M5	Ordenador personal	l	4 meses	18,33	73,32
	MICROSOFT VISUAL BASIC. NET. LENGUAJE Y APLICACIONES (3ª ED.)	l	1ud	33,15	33,15
				Total(l):	1544,47
				Costes indirectos	2%
				Subtotal(l)	1575,3594
				Coste total(sinIV	6325,72
				IVA	21%
			Coste total del proyecto(l)		7654,12

El coste final del proyecto es de:

SIETE MIL SEISCIENTOS CINCUENTA Y CUATRO EUROS Y 12 CÉNTIMOS

III.

Anexos

Índice de Anexos

Regla “Evaluate Model”

Main function

```
'Executes the program, call the functions
	'PreviousReviews(...)
	'interf(...)
	'CheckTotalVolume(...)
	'CountFeature(...)
	'CountFeatureByType(...)  'For all the type to evaluate
	'ResEvalFeature(...)
	'ResFeatVolum(...)
	'ResConstr(...)

Sub Main()
	'The list with all the document names from the evaluating
document folder
	FileList()
	Dim firstrun = True
	excelline=3
	'A marker to repeat the code
	Start:
	Dim sfeatu As Parameter 'Variable to tak the name of a feature
to use on the functions
	'Declarevariables
	Dim Asem As AssemblyDocument
	Asem = ThisApplication.ActiveDocument
	Dim doc As Document
	Dim  Resiterf, ResTotalVolume As Double
	'Variables to take results of the functions
	Dim ResCountFeat, ResHole, ResPattern _
	, ResSweep, ResMirror, ResShell As Integer
	Dim oFeatureName() = {"Feature1","Feature2", "Feature3",
"Feature4", "Feature5"}
	Dim ResEvalFeature(5), ResFeatVolum(5), ResConstr(5) As Integer
	'Variables to safe documents
	Dim ref As Document
	Dim eval As Document
	Dim DocumentCounter = 0
	'Search each parte on the asembly document and save on ref or
eval.

	For Each doc In Asem.AllReferencedDocuments
		DocumentCounter = DocumentCounter + 1
		If
doc.FullFileName.Contains(SharedVariable("RefFileName"))
			ref = doc
		Else
			eval = doc

		End If

	Next

'If the function previus reviews find any error stop runing the
program.
	If PreviousReviews(DocumentCounter, firstrun) = 1

		Exit Sub
```

```
        End If

        'A variable to take information and write a final message.The
variable is argument of all the functions and takes the result off
each one
        'Firts take the file name and and "Failure Summary" then all
the results

        LocatePart(eval, ref,sFinalText)
        'First the functions that aplies to all the part
        Resiterf=interf(sFinalText) 'Checks interference between parts
:0 if the result of the interference is 100%, -1 interference <> 100
        VolumeDiferenceForAll(ref, eval,sFinalText)'Compares total
volume 0 to same volume -1 to different
        ResTotalVolume = CheckTotalVolume(sFinalText)'0 if the volume
is equal , -1 is not
        CountFeature(ref, eval, sFinalText)'Count the number of
features used and indicates if is different than on reference - only
informative
        ResCountFeat =FeatureEntitiesCount(eval,sFinalText) 'Retuns the
number of errors
        'Count the number of features by type on all the document, if
is the same returns 0 ,-1 if there are less and 1 if there are more-
only informative
        'ResHole=CountFeatureByType(ref, eval, "Hole")
        'ResCPattern = CountFeatureByType(ref, eval, "Circle")
        'ResRPattern=CountFeatureByType(ref, eval, "Rectangle")
        'ResSweep=CountFeatureByType(ref, eval, "Sweep")
        'ResMirror=CountFeatureByType(ref, eval, "Mirror")
        'ResShell = CountFeatureByType(ref, eval, "Shell")
        'ResFillet = CountFeatureByType(ref, eval, "Fillet")
        'ResChamfer = CountFeatureByType(ref, eval, "Chamfer")
        'ResRib = CountFeatureByType(ref, eval, "Rib")

'Seconly run the funtions to evaluate a especific feature
'Use a for to write on all the positions of the vector to save the
results
'If the user didn't select any feature to that position on the form
the result is 0
iFeatCounter=SharedVariable("FeatCounter")
        iFeatCounter=iFeatCounter-1
        For i As Integer = 0 To iFeatCounter

                If CheckFeatureExist(eval,
SharedVariable("sFeatureNames")(i)) = -1
                        ResEvalFeature(i) = -1
                        ResFeatVolum(i) = -1
                        ResConstr(i) = -1
                        sFinalText=sFinalText &
SharedVariable("sFeatureNames")(i)& ": doesn't exist;"
                Else

                        If
SharedVariable("sFeatureNames")(i).ToString.Equals("None")

                                ResEvalFeature(i)=2
                                ResFeatVolum(i)=2
                                ResConstr(i) = 2
        'If is not empty, call the functions to evaluate
                        Else
```

```
                                ResEvalFeature(i) =
EvaluateFeature(SharedVariable("sFeatureNames")(i), ref,
eval,sFinalText)' 0 to same type 1 to different but valid type -1 to
wrong type

                                ResFeatVolum(i) =
VolumeDiference(SharedVariable("sFeatureNames")(i), ref, eval,
sFinalText) '0 to same volume -1 to different volume

                                ResConstr(i) = SketchConstraints(ref,
eval, SharedVariable("sFeatureNames")(i), sFinalText)'0 fully
constrait and equal to reference,1 fully constrait but different, -1
not fully constrait or only with dimensional

                        End If
                End If

        Next

FinalPunctuation(ResEvalFeature ,ResFeatVolum, ResConstr,Resiterf ,
ResTotalVolume ,ResCountFeat, sFinalText)
        'Shows the resume message
        AddtoExcelfile(excelline,eval,sFinalText)
        'Remove the evaluated document of the path list
        RemoveListElement(eval)
        'Eliminate the part and add a new one
        deletepart(eval)
        InventorVb.DocumentUpdate()
        'If the next document exist go to start and run an other time
the program if not finish program
        If SharedVariable("PathList")(0)<>"End"
        GoTo start
        End If
MsgBox("Evaluation finished")
End Sub
```

SummaryMessage(...)

```
'This function shows a message before starting the evaluation of the
model _
'shows a message indicating the features that are going to be
inspected on detail.
'This function is executed inside the function previusrevisions(...)
Function SummaryMessage(ByRef Firstrun As Boolean) As Integer

If Firstrun=True
        oText = ""
        Dim sFeatureNames As New List(Of String)
        Dim FeatureWeight As New List(Of Double)
        Dim FeatCounter As Integer
        FeatCounter=0
        'Add to a text variable the name of the features that are going
to be checked,
        If Equals(parameter("Feature1"),"None")=False
        oText = oText & vbLf & parameter("Feature1")
        sFeatureNames.Add(parameter("Feature1"))
        FeatureWeight.Add( parameter("Punctuation1"))
        FeatCounter = FeatCounter + 1
        End If
         If Equals(parameter("Feature2"),"None")=False
        oText = oText & vbLf & parameter("Feature2")
        sFeatureNames.Add(parameter("Feature2"))
        FeatureWeight.Add( parameter("Punctuation2"))
        FeatCounter = FeatCounter + 1
        End If
         If Equals(parameter("Feature3"),"None")=False
        oText = oText & vbLf & parameter("Feature3")
        sFeatureNames.Add(parameter("Feature3"))
        FeatureWeight.Add( parameter("Punctuation3"))
        FeatCounter=FeatCounter+1
        End If
        If Equals(parameter("Feature4"),"None")=False
        oText = oText & vbLf & parameter("Feature4")
        sFeatureNames.Add(parameter("Feature4"))
        FeatureWeight.Add( parameter("Punctuation4"))
        FeatCounter = FeatCounter + 1
        End If
        If Equals(parameter("Feature5"),"None")=False
        oText = oText & vbLf & parameter("Feature5")
        sFeatureNames.Add(parameter("Feature5"))
        FeatureWeight.Add( parameter("Punctuation5"))
        FeatCounter = FeatCounter + 1
        End If
        If Equals(parameter("Feature6"),"None")=False
        oText = oText & vbLf & parameter("Feature6")
        sFeatureNames.Add(parameter("Feature6"))
        FeatureWeight.Add( parameter("Punctuation6"))
        FeatCounter = FeatCounter + 1
        End If
        If Equals(parameter("Feature7"),"None")=False
        oText = oText & vbLf & parameter("Feature7")
        sFeatureNames.Add(parameter("Feature7"))
        FeatureWeight.Add( parameter("Punctuation7"))
        FeatCounter = FeatCounter + 1
        End If
        If Equals(parameter("Feature8"),"None")=False
```

```
        oText = oText & vbLf & parameter("Feature8")
        sFeatureNames.Add(parameter("Feature8"))
        FeatureWeight.Add( parameter("Punctuation8"))
        FeatCounter = FeatCounter + 1
        End If
        If Equals(parameter("Feature9"),"None")=False
        oText = oText & vbLf & parameter("Feature9")
        sFeatureNames.Add(parameter("Feature9"))
        FeatureWeight.Add( parameter("Punctuation9"))
        FeatCounter = FeatCounter + 1
        End If
        If Equals(parameter("Feature10"),"None")=False
        oText = oText & vbLf & parameter("Feature10")
        sFeatureNames.Add(parameter("Feature10"))
        FeatureWeight.Add( parameter("Punctuation10"))
        FeatCounter = FeatCounter + 1
        End If
        SharedVariable("sFeatureNames")=sFeatureNames
        SharedVariable("FeatCounter") = FeatCounter
        SharedVariable("FeatureWeight") =FeatureWeight
        'Show on a msgbox the text before executing the program
        i=MessageBox.Show("Features to review" & vbCrLf & oText , _
        "Summary", MessageBoxButtons.YesNo, MessageBoxIcon.Asterisk,
MessageBoxDefaultButton.Button1)
        Firstrun = False
        CreateNewExcel()
        End If
 If i = 7
         Return 1
 End If
 Return 0
End Function
```

PreviousReviews()

```
'This function inspects the values that the user has introduced on the
form and shows the summary message_
'using the function SummaryMessage().
'Checks that the values are correct.
Function PreviousReviews(FileCounter As Integer,ByRef firstrun As
Boolean) As Integer
CenterPieces()
Dim i = 0
i = SummaryMessage(firstrun)
'checks the value of each Parameter to ensure that if its value Is
none its score Is 0
FeaturesCounter=SharedVariable("FeatCounter")
'checks that the sum of the punctuations is 10 if it's not shows an
error message.


If FeaturesCounter = 10
        i = MessageBox.Show("There isn't features to evaluate ",
"Error", _
                MessageBoxButtons.OK, MessageBoxIcon.Hand,
MessageBoxDefaultButton.Button3)

End If

'Error message for problem with files
If FileCounter <2
        i = MessageBox.Show("A file is missing ", "Error", _
        MessageBoxButtons.OK, MessageBoxIcon.Hand,
MessageBoxDefaultButton.Button3)

        Else If FileCounter >2
        i = MessageBox.Show("There are too many files ", "Error", _
        MessageBoxButtons.OK, MessageBoxIcon.Hand,
MessageBoxDefaultButton.Button3)
End If
If i = 1
        Return i

Else
        Return 0
End If

End Function
```

CheckFeatureExist(...)

```
'Function created to verify if the feature to evaluate exist on the
model. Helps to aboit errors.
'Checks by the name is the feature is created on the model.
'Arguments:
        'eval: Evaluating document
        'sfeatu : The name as string of the feature to evaluate, we get
this name from the userparameters that the program creates, then the
user selects
'Returns 0 is the parameter exist -1 if not.
Function CheckFeatureExist(eval As Document, sfeatu As String) As
Integer
        Dim oFeature As PartFeature
        Dim oFeatures As PartFeatures
        oFeatures = eval.ComponentDefinition.Features
        For Each oFeature In oFeatures

                If sfeatu="None" Or sfeatu=oFeature.Name
                        Return 0
                End If
        Next
        Return -1
End Function
```

Centerpieces()

```
'Is used to make sure that the gravity center of the 2 pieces is on
the same point
'This function is used on PreviousReviews(...) to ensure the parts are
ready for checking
Function CenterPieces()
'Declares the variables needed to access to the assembly elements and
his names on the model browser.
'Also declares the variable that are needed to move the part.
Dim asemCompDef As AssemblyComponentDefinition
asemCompDef = ThisApplication.ActiveDocument.ComponentDefinition
Dim oOccurrence As ComponentOccurrence
Dim temp As TransientGeometry = ThisApplication.TransientGeometry
'Now gets the center of mass of all the parts on the assembly
For Each oOccurrence In asemCompDef.Occurrences
Dim centromasa As Point =
oOccurrence.Definition.MassProperties.CenterOfMass
'Gets the values of the Center of mass
centromasa.TransformBy(oOccurrence.Transformation)
'Creates the matrix needed to make the traslation to point 0,0,0
Dim oMatrix As Matrix = oOccurrence.Transformation
'Makes the movement to the 0,0,0
Dim oTranslation As Vector = oMatrix.Translation.Copy
oTranslation.AddVector(centromasa.VectorTo(temp.CreatePoint(0, 0, 0)))
oMatrix.SetTranslation(oTranslation, False)
oOccurrence.Transformation = oMatrix
Next
End Function
```

Interf(...)

```
'Arguments needed:
	'sFinalText : a String used by all the functions to get
information about the results of the checking
'At the end of the program it shows that information to the user
'Takes the two parts and checks the interference, if the interference
volume equals the referece
'volume the checked part will be geometrically equal
	'Returns a integer .
	'	Returns 0 if the interference volume equals the
reference piece.
	'	Returns -1 if the interference volume is different from
the volume of the reference piece.
Function interf(ByRef sFinalText As String) As Double
'Creates variables needed to access all data
Dim oAsmDoc As AssemblyDocument
oAsmDoc = ThisApplication.ActiveDocument
Dim oAsmCompDef As AssemblyComponentDefinition
oAsmCompDef = oAsmDoc.ComponentDefinition
' Adds each occurrence in the assembly to the object collection.
Dim oCheckSet As ObjectCollection
oCheckSet= ThisApplication.TransientObjects.CreateObjectCollection
Dim oOccurrence As ComponentOccurrence
'adds all the parts to the object list
'if any of them is the reference, takes its volume
For Each oOccurrence In oAsmCompDef.Occurrences
	If oOccurrence.Name.Contains("Referencia")
		volumenref =
iProperties.VolumeOfComponent(oOccurrence.Name)
		volumenref = Round(volumenref / 1000,5)

	End If
	oCheckSet.Add (oOccurrence)
Next
'takes interference volume of the two objects on the oCheckSet
Dim oResults As InterferenceResults
oResults = oAsmCompDef.AnalyzeInterference(oCheckSet)
For Each oResult In oResults
	volumeninterferencia=Round(oResult.Volume,5)
Next
porcentaje=Round(100*volumeninterferencia/volumenref,3)
'Compares the interference volume with the reference volume, if they
are equal returns 0
'If they aren't equal adds a message to sFinaltext and returns -1
If volumeninterferencia = volumenref
	'MsgBox("las piezas son iguales")
	Return porcentaje
Else
	sFinalText=sFinalText & "The part interference is not the
same;"

End If

Return porcentaje
End Function
```

CheckTotalVolume(...)

```
'Arguments:
	'sFinalText : a String used by all the functions to get
information about the results of the checking
	'At the end of the program it shows that information to the
user
'Takes the volume of the two parts and checks if they are equal.
'Returns an integer .
'Returns 0 if the volumen of the two parts is equal
'Returns -1 if the volume is not equal

Function CheckTotalVolume(ByRef sFinalText As String) As
Double'comprueba el volumen total
	Dim volumenref,volumeneval As Double
	Dim oAsmCompDef As AssemblyComponentDefinition
	oAsmCompDef =
ThisApplication.ActiveDocument.ComponentDefinition
	Dim oOccurrence As ComponentOccurrence
	For Each oOccurrence In oAsmCompDef.Occurrences
		If oOccurrence.Name.Contains("eferencia")

	volumenref=Round(iProperties.VolumeOfComponent(oOccurrence.Name
),7)
		Else
			volumeneval =
Round(iProperties.VolumeOfComponent(oOccurrence.Name), 7)

		End If

	Next
	porcentaje = 100*volumeneval / volumenref
	If porcentaje>100
		porcentaje = 100 - (porcentaje - 100)
	End If
	If porcentaje<0 Or porcentaje>100
		porcentaje = 0
	End If
	If volumeneval = volumenref
		Return porcentaje
	Else If volumeneval <> volumenref
		sFinalText=sFinalText &" The volume of the part is
different from the reference volume;"
		Return porcentaje
	End If

End Function
```

CountFeature(...)

```
'Counts the total number of features that are needed to complete the
design on the reference and the model to evaluate
'and compares one against the other to check if they are the same.

	'sFinalText : a String used by all the functions to get
information about the results of the checking
	'At the end of the program it shows that information to the
user
	'ref: Reference document
	'eval: Evaluating document
'Returns: 0 if the number of features on both the model and the
reference are the same
'Returns: 1 if the model has less features
'Returns: -1 if the model has more features

Function CountFeature(ref As Document,eval As Document,ByRef
sFinalText As String)
	Dim oFeature As PartFeature
	Dim oFeatures As PartFeatures
	oFeatures = ref.ComponentDefinition.Features
	RefFeat = 0
	For Each oFeature In oFeatures
		RefFeat =RefFeat+1
	Next

	oFeatures = eval.ComponentDefinition.Features
	EvalFeat = 0
	For Each oFeature In oFeatures
		EvalFeat =EvalFeat+1
	Next

	If EvalFeat < RefFeat
		sFinalText=sFinalText &"Less total feature number than
on reference;"

	Else If EvalFeat > RefFeat
		sFinalText=sFinalText &" More total feature number than
on reference;"


	End If
End Function
```

FeatureEntitiesCount(...)

```
'Count the entities of the "other features" and indicates the number
of errors
'Arguments()
	'sFinalText : a String used by all the functions to get
information about the results of the checking
	'At the end of the program it shows that information to the
user
	'eval: Evaluating document
'Return: The number of errors that has found , FinalPunctuation will
get this information
Function FeatureEntitiesCount(eval As Document, ByRef sFinalText As
String)As Integer
	Dim oFeature As PartFeature
	oFeatures = eval.ComponentDefinition.Features
	Dim oChamferCount = 0
	Dim oFilletCount = 0
	Dim oHoleCount = 0
	Dim oRibCount = 0
	Dim oSweepCount = 0
	Dim oCPatternCount = 0
	Dim oRPatternCount = 0
	Dim oMirrorCount = 0
	Dim oShellCount = 0
	Dim oTotalCount=0
	EvalFeat=0
	For Each oFeature In oFeatures
		EvalFeat=EvalFeat+1
		Call oFeature.SetEndOfPart(True)
		If TypeOf (oFeature) Is RibFeature
			oRibCount = oRibCount + 1

		End If
		If TypeOf (oFeature) Is SweepFeature
			oSweepCount = oSweepCount + 1

		End If
		If TypeOf (oFeature) Is CircularPatternFeature

			oCPatternCount = oCPatternCount + 1


		End If
		If TypeOf (oFeature) Is RectangularPatternFeature
			oRPatternCount = oRPatternCount + 1

		End If
		If TypeOf (oFeature) Is MirrorFeature
			oMirrorCount = oMirrorCount + 1

		End If
		If TypeOf (oFeature) Is ShellFeature
			 oShellCount = oShellCount + 1

		End If

		If TypeOf (oFeature) Is ChamferFeature
```

```
                        oChamferCount =oChamferCount+
oFeature.Definition.ChamferedEdges.Count

                End If
                If TypeOf (oFeature) Is FilletFeature
                        Call oFeature.SetEndOfPart(False)

                        oFilletCount =oFilletCount+
oFeature.Faces.Count

                End If
                If TypeOf (oFeature) Is HoleFeature
                                oHolesCount =oHolesCount+
oFeature.HoleCenterPoints.Count
                End If
                Call oFeature.SetEndOfPart(False)
        Next
        position = 0
        'Verifies the position of the feature and if is a chamfer or a
fillet adds a error message if they arent at the end of the model
        For Each oFeature In oFeatures
                position=position+1
                If TypeOf (oFeature) Is ChamferFeature Or TypeOf
(oFeature) Is FilletFeature
                        If position<EvalFeat / 2
                                sFinalText = sFinalText & oFeature.Name
& ": Bad feature priority"
                        End If
                End If
        Next
        oTotalCount= oChamferCount+oFilletCount+oHolesCount+oRibCount
+oSweepCount+ oCPatternCount+oRPatternCount+oMirrorCount+oShellCount
        BadCount=0
        If Parameter("CountChamfer") <> oChamferCount
                sFinalText = sFinalText & "Different number of
chamfers;"
                BadCount = BadCount + Abs(Parameter("CountChamfer") -
oChamferCount)
        End If

        If Parameter("CountFillet") <> oFilletCount
                sFinalText = sFinalText & "Different number of
fillets;"
                BadCount=BadCount+Abs(Parameter("CountFillet") -
oFilletCount)
        End If

        If Parameter("CountHole")<> oHolesCount
                sFinalText = sFinalText & "Different number of holes;"
                BadCount=BadCount+Abs(Parameter("CountHole")-
oHolesCount)
        End If
        If Parameter("CountRib") <> oRibCount
                sFinalText = sFinalText & "Different number of ribs;"
                BadCount=BadCount+Abs(Parameter("CountRib") -
oRibCount)
        End If
        If Parameter("CountSweep") <> oSweepCount
                sFinalText = sFinalText & "Different number of sweeps;"
                BadCount=BadCount+Abs(Parameter("CountSweep") -
oSweepCount )
```

```
        End If
        If Parameter("CountCpattern")<> oCPatternCount
                sFinalText = sFinalText & "Different number of circular
patterns;"
                BadCount=BadCount+Abs(Parameter("CountCpattern")-
oCPatternCount)
        End If
        If Parameter("CountRpattern") <> oRPatternCount
                sFinalText = sFinalText & "Different number of
rectangular patterns;"
                BadCount=BadCount+Abs( Parameter("CountRpattern") -
oRPatternCount)
        End If
        If Parameter("CountMirror")<> oMirrorCount
                sFinalText = sFinalText & "Different number of mirror;"
                BadCount=BadCount+Abs( Parameter("CountMirror")-
oMirrorCount)

        End If
        If Parameter("CountShell")<> oShellCount
                sFinalText = sFinalText &"Different number of shells;"
        End If
        Return BadCount
End Function
```

CountFeatureByType

```
'Arguments:
	'sFinalText : a String used by all the functions to get
information about the results of the checking
	'At the end of the program it shows that information to the
user
	'ref: Reference document
	'eval: Evaluating document
	'Type: The name as string of the feature we want to count the
elements from
'With the type indicated on the arguments, checks how many features of
this type there are in the two models and compares them
'Returns 0 if both the model and the reference have the same number of
features of that type
'Returns 1 if there are more features on the model
'Returns -1 if there are less features on the model
Function CountFeatureByType(ref As Document,eval As Document,Type As
String ) As Integer 'Cuenta y verifica que el numero de operaciones es
el mismo
	Dim oFeature As PartFeature
	Dim oFeatures As PartFeatures
	oFeatures = ref.ComponentDefinition.Features
	RefFeat = 0
	For Each oFeature In oFeatures
		If oFeature.Type.ToString.Contains(Type)
		RefFeat = RefFeat + 1
		End If
	Next
	oFeatures = eval.ComponentDefinition.Features
	EvalFeat = 0
	For Each oFeature In oFeatures
		If oFeature.Type.ToString.Contains(Type)
		EvalFeat = EvalFeat + 1
		End If
	Next
	If EvalFeat > RefFeat
		sFinalText=sFinalText & "There is more" & Type & "on
the model than on the reference;"
		Return 1
	Else If EvalFeat < RefFeat'hay mas operaciones de ese Type
		sFinalText=sFinalText & "There is less"& Type & "on the
model than on the reference;"
		Return -1
	Else
		Return 0
	End If
End Function
```

TakeFeaturetype()

```
'Returns the feature type from a feature with a provided name and the
document to be looked in.
'Arguments:
	'sfeatu : The name as string of the feature to evaluate, we get
this name from the userparameters that the program creates, then the
user selects
	'Doc : The document where the program has to search the
parameter
'Returns the feature type as string.
'Called by :
	'EvaluateFeature(...)

Function TakeFeaturetype(sfeatu As String, Doc As Document) As String
'devuelve el tipo de operacion de la operación en el documento
	Dim oFeature As PartFeature
	Dim oFeatures As PartFeatures
	oFeatures = Doc.ComponentDefinition.Features
	For Each oFeature In oFeatures

		If oFeature.Name = sfeatu.ToString
			Return oFeature.Type.ToString
		End If
	Next
	Return ""
End Function
```

EvaluateFeature(...)

```
'Using the function  TakeFeaturetype compares the type of the selected
feature in the reference and the model
'Arguments
        'sfeatu : The name as string of the feature to evaluate, we get
this name from the userparameters that the program creates, then the
user selects
        'sFinalText : a String used by all the functions to get
information about the results of the checking
        'At the end of the program it shows that information to the
user
        'ref: Reference document
        'eval: Evaluating document
'Returns 0 if the feature type is the same on both documents
'Returns 1 if the feature type is diferent but correct
'Returns -1 if the feature doesn't exist,or if it's different and is
one of the incorrect cases


Function EvaluateFeature(sfeatu As String,ref As Document, eval As
Document,ByRef sFinalText As String) As Integer'comprueba si la
operacion oper es la misma en el documento eval y ref
        'Declares variables
        Dim sFeatureName As String
        sFeatureName = sfeatu
        Dim sEvaltype As String
        Dim sReftype As String
'Takes the feature value , it's sure that have a value becouse it has
been checked by CheckFeatureExist(...)

                sEvaltype = TakeFeaturetype(sFeatureName, eval)
                sReftype = TakeFeaturetype(sFeatureName, ref)
                'starts comparing the types
                'If the types are equal
                If sEvaltype = sReftype

                        Return 0
                'Some feature changes are 100% wrong, returns -1
                Else If sReftype.Contains("Hole") And
sEvaltype.Contains("Extru")
                        sFinalText=sFinalText & sFeatureName & ":  The
feature type is wrong;"
                        Return -1
                Else If sReftype.Contains("Chamfer") And
sEvaltype.Contains("Extru")
                        sFinalText=sFinalText & sFeatureName & ":  The
feature type is wrong;"
                        Return -1
                Else If sReftype.Contains("Rib") And
sEvaltype.Contains("Extru")
                        sFinalText=sFinalText & sFeatureName & ":  The
feature type is wrong;"
                        Return -1
                Else If sReftype.Contains("Stiffener") And
sEvaltype.Contains("Extru")
                        sFinalText=sFinalText & sFeatureName & ":  The
feature type is wrong;"
                Return -1
```

```
                'For the rest of options returns 1; for example, it is
correct to do a revolution or a extrusion to make a cylinder
                Else
                        sFinalText=sFinalText & sFeatureName &":  The
used feature is diferent from the reference;"
                        Return 1
                End If


End Function
```

SketchConstraints ()

```
' Checks if the sketch consumed by the feature is fully constricted
and compares both to check if there are the same number of geometric
and dimensional constraints
'Arguments:
	'sfeatu : The name as string of the feature to evaluate, we get
this name from the userparameters that the program creates, then the
user selects
	'sFinalText : a String used by all the functions to get
information about the results of the checking
	'At the end of the program it shows that information to the
user
	'ref: Reference document
	'eval: Evaluating document
'Returns 0 if it is fully constricted and with the same number of
constraints of each type
'Returns 1 if the sketch if fully constricted but with different
number of constraints
'Returns -1 if is not fullyconstricted
'return 3 for the features that dont use sketch
Function SketchConstraints(ref As Document, eval As Document, sfeatu
As String, ByRef sFinalText As String) As Integer
Dim oFeature As PartFeature
Dim oFeatures As PartFeatures
Dim restringido As String
oFeatures = ref.ComponentDefinition.Features
For Each oFeature In oFeatures
	If oFeature.Name.Equals(sfeatu)
	Dim Sketchref As PlanarSketch
	If TypeOf(oFeature) Is HoleFeature
			Sketchref = oFeature.Sketch
	Else If TypeOf(oFeature) Is RibFeature
			Return 0
	Else If TypeOf(oFeature) Is FilletFeature Or
oFeature.Type.ToString.Contains("Chamfer") Or
oFeature.Type.ToString.Contains("Pattern") Or
oFeature.Type.ToString.Contains("Mirror") Or
oFeature.Type.ToString.Contains("Sweep") Or
oFeature.Type.ToString.Contains("Shell")
		Return 0
	Else
		Sketchref = oFeature.profile.parent
	End If
Try
		Rgref = Sketchref.GeometricConstraints.Count
		Rdref = Sketchref.DimensionConstraints.Count
		Catch
End Try
	End If
Next
oFeatures = eval.ComponentDefinition.Features
For Each oFeature In oFeatures
	If oFeature.Name.Equals(sfeatu)
	Dim Sketcheval As PlanarSketch
	If oFeature.Type.ToString.Contains("Hole")
		Sketcheval = oFeature.Sketch
	Else If oFeature.Type.ToString.Contains("Rib")
		Sketcheval =
oFeature.Definition.ProfileCurves(1).Parent
```

```
        Else If oFeature.Type.ToString.Contains("Fillet") Or
oFeature.Type.ToString.Contains("Chamfer")
                Return 3
        Else
                Sketcheval = oFeature.profile.parent
        End If
        'Uses a try because there are some features that don't need
sketch
        Try
                Rgeval = Sketcheval.GeometricConstraints.Count
                Rdeval = Sketcheval.DimensionConstraints.Count
                restringido = Sketcheval.ConstraintStatus.ToString
                Catch
        End Try
        End If
Next
Try
        If restringido.Contains("Full")
                                If Rgref = Rgeval And Rdref = Rdeval
                                        Return 0
                                Else If Rgeval = 0
                                        sFinalText=sFinalText & sfeatu &
": Only dimensionalconstrainst"
                                Else
                                        sFinalText=sFinalText & sfeatu &
": Constraints are not the same; "
                                        Return 1
                                End If
        Else
                sFinalText=sFinalText & sfeatu & ": isn't
fullyConstraint; "
                Return -1
        End If
        Catch
                Return 3
End Try

Return 0

End Function
```

VolumeDiference()

```
'Checks if the volumen added by a feature, indicated by the user, is
the same on the both documents , reference and model
'Needs the functions ActiveAllFeatures (...) and
DeactivateFeatures(...)
'Firts deactives all the features added after the selected one and
takes the volume of both documents
'Then actives the selected feature an takes a second volume of each
document
'Finally calcules the diference between firts and second volume and
compares for both documents
'Arguments:
	'sfeatu : The name as string of the feature to evaluate, we get
this name from the userparameters that the program creates, then the
user selects
	'sFinalText : a String used by all the functions to get
information about the results of the checking
	'At the end of the program it shows that information to the
user
	'ref: Reference document
	'eval: Evaluating document
'Returs 0 if the difference of volumes on both document is equal
'Returns -1 if the difference of volumes on both document is
different.

Function VolumeDiference(sfeatu As String, ref As Document, eval As
Document, ByRef sFinalText As String) As Integer

	Dim oAsmCompDef As AssemblyComponentDefinition
	oAsmCompDef =
ThisApplication.ActiveDocument.ComponentDefinition
	Dim oOccurrence As ComponentOccurrence
	InventorVb.DocumentUpdate()
	DeactivateFeatures("Referencia",sfeatu,False,ref)
	DeactivateFeatures(OccurenceName, sfeatu, False, eval)
	InventorVb.DocumentUpdate()
	For Each oOccurrence In oAsmCompDef.Occurrences
		If oOccurrence.Name.Contains("eferencia")

	volumen1ref=Round(iProperties.VolumeOfComponent(oOccurrence.Nam
e),7)
		Else

	volumen1eval=Round(iProperties.VolumeOfComponent(oOccurrence.Na
me),7)
		End If
	Next
	DeactivateFeatures("Referencia",sfeatu,True,ref)
	DeactivateFeatures(OccurenceName, sfeatu, True, eval)
	InventorVb.DocumentUpdate()
	For Each oOccurrence In oAsmCompDef.Occurrences
		If oOccurrence.Name.Contains("eferencia")

	volumen2ref=Round(iProperties.VolumeOfComponent(oOccurrence.Nam
e),7)
		Else

	volumen2eval=Round(iProperties.VolumeOfComponent(oOccurrence.Na
me),7)
		End If
```

```
        Next


        difref = volumen2ref - volumen1ref
        difeval = volumen2eval - volumen1eval

        If difref = difeval

                ActiveAllFeatures("Referencia",ref)
                ActiveAllFeatures(OccurenceName,eval)
                InventorVb.DocumentUpdate()


                Return 0

        Else
                sFinalText=sFinalText & sfeatu & "The volume is not the
same;"
                ActiveAllFeatures("Referencia",ref)
                ActiveAllFeatures(OccurenceName,eval)
                InventorVb.DocumentUpdate()
                Return -1
        End If
        ActiveAllFeatures("Referencia",ref)
        ActiveAllFeatures(OccurenceName,eval)
End Function
'Auxiliares
```

OccurenceName ()

```
'Give the occurence name from a document diferent from reference
'Returns a string with the occurrence name.
Function OccurenceName () As String
        Dim Asem As AssemblyDocument
        Asem = ThisApplication.ActiveDocument
        Dim oAsmCompDef As AssemblyComponentDefinition
        oAsmCompDef =
ThisApplication.ActiveDocument.ComponentDefinition
        Dim oOccurrence As ComponentOccurrence
        For Each oOccurrence In oAsmCompDef.Occurrences
                If oOccurrence.Name.Contains("eferencia")

                Else
                        Return oOccurrence.Name
                End If
        Next
End Function
```

ActiveAllFeatures()

```
'Active all the features than has been disabled
'Needs the function OccurenceName()
'Arguments
        'sName : name from the occurence
        'Doc : Document to activate
Function ActiveAllFeatures(sName As String ,Doc As Document)
        Dim oFeature As PartFeature
        Dim oFeatures As PartFeatures
        Dim Asem As AssemblyDocument
        Asem = ThisApplication.ActiveDocument
        oFeatures = Doc.ComponentDefinition.Features
                For Each oFeature In oFeatures
                        Feature.IsActive(sName, oFeature.Name) = True
                Next
End Function
```

DeactivateFeatures()

```
'Disable the features added after the selected one
'Needs the function OccurenceName()
'Arguments
        'bPosition: False to disable all features after
sfeatu,including sfeatu /True to disable all features after sfeatu,not
including sfeatu
        'sName : name from the occurence
        'Doc : Document to activate
        'sfeatu : The name as string of the feature to evaluate, we get
this name from the userparameters that the program creates, then the
user selects
Function DeactivateFeatures(sName As String, sfeatu As String,
bPosition As Boolean,Doc As Document)
        Dim oFeature As PartFeature
        Dim oFeatures As PartFeatures
        Dim Asem As AssemblyDocument
        Asem = ThisApplication.ActiveDocument
        oFeatures = Doc.ComponentDefinition.Features
                i = 0
                For Each oFeature In oFeatures
                        Feature.IsActive(sName, oFeature.Name) = False
                        If oFeature.Name.Equals(sfeatu)
                                Feature.IsActive(sName, oFeature.Name) =
bPosition
                                 i=1
                        Else If oFeature.Name<>sfeatu And i=0
                                Feature.IsActive(sName, oFeature.Name)
=True
                        Else If i=1
                                Feature.IsActive(sName, oFeature.Name)
=False
                        End If

                Next


End Function
```

VolumeDiferenceForAll()

```
'Checks if the volumen added by a feature is the same on the both
documents,for all the features on reference and model
'Needs the functions ActiveAllFeatures (...) and
DeactivateFeatures(...)
'Firts deactives all the features added after the selected one and
takes the volume of both documents
'Then actives the selected feature an takes a second volume of each
document
'Finally calcules the diference between firts and second volume and
compares for both documents
'Arguments:

        'sFinalText : a String used by all the functions to get
information about the results of the checking
        'At the end of the program it shows that information to the
user
        'ref: Reference document
        'eval: Evaluating document
'Returns the number of wrong features that found
Function VolumeDiferenceForAll(ref As Document, eval As Document,ByRef
sFinalText As String)As Integer
        Dim oAsmCompDef As AssemblyComponentDefinition
        oAsmCompDef =
ThisApplication.ActiveDocument.ComponentDefinition
        Dim oOccurrence As ComponentOccurrence
        InventorVb.DocumentUpdate()
        Dim oFeature As PartFeature
        Dim oFeatures As PartFeatures
        oFeatures = ref.ComponentDefinition.Features
        CountWrong=0
        For Each oFeature In oFeatures
                sfeatu = oFeature.Name
                If 0=CheckFeatureExist(eval, sfeatu)
                        DeactivateFeatures("Referencia", sfeatu, False,
ref)
                        DeactivateFeatures(OccurenceName, sfeatu,
False, eval)
                        InventorVb.DocumentUpdate()
                        For Each oOccurrence In oAsmCompDef.Occurrences
                                If
oOccurrence.Name.Contains("eferencia")

        volumen1ref=Round(iProperties.VolumeOfComponent(oOccurrence.Nam
e),7)
                                Else

        volumen1eval=Round(iProperties.VolumeOfComponent(oOccurrence.Na
me),7)
                                End If

                        Next


        DeactivateFeatures("Referencia",sfeatu,True,ref)
                        DeactivateFeatures(OccurenceName, sfeatu, True,
eval)
                        InventorVb.DocumentUpdate()
                        For Each oOccurrence In oAsmCompDef.Occurrences
```

```
                                If
oOccurrence.Name.Contains("eferencia")

        volumen2ref=Round(iProperties.VolumeOfComponent(oOccurrence.Nam
e),7)
                                Else

        volumen2eval=Round(iProperties.VolumeOfComponent(oOccurrence.Na
me),7)
                                End If

                        Next


                        difref = volumen2ref - volumen1ref
                        difeval = volumen2eval - volumen1eval
                        If difref = difeval
                                'MsgBox("el volumen de la operacion 1 es
el mismo")
                                ActiveAllFeatures("Referencia",ref)
                                ActiveAllFeatures(OccurenceName,eval)
                                InventorVb.DocumentUpdate()

                        Else
                                CountWrong=CountWrong+1
                                sFinalText=sFinalText & sfeatu & ": The
volume is diferent;"
                                ActiveAllFeatures("Referencia",ref)
                                ActiveAllFeatures(OccurenceName,eval)
                                InventorVb.DocumentUpdate()

                        End If
                Else
                        sFinalText = sFinalText & sfeatu &  ": The
feature called dosen't exist on the evaluated model;"
                End If
        Next
        Return CountWrong
End Function
```

Rotate()

```
'With a given axis the function rotate the model 90 degrees on this
axis.
Function Rotate(axis As String)
Dim oAsmCompDef As AssemblyComponentDefinition
oAsmCompDef = ThisApplication.ActiveDocument.ComponentDefinition
Dim oOccurrence As ComponentOccurrence
For Each oOccurrence In oAsmCompDef.Occurrences
If oOccurrence.Name.Equals("Referencia")
Else
                Dim oADoc As AssemblyDocument = ThisAssembly.Document
                Dim oADef As AssemblyComponentDefinition =
oADoc.ComponentDefinition
                Dim oTG As TransientGeometry =
ThisApplication.TransientGeometry
                Dim oOcc As ComponentOccurrence =
oADef.Occurrences.ItemByName(oOccurrence.Name)
                Dim oXAxis As WorkAxis = oADef.WorkAxes.Item(axis)
                Dim oLinex As Line = oXAxis.Line


                Dim oOrigMatrix As Matrix = oOcc.Transformation
                Dim oNewMatrix As Matrix = oTG.CreateMatrix
                oNewMatrix.SetToRotation((Math.PI / 2),
oLinex.Direction.AsVector, oLinex.RootPoint)
                oOrigMatrix.PreMultiplyBy(oNewMatrix)
                oOcc.Transformation = oOrigMatrix
End If
Next


End Function
```

LocatePart(...)

```
'Arguments
        'ref: Reference document
        'eval: Evaluating document
'This function is for rotating the axis of the model to evaluate if
they are wrong located.
'Search the Max And Min inertia moment of the reference and the
evaluated model
'and make them coincide using rotate function.
Function LocatePart(eval As Document, ref As Document,byref sFinalText
As String)
'First of all get all the necesary values and create some variables
        Dim xok, yok, zok As Boolean
        xok = False
        yok = False
        zok = False
        'Inertia of ansem get it with momentAsem(),this must be
actualizated in each iteration
        Dim momentAsem() As Double
        momentAsem = inertiaAsem()
        IxxAsem = momentAsem(0)
        IyyAsem = momentAsem(1)
        IzzAsem = momentAsem(2)
        'Then evaluated document
        massPropseval = eval.ComponentDefinition.MassProperties
        Dim momentseval(5) As Double
        massPropseval.XYZMomentsOfInertia(momentseval(0),
momentseval(1), momentseval(2), momentseval(3), momentseval(4),
momentseval(5))
        For i = 0 To 5
         momentseval(i) =
eval.UnitsOfMeasure.ConvertUnits(momentseval(i), "kg cm^2", "kg mm^2")
        Next
        'Is not necesary to make this varible change , is only to help
to understand the code
        Ixxeval = momentseval(0)
        Iyyeval = momentseval(1)
        Izzeval = momentseval(2)
        Ixxeval = Round(Ixxeval,3)
        Iyyeval = Round(Iyyeval,3)
        Izzeval = Round(Izzeval, 3)

 'Finally Get inertia from reference document
        massPropsref = ref.ComponentDefinition.MassProperties
        Dim momentsref(5) As Double
        massPropsref.XYZMomentsOfInertia(momentsref(0), momentsref(1),
momentsref(2), momentsref(3), momentsref(4), momentsref(5))
        For i = 0 To 5
         momentsref(i) = ref.UnitsOfMeasure.ConvertUnits(momentsref(i),
"kg cm^2", "kg mm^2")
        Next
'Is not necesary to make this varible change , is only to help to
understand the code
        Ixxref = momentsref(0)
        Iyyref = momentsref(1)
        Izzref = momentsref(2)
        Ixxref = Round(Ixxref,3)
        Iyyref = Round(Iyyref,3)
        Izzref = Round(Izzref, 3)
        dif1=Round(Ixxeval + Ixxref -IxxAsem,3)
```

```
        dif2=Round(Iyyeval + Iyyref - IyyAsem,3)
        dif3=Round(Izzeval + Izzref - IzzAsem,3)
        If dif1<0.05 And dif2<0.05 And dif3<0.05
        Else
                sFinalText=sFinalText & "Wrong Axis Orientation; "
        End If

'Start iterations to make coincide the max and min inertia of each
part
        'Search the max on the reference and check wich is the max on
the eval model. Is is on the same axis is OK
        'If is different rotate the part .
        'Repeat this until the asembly inertia on the corresponding
axis is the sum of the max inertia axis of both models.
'Repeat the same process with minimum inertia.
While (xok = False And yok = False) Or (xok = False And zok = False)
Or (yok = False And zok = False)
'Search the max inertia axis of each part and make them coincide
'Star searching max on the ref
        If Ixxref = MaxOfMany(Ixxref, Iyyref, Izzref)
        'If the max is on X axis search the max on eval
                If Ixxeval = MaxOfMany(Ixxeval, Iyyeval, Izzeval)
        'If the maxim values is also on X this axis is ok
                xok = True

                Else If Iyyeval = MaxOfMany(Ixxeval, Iyyeval, Izzeval)
                'If max is on y rotate eval part arround z axis until
they are on the same position, this is when the inertia on xx of the
asembly is xref+yeval
                        dif = IxxAsem - (Ixxref + Iyyeval)
                        'The value must have an error < 0.005
                        While dif>0.005
                                Rotate("Z Axis")
                                CenterPieces()
                                'The value of inertia assembly must be
actualiced on ech iteration.
                                momentAsem=inertiaAsem()
                                IxxAsem = momentAsem(0)
                                IyyAsem = momentAsem(1)
                                IzzAsem = momentAsem(2)
                                IxxAsem = Round(IxxAsem,3)
                                IyyAsem = Round(IyyAsem,3)
                                IzzAsem = Round(IzzAsem, 3)
                                dif= IxxAsem - (Ixxref + Iyyeval)

                        End While
                        xok = True
                Else If Izzeval = MaxOfMany(Ixxeval, Iyyeval, Izzeval)
                        dif=IxxAsem - (Ixxref + Izzeval)
                        While dif>0.005
                                Rotate("Y Axis")
                                CenterPieces()
                                momentAsem=inertiaAsem()
                                IxxAsem = momentAsem(0)
                                IyyAsem = momentAsem(1)
                                IzzAsem = momentAsem(2)
                                IxxAsem = Round(IxxAsem,3)
                                IyyAsem = Round(IyyAsem,3)
                                IzzAsem = Round(IzzAsem, 3)
                                dif = IxxAsem -( Ixxref + Izzeval)
```

```
                        End While
                        xok = True

                End If
'Same with Y axis
        Else If Iyyref = MaxOfMany(Ixxref, Iyyref, Izzref)
                If Iyyeval = MaxOfMany(Ixxeval, Iyyeval, Izzeval)
                        yok=True
                Else If Ixxeval = MaxOfMany(Ixxeval, Iyyeval, Izzeval)
                dif=IyyAsem - (Iyyref + Ixxeval)
                                While dif>0.005
                                        Rotate("Z Axis")
                                        CenterPieces()
                                        momentAsem=inertiaAsem()
                                        IxxAsem = momentAsem(0)
                                        IyyAsem = momentAsem(1)
                                        IzzAsem = momentAsem(2)
                                        IxxAsem = Round(IxxAsem,3)
                                        IyyAsem = Round(IyyAsem,3)
                                        IzzAsem = Round(IzzAsem, 3)
                                        dif=IyyAsem - (Iyyref + Ixxeval)
                                End While
                                yok = True
                Else If Izzeval = MaxOfMany(Ixxeval, Iyyeval, Izzeval)
                dif=IyyAsem -(Iyyref + Izzeval)
                                While dif>0.005
                                        Rotate("X Axis")
                                        CenterPieces()
                                        momentAsem=inertiaAsem()
                                        IxxAsem = momentAsem(0)
                                        IyyAsem = momentAsem(1)
                                        IzzAsem = momentAsem(2)
                                        IxxAsem = Round(IxxAsem,3)
                                        IyyAsem = Round(IyyAsem,3)
                                        IzzAsem = Round(IzzAsem, 3)
                                        dif=IyyAsem -(Iyyref + Izzeval)
                                End While
                                yok = True

                        End If

        'Same with Z
        Else If Izzref = MaxOfMany(Ixxref, Iyyref, Izzref)
                If Izzeval = MaxOfMany(Ixxeval, Iyyeval, Izzeval)

                                zok=True
                Else If Ixxeval = MaxOfMany(Ixxeval, Iyyeval, Izzeval)
                dif=IzzAsem-( Izzref + Ixxeval)
                                        While dif>0.005
                                                Rotate("Y Axis")
                                                CenterPieces()
                                                momentAsem=inertiaAsem()
                                                IxxAsem = momentAsem(0)
                                                IyyAsem = momentAsem(1)
                                                IzzAsem = momentAsem(2)
                                                IxxAsem =
Round(IxxAsem,3)
                                                IyyAsem =
Round(IyyAsem,3)
                                                IzzAsem = Round(IzzAsem,
3)
```

```
                                        dif=IzzAsem- (Izzref +
Ixxeval)
                                End While
                                zok = True
                Else If Iyyeval = MaxOfMany(Ixxeval, Iyyeval, Izzeval)
                dif=IzzAsem - (Izzref + Iyyeval)
                                While dif>0.005
                                        Rotate("X Axis")
                                        CenterPieces()
                                        momentAsem=inertiaAsem()
                                        IxxAsem = momentAsem(0)
                                        IyyAsem = momentAsem(1)
                                        IzzAsem = momentAsem(2)
                                        IxxAsem =
Round(IxxAsem,3)
                                        IyyAsem =
Round(IyyAsem,3)
                                        IzzAsem = Round(IzzAsem,
3)
                                        dif=IzzAsem - (Izzref +
Iyyeval)
                                End While
                                zok = True

                End If

        End If

'Star of min comprobation,same process than with maximum

        If Ixxref = MinOfMany(Ixxref, Iyyref, Izzref)
        'Luego el maximo de eval
                If Ixxeval = MinOfMany(Ixxeval, Iyyeval, Izzeval)
                xok=True
                Else If Iyyeval = MinOfMany(Ixxeval, Iyyeval, Izzeval)
                        dif= IxxAsem -(Ixxref + Iyyeval)
                        While dif>0.005
                                Rotate("Z Axis")
                                CenterPieces()
                                momentAsem=inertiaAsem()
                                IxxAsem = momentAsem(0)
                                IyyAsem = momentAsem(1)
                                IzzAsem = momentAsem(2)
                                IxxAsem = Round(IxxAsem,3)
                                IyyAsem = Round(IyyAsem,3)
                                IzzAsem = Round(IzzAsem, 3)
                                dif= IxxAsem -(Ixxref + Iyyeval)
                        End While
                        xok = True
                Else If Izzeval = MinOfMany(Ixxeval, Iyyeval, Izzeval)
                dif=IxxAsem - (Ixxref + Izzeval)
                        While dif>0.005
                                Rotate("Y Axis")
                                CenterPieces()
                                momentAsem=inertiaAsem()
                                IxxAsem = momentAsem(0)
                                IyyAsem = momentAsem(1)
                                IzzAsem = momentAsem(2)
                                IxxAsem = Round(IxxAsem, 3)
                                IyyAsem = Round(IyyAsem,3)
                                IzzAsem = Round(IzzAsem, 3)
```

```
                                dif=IxxAsem - (Ixxref + Izzeval)
                        End While
                        xok = True

                End If

        Else If Iyyref = MinOfMany(Ixxref, Iyyref, Izzref)
                If Iyyeval = MinOfMany(Ixxeval, Iyyeval, Izzeval)
                        yok=True
                Else If Ixxeval = MinOfMany(Ixxeval, Iyyeval, Izzeval)
                dif = IyyAsem - (Iyyref + Ixxeval)

                                While dif > 0.005
                'aqui
                                        Rotate("Z Axis")
                                        CenterPieces()
                                        momentAsem=inertiaAsem()
                                        IxxAsem = momentAsem(0)
                                        IyyAsem = momentAsem(1)
                                        IzzAsem = momentAsem(2)
                                        IxxAsem = Round(IxxAsem,3)
                                        IyyAsem = Round(IyyAsem,3)
                                        IzzAsem = Round(IzzAsem, 3)
                                        dif = IyyAsem - (Iyyref +
Ixxeval)
                                End While
                                yok = True
        Else If Izzeval = MinOfMany(Ixxeval, Iyyeval, Izzeval)
                dif=IyyAsem -(Iyyref + Izzeval)
                                While dif>0.005
                                        Rotate("X Axis")
                                        CenterPieces()
                                        momentAsem=inertiaAsem()
                                        IxxAsem = momentAsem(0)
                                        IyyAsem = momentAsem(1)
                                        IzzAsem = momentAsem(2)
                                        IxxAsem = Round(IxxAsem,3)
                                        IyyAsem = Round(IyyAsem,3)
                                        IzzAsem = Round(IzzAsem, 3)
                                        dif=IyyAsem -(Iyyref + Izzeval)
                                End While
                                yok = True

                        End If

        Else If Izzref = MinOfMany(Ixxref, Iyyref, Izzref)
                If Izzeval = MinOfMany(Ixxeval, Iyyeval, Izzeval)
                                zok=True
        Else If Ixxeval = MinOfMany(Ixxeval, Iyyeval, Izzeval)
                dif=IzzAsem - (Izzref + Ixxeval)
                                        While dif>0.005
                                                Rotate("Y Axis")
                                                CenterPieces()
                                                momentAsem=inertiaAsem()
                                                IxxAsem = momentAsem(0)
                                                IyyAsem = momentAsem(1)
                                                IzzAsem = momentAsem(2)
                                                IxxAsem =
Round(IxxAsem,3)
                                                IyyAsem =
Round(IyyAsem,3)
```

```
                                        IzzAsem = Round(IzzAsem,
3)
                                        dif=IzzAsem - (Izzref +
Ixxeval)
                                End While
                                zok = True
                Else If Iyyeval = MinOfMany(Ixxeval, Iyyeval, Izzeval)
                dif=IzzAsem - (Izzref + Iyyeval)
                                While dif>0.005
                                        Rotate("X Axis")
                                        CenterPieces()
                                        momentAsem=inertiaAsem()
                                        IxxAsem = momentAsem(0)
                                        IyyAsem = momentAsem(1)
                                        IzzAsem = momentAsem(2)
                                        IxxAsem =
Round(IxxAsem,3)
                                        IyyAsem =
Round(IyyAsem,3)
                                        IzzAsem = Round(IzzAsem,
3)
                                        dif=IzzAsem -( Izzref +
Iyyeval)
                                End While
                                zok = True
                End If

        End If
End While
'Here the axis must be located but the orientation maybe is wrong.
'To correct that uses the interference value between the parts
'Firts gets all the interference values and then locates the part on
the position that makes maximum this value.
Dim interferences(4) As Double
interferences(0) = interf()
Rotate("X Axis")
Rotate("X Axis")
CenterPieces()
interferences(1) = interf()
Rotate("X Axis")
Rotate("X Axis")

Rotate("Y Axis")
Rotate("Y Axis")
CenterPieces()
interferences(2) = interf()
Rotate("Y Axis")
Rotate("Y Axis")

Rotate("Z Axis")
Rotate("Z Axis")
CenterPieces()
interferences(3) = interf()
Rotate("Z Axis")
Rotate("Z Axis")
MaxInterf = MaxOfMany(interferences(0), interferences(1),
interferences(2), interferences(3))
If MaxInterf = interferences(0)

Else If MaxInterf = interferences(1)
        Rotate("X Axis")
```

```
        Rotate("X Axis")
        CenterPieces()
Else If MaxInterf = interferences(2)
        Rotate("Y Axis")
        Rotate("Y Axis")
        CenterPieces()
Else If MaxInterf = interferences(3)
        Rotate("Z Axis")
        Rotate("Z Axis")
        CenterPieces()
End If
End Function
```

Interf()

```
'No Arguments needed:
'Takes the two parts and checks the interference,
'volume the checked part will be geometrically equal
        'Returns a double .
        'Returns the interference volume
'This function is called by function locatepart to locate correctly
the axis

Function interf() As Double
'Creates variables needed to access all data
Dim oAsmDoc As AssemblyDocument
oAsmDoc = ThisApplication.ActiveDocument
Dim oAsmCompDef As AssemblyComponentDefinition
oAsmCompDef = oAsmDoc.ComponentDefinition
' Adds each occurrence in the assembly to the object collection.
Dim oCheckSet As ObjectCollection
oCheckSet= ThisApplication.TransientObjects.CreateObjectCollection
Dim oOccurrence As ComponentOccurrence
'adds all the parts to the object list
'if any of them is the reference, takes its volume

For Each oOccurrence In oAsmCompDef.Occurrences
        If oOccurrence.Name.Contains("Referencia")
                volumenref =
iProperties.VolumeOfComponent(oOccurrence.Name)
                volumenref = Round(volumenref /1000,5)

        End If
        oCheckSet.Add (oOccurrence)
Next
'takes interference volume of the two objects on the oCheckSet
Dim oResults As InterferenceResults
oResults = oAsmCompDef.AnalyzeInterference(oCheckSet)
For Each oResult In oResults
        volumeninterferencia=Round(oResult.Volume,5)
Next
'Compares the interference volume with the reference volume, if they
are equal returns 0
'If they aren't equal adds a message to sFinaltext and returns -1
Return volumeninterferencia
End Function
```

inertiaAsem()

```
'Takes the inertia values from the ipropieties of the assembly.
'This Function Is called By locate part.
Function inertiaAsem() As Double()
        Dim Asem As AssemblyDocument
        Asem = ThisApplication.ActiveDocument
        massPropAsem = Asem.ComponentDefinition.MassProperties
        Dim momentAsem(6) As Double
        massPropAsem.XYZMomentsOfInertia(momentAsem(0), momentAsem(1),
momentAsem(2), momentAsem(3),momentAsem(4), momentAsem(5))
        For i = 0 To 5

         momentAsem(i) =
Asem.UnitsOfMeasure.ConvertUnits(momentAsem(i), "kg cm^2", "kg mm^2")
        Next

        Return momentAsem
End Function
```

FinalPunctuation()

```
'Calculates the final punctuation for the model. Gets all the results
of the other functions and calculates a punctuation using the
parameters on the form
Function FinalPunctuation(ResEvalFeature() As Integer ,ResFeatVolum()
As Integer, ResConstr() As Integer,Resiterf As Double, ResTotalVolume
As Double,ResCountFeat As Integer,ByRef sFinalText As String)
        Dim featurePunctuation
        Dim geo = 34
        Dim Sketch = 33
        Dim Type = 33
        Dim medio = 0.5
        Dim Ok = 1
        Dim NOK = 0
        Dim Punctuationlist As New List(Of Double)
        iFeatCounter=SharedVariable("FeatCounter")
        iFeatCounter = iFeatCounter - 1
        For i As Integer = 0 To iFeatCounter
                p = i + 1
                'for the features that dont have sketch
                If ResConstr(i) = 3
                         geo = 50
                        Sketch = 0
                         Type = 50
                End If
                If ResEvalFeature(i) = 0 And ResFeatVolum(i)=0 And
ResConstr(i)=0 'All correct
                        featurePunctuation = Type * Ok + geo * Ok +
Sketch * Ok

                Else If ResEvalFeature(i) = -1 And ResFeatVolum(i)=-1
And ResConstr(i)=-1 'All bad or feature not found
                        featurePunctuation=Type*NOK+geo*NOK+ Sketch*NOK
                Else If ResEvalFeature(i) = 1 And ResFeatVolum(i)=0 And
ResConstr(i)=0 'Has a different feature ,same sketch
                        featurePunctuation=Type*medio+geo*Ok+ Sketch*Ok

                Else If ResEvalFeature(i) = 1 And ResFeatVolum(i) = -1
And ResConstr(i) = 0 'sketch ok and Feature valid but final result not
Ok
                        featurePunctuation=Type*medio+geo*NOK+
Sketch*Ok

                Else If ResEvalFeature(i) = 1 And ResFeatVolum(i) = 0
And ResConstr(i) = 1 'different feature and different constrait but
same final result
                        featurePunctuation=Type*medio+geo*Ok+
Sketch*medio

                Else If ResEvalFeature(i) = 1 And ResFeatVolum(i) = -1
And ResConstr(i) = -1 'Bad final result,not constrait but correct
feature
                        featurePunctuation=Type*medio+geo*NOK+
Sketch*NOK

                Else If ResEvalFeature(i) = 1 And ResFeatVolum(i) = 0
And ResConstr(i) = -1 'correct final result but different feature and
not constrait
                        featurePunctuation=Type*medio+geo*Ok+
Sketch*NOK
```

```
                Else If ResEvalFeature(i) = 1 And ResFeatVolum(i)=-1
And ResConstr(i)=1 'different feature, wrong final result ,different
constrits
                        featurePunctuation=Type*medio+geo*NOK+
Sketch*medio

                Else If ResEvalFeature(i) = 0 And ResFeatVolum(i)=-1
And ResConstr(i)=0 'correct feature,wrong final geometry and well
constrait
                        featurePunctuation=Type*Ok+geo*NOK+ Sketch*Ok

                Else If ResEvalFeature(i) = 0 And ResFeatVolum(i) = 0
And ResConstr(i) = 1 'Correct feature, correct final results, but not
same constraits
                        featurePunctuation=Type*Ok+geo*Ok+ Sketch*medio

                Else If ResEvalFeature(i) = 0 And ResFeatVolum(i)=-1
And ResConstr(i)=-1 'Only feature is ok
                        featurePunctuation=Type*Ok+geo*NOK+ Sketch*NOK

                Else If ResEvalFeature(i) = 0 And ResFeatVolum(i)=0 And
ResConstr(i)=-1 'Scketch bas constrait
                        featurePunctuation=Type*Ok+geo*Ok+ Sketch*NOK

                Else If ResEvalFeature(i) = 0 And ResFeatVolum(i)=-1
And ResConstr(i)=1 'Feature ok and fully constrit but different
consstraits and bad final result
                        featurePunctuation=Type*Ok+geo*NOK+
Sketch*medio

                Else
                        MsgBox("ERROR CALCULATING PUNCTUATION,CALL TO
THE DEVELOPER")
                End If

                Punctuationlist.Add(featurePunctuation)

        Next
        'General ckecks
        If ResCountFeat>sharedvariable("OtherFeaturesCount")
                SharedVariable("OtherFeaturePunctuation") = 0
        Else
        SharedVariable("OtherFeaturePunctuation") = 100 - ResCountFeat
* 100 / sharedvariable("OtherFeaturesCount")
        End If
        If SharedVariable("OtherFeaturePunctuation") <0 Or
SharedVariable("OtherFeaturePunctuation") >100
                SharedVariable("OtherFeaturePunctuation") = 0
        End If
        SharedVariable("Punctuations")=Punctuationlist
        sharedVariable("General")=Resiterf*ResTotalVolume/100
End Function
```

CreateNewExcel()

```
'Creates the excel with the necesay spaces and titles
'colum student- the name of the author of the parte
'Colum Filename- The nam of the file
'Features- All the features that the teacher selects
'General for the calification of general maching
'Other Features- Calification of other features count
'Failures found- the fails that the program found on the part
Function CreateNewExcel()

        Dim colum As Char
        Dim Box As String
        'To creates the new document on Desktop with the name of the
reference model uncomment
        'spath = "C:\Users\Usuario\Desktop\" + spath
        'If not the file will be saved on the same folder dan the
reference
        'Get the Inventor user name From the Inventor Options
        spath = SharedVariable("Path")
        iFeatCounter=SharedVariable("FeatCounter")
        'define Excel Application object
        excelApp = CreateObject("Excel.Application")
        'set Excel to run visibly, change to false if you want to run
it invisibly
        excelApp.Visible = False
        'suppress prompts (such as the compatibility checker)
        excelApp.DisplayAlerts = False
        'check for existing file
        If Dir(spath) <> "" Then
        'workbook exists, open it
        excelWorkbook = excelApp.Workbooks.Open(spath)
        'set the first sheet active
        excelSheet = excelWorkbook.Worksheets(1).activate
        Else
        'workbook does NOT exist, so create a new one
        excelWorkbook = excelApp.Workbooks.Add

        End If
        GoExcel.Save
        'complete the cells of the header
        With excelApp
                    .Range("A1").Select
                    .ActiveCell.Value = "Student"
                              .Range("B1").Select
                    .ActiveCell.Value = "File name"
                              colum = "C"

                              iFeatCounter=iFeatCounter-1
                              For counter As Integer = 0 To
iFeatCounter
                              FBox = colum & "1"
                              SBox = colum & "2"
                              colum = Chr(Asc(colum )+ 1)
                              .Range(FBox).Select
                    .ActiveCell.Value = "Feature:    " &
SharedVariable("sFeatureNames")(counter)
                              .Range(SBox).Select
                    .ActiveCell.Value = "Weight " &
SharedVariable("FeatureWeight")(counter)
                              Next
```

```
                                    SBox = colum & "2"
                                    FBox = colum & "1"
                                    .Range(SBox).Select
                        .ActiveCell.Value = "Weight: " &
Parameter("GlobalGeometry")
                                    .Range(FBox).Select
                                    .ActiveCell.Value = "General"
                                    colum = Chr(Asc(colum )+ 1)
                                    FBox = colum & "1"
                                    .Range(FBox).Select
                                    .ActiveCell.Value = "Other Features"
                                    SBox = colum & "2"
                                    .Range(SBox).Select
                        .ActiveCell.Value = "Weight: " &
Parameter("OtherFeatures")
                                    colum = Chr(Asc(colum )+ 1)
                                    FBox = colum & "1"
                                    .Range(FBox).Select
                                    .ActiveCell.Value = "Calification"
                                    colum = Chr(Asc(colum )+ 1)
                                    FBox = colum & "1"
                                    .Range(FBox).Select
                                    .ActiveCell.Value = "Failures found"


        End With

        'set all of the columns to autofit
        excelApp.Columns.AutoFit
        'save the file
        excelWorkbook.SaveAs (spath)
        'Close the file
        excelWorkbook.Close
        excelApp.Quit
        excelApp = Nothing


End Function
```

AddtoExcelfile()

```
'Add the data of each model to the excel
'Creates the excel with the necesay spaces and titles
'colum student- the name of the author of the parte
'Colum Filename- The nam of the file
'Features- All the features that the teacher selects
'General for the calification of general maching
'Other Features- Calification of other features count
'Failures found- the fails that the program found on the part
Function AddtoExcelfile(ByRef excelline As Integer,eval As
Document,ByRef FinalText As String)
'To creates the new document on Desktop with the name of the reference
model uncomment
'spath = "C:\Users\Usuario\Desktop\" + spath
'If not the file will be saved on the same folder dan the reference
'Get the Inventor user name From the Inventor Options

spath=SharedVariable("Path")

'define Excel Application object
excelApp = CreateObject("Excel.Application")
'set Excel to run visibly, change to false if you want to run it
invisibly
excelApp.Visible = True
'suppress prompts (such as the compatibility checker)
excelApp.DisplayAlerts = False
'check for existing file
If Dir(spath) <> "" Then
'workbook exists, open it
excelWorkbook = excelApp.Workbooks.Open(spath)
'set the first sheet active
excelSheet = excelWorkbook.Worksheets(1).activate
Else
'workbook does NOT exist, so create a new one
excelWorkbook = excelApp.Workbooks.Add

End If
GoExcel.Save

iFeatCounter=SharedVariable("FeatCounter")
        With excelApp
                Name = "A" & excelline.ToString
                Filename = "B" & excelline.ToString
                .Range(Name).Select
            .ActiveCell.Value = iProperties.Value(OccurenceName,
"Summary", "Author" )
                .Range(Filename).Select
            .ActiveCell.Value =
IO.Path.GetFileName(SharedVariable("PathList")(0))
                colum = "C"
                FinalCalification=0
                iFeatCounter=iFeatCounter-1
                For counter As Integer = 0 To iFeatCounter
                        SBox = colum & excelline.ToString
                        colum = Chr(Asc(colum )+ 1)
                                .Range(SBox).Select
                    .ActiveCell.Value =
SharedVariable("Punctuations")(counter)
```

```
                                FinalCalification=
FinalCalification+SharedVariable("Punctuations")(counter)*SharedVariab
le("FeatureWeight")(counter)
                Next
                SBox = colum & excelline.ToString
                .Range(SBox).Select
            .ActiveCell.Value = sharedVariable("General")
                colum = Chr(Asc(colum) + 1)
                SBox = colum & excelline.ToString
                .Range(SBox).Select
            .ActiveCell.Value =
SharedVariable("OtherFeaturePunctuation")
                FinalCalification=
FinalCalification+Parameter("GlobalGeometry")*sharedVariable("General"
)
                FinalCalification=
FinalCalification+SharedVariable("OtherFeaturePunctuation")*Parameter(
"OtherFeatures")
                colum = Chr(Asc(colum) + 1)
                SBox = colum & excelline.ToString
                .Range(SBox).Select
            .ActiveCell.Value = FinalCalification/100
                colum = Chr(Asc(colum) + 1)
                SBox = colum & excelline.ToString
                .Range(SBox).Select
            .ActiveCell.Value = FinalText
                FinalText = ""


        End With

excelline=excelline+1
'set all of the columns to autofit
excelApp.Columns.AutoFit
'save the file
excelWorkbook.SaveAs (spath)
'Close the file
excelWorkbook.Close
excelApp.Quit
excelApp = Nothing

End Function
```

deletepart()

```
Function deletepart( eval As Document)
Dim compOcc As ComponentOccurrence =
Component.InventorComponent(IO.Path.GetFileName(eval.fullfilename))
compOcc.Delete()
NewFile = IO.Path.GetFileName(SharedVariable("PathList")(0))
If SharedVariable("PathList")(0) <> "End"
Dim componentc = Components.Add(NewFile,
SharedVariable("PathList")(0), position := Nothing, grounded := False,
visible := True, appearance := Nothing)
End If
End Function
```

FileList()

```
'Functions to open all the files needed
'Create a list with the path of all the files on the same folder than
the eval document.
'Needs to take the path of a shared variable that is created on the
rule "select file to evaluate"
'At the end adds to the list the string "end" ass a refference to
indicate that the list is finished
Function FileList()
  Dim strFileSize As String = ""
        spath = SharedVariable("EvalPath")
    Dim di As New IO.DirectoryInfo(spath)
    Dim aryFi As IO.FileInfo() = di.GetFiles("*.ipt")
    Dim fi As IO.FileInfo
        Dim adaptiveStringList As New List(Of String)
    For Each fi In aryFi
                NewPath = spath + "\" + fi.Name
                adaptiveStringList.Add(NewPath)
    Next
        adaptiveStringList.Add("End")

        SharedVariable("PathList")=adaptiveStringList
End Function
```

RemoveListElement

```
'Removes a element from the list of paths .
'The program use this to remove the files that the program has just
evaluated
Function RemoveListElement(eval As Document)
        Dim adaptiveStringList As New List(Of String)
        adaptiveStringList=SharedVariable("PathList")
        adaptiveStringList.Remove(eval.fullfilename)
        SharedVariable("PathList")=adaptiveStringList
End Function
```

Regla “Select Reference File”

Sub main()

```
'This rule is for selecting the reference file and create the needed
parameter for the form .
'Is executed on the formulary as a button.

Sub main()
'First get file with windows explorer
'Create a file dialog
	Dim Filedialog As Inventor.FileDialog = Nothing
	ThisApplication.CreateFileDialog(Filedialog)
	Filedialog.CancelError = True
'File dialog Title
	Filedialog.DialogTitle = "Select reference file."
	Filedialog.InitialDirectory =
ThisApplication.DesignProjectManager.ActiveDesignProject.WorkspacePath
	Filedialog.InsertMode = False
	Filedialog.Filter = "Inventor Files
(*.iam;*.ipt)|*.iam;*.ipt|All Files (*.*)|*.*"
	Filedialog.MultiSelectEnabled = False
	Filedialog.OptionsEnabled = True
	Filedialog.ShowQuickLaunch = True 'only when opening, not when
inserting
	Filedialog.ShowOpen
	'Some errors
	If Err.Number <> 0 Then
		MsgBox("Dialog was cancelled.", vbOKOnly, " ")
		Return
	End If
	If Filedialog.FileName = "" Then
		MsgBox("No file was selected.", vbOKOnly, " ")
		Return
	End If
	'If the value taken from the file dialog is different from 0,
insert the component
	If Filedialog.FileName <> "" Then
		Dim pointA = ThisDoc.Geometry.Point(0,0,0)
		Dim componentA = Components.Add("Referencia",
Filedialog.FileName, position := pointA, _
		grounded := False, visible := True, appearance :=
Nothing)
	End If
'Shares the file name so it can be used on other rules.
	SharedVariable("RefFileName") =
IO.Path.GetFileNameWithoutExtension(Filedialog.FileName)
	'Function to create parameters for the form

	createselector()
	FeatureEntitiesCount()
	'Prepare to start the excel file
	spath = IO.Path.GetDirectoryName(Filedialog.FileName).ToString
+ "\" + IO.Path.GetFileNameWithoutExtension(Filedialog.FileName) +
".xlsx"
'Shares the file name so it can be used on other rules.
	SharedVariable("Path") = spath

	End Sub
```

createselector( )

```
Sub createselector( )
        Dim asmDoc As AssemblyDocument
        asmDoc = ThisApplication.ActiveDocument
        Dim doc As Document
        Dim adaptiveStringList As New List(Of String)
        'Get the all the feature name from the reference document and
save it on a list
        'This listis needed to create the drop down that appears on the
formulary to select the feature
        spath = SharedVariable("RefFileName")
        spath=IO.Path.GetFileNameWithoutExtension(spath)
        For Each doc In asmDoc.AllReferencedDocuments
                If doc.FullFileName.Contains(spath)
                Dim oFeature As PartFeature
                Dim oFeatures As PartFeatures

                oFeatures = doc.ComponentDefinition.Features

                For Each oFeature In oFeatures
                adaptiveStringList.Add(oFeature.Name)
                Next
        End If
        Next
        'Get access to the user params to create the params that are
necesary for the formulary
                Dim userParams As UserParameters _
                =
ThisDoc.Document.ComponentDefinition.Parameters.UserParameters
                Dim newParam As UserParameter
                adaptiveStringList.Add("None")
'Now start the creation of all the parameters
        Try
        'First  create the parameters to select the feature name to
evaluate
        'Checks if the parameter is created .

                oParam =
ThisDoc.Document.ComponentDefinition.Parameters("Feature1")
                oParam =
ThisDoc.Document.ComponentDefinition.Parameters("Feature2")
                oParam =
ThisDoc.Document.ComponentDefinition.Parameters("Feature3")
                oParam =
ThisDoc.Document.ComponentDefinition.Parameters("Feature4")
                oParam =
ThisDoc.Document.ComponentDefinition.Parameters("Feature5")
                oParam
=ThisDoc.Document.ComponentDefinition.Parameters("Feature6")
                oParam =
ThisDoc.Document.ComponentDefinition.Parameters("Feature7")
                oParam =
ThisDoc.Document.ComponentDefinition.Parameters("Feature8")
                oParam =
ThisDoc.Document.ComponentDefinition.Parameters("Feature9")
                oParam =
ThisDoc.Document.ComponentDefinition.Parameters("Feature10")
        Catch
```

```
                'Create the parameters and give the list of features as
value.
                '10 parameters in total.
                oNewParam = userParams.AddByValue("Feature1","None",
UnitsTypeEnum.kTextUnits)
                MultiValue.SetList("Feature1",
adaptiveStringList.ToArray)
                MultiValue.UpdateAfterChange = True
                oNewParam = userParams.AddByValue("Feature2","None",
UnitsTypeEnum.kTextUnits)
                MultiValue.SetList("Feature2",
adaptiveStringList.ToArray)
                MultiValue.UpdateAfterChange = True
                oNewParam = userParams.AddByValue("Feature3","None",
UnitsTypeEnum.kTextUnits)
                MultiValue.SetList("Feature3",
adaptiveStringList.ToArray)
                MultiValue.UpdateAfterChange = True
                oNewParam = userParams.AddByValue("Feature4","None",
UnitsTypeEnum.kTextUnits)
                MultiValue.SetList("Feature4",
adaptiveStringList.ToArray)
                MultiValue.UpdateAfterChange = True
                oNewParam = userParams.AddByValue("Feature5","None",
UnitsTypeEnum.kTextUnits)
                MultiValue.SetList("Feature5",
adaptiveStringList.ToArray)
                MultiValue.UpdateAfterChange = True
                oNewParam = userParams.AddByValue("Feature6","None",
UnitsTypeEnum.kTextUnits)
                MultiValue.SetList("Feature6",
adaptiveStringList.ToArray)
                MultiValue.UpdateAfterChange = True
                oNewParam = userParams.AddByValue("Feature7","None",
UnitsTypeEnum.kTextUnits)
                MultiValue.SetList("Feature7",
adaptiveStringList.ToArray)
                MultiValue.UpdateAfterChange = True
                oNewParam = userParams.AddByValue("Feature8","None",
UnitsTypeEnum.kTextUnits)
                MultiValue.SetList("Feature8",
adaptiveStringList.ToArray)
                MultiValue.UpdateAfterChange = True
                oNewParam = userParams.AddByValue("Feature9","None",
UnitsTypeEnum.kTextUnits)
                MultiValue.SetList("Feature9",
adaptiveStringList.ToArray)
                MultiValue.UpdateAfterChange = True
                oNewParam = userParams.AddByValue("Feature10","None",
UnitsTypeEnum.kTextUnits)
                MultiValue.SetList("Feature10",
adaptiveStringList.ToArray)
                MultiValue.UpdateAfterChange = True
        End Try
Try
        'Then the puntuation parameters
        'Geometry Sketch and Feature type for evaluate each selected
feature
        oParam =
ThisDoc.Document.ComponentDefinition.Parameters("Punctuation1")
```

```
        oParam =
ThisDoc.Document.ComponentDefinition.Parameters("Punctuation2")

        oParam =
ThisDoc.Document.ComponentDefinition.Parameters("Punctuation3")

        oParam =
ThisDoc.Document.ComponentDefinition.Parameters("Punctuation4")

        oParam =
ThisDoc.Document.ComponentDefinition.Parameters("Punctuation5")

        oParam =
ThisDoc.Document.ComponentDefinition.Parameters("Punctuation6")

        oParam =
ThisDoc.Document.ComponentDefinition.Parameters("Punctuation7")

        oParam =
ThisDoc.Document.ComponentDefinition.Parameters("Punctuation8")

        oParam =
ThisDoc.Document.ComponentDefinition.Parameters("Punctuation9")

        oParam =
ThisDoc.Document.ComponentDefinition.Parameters("Punctuation10")

        'Then a puntuation for the final result of the part
        oParam =
ThisDoc.Document.ComponentDefinition.Parameters("GlobalGeometry")
        Param =
ThisDoc.Document.ComponentDefinition.Parameters("OtherFeatures")
        'Finally a puntuation to check if the student has the same
number of some secundary features
        oParam =
ThisDoc.Document.ComponentDefinition.Parameters("CountFillet")
        oParam =
ThisDoc.Document.ComponentDefinition.Parameters("CountChamfer")
        oParam =
ThisDoc.Document.ComponentDefinition.Parameters("CountHole")
        oParam =
ThisDoc.Document.ComponentDefinition.Parameters("CountRib")
        oParam =
ThisDoc.Document.ComponentDefinition.Parameters("CountMirror")
        oParam =
ThisDoc.Document.ComponentDefinition.Parameters("CountCpattern")
        oParam =
ThisDoc.Document.ComponentDefinition.Parameters("CountRpattern")
        oParam =
ThisDoc.Document.ComponentDefinition.Parameters("CountShell")
        oParam =
ThisDoc.Document.ComponentDefinition.Parameters("CountSweep")


Catch 'If the parameter was not found, then create a new
one.initialice to 0
        oNewParam =
userParams.AddByValue("Punctuation1",0,UnitsTypeEnum.kUnitlessUnits)

        oNewParam =
userParams.AddByValue("Punctuation2",0,UnitsTypeEnum.kUnitlessUnits)
```

```
        oNewParam =
userParams.AddByValue("Punctuation3",0,UnitsTypeEnum.kUnitlessUnits)

        oNewParam =
userParams.AddByValue("Punctuation4",0,UnitsTypeEnum.kUnitlessUnits)

        oNewParam =
userParams.AddByValue("Punctuation5",0,UnitsTypeEnum.kUnitlessUnits)

        oNewParam =
userParams.AddByValue("Punctuation6",0,UnitsTypeEnum.kUnitlessUnits)

        oNewParam =
userParams.AddByValue("Punctuation7",0,UnitsTypeEnum.kUnitlessUnits)

        oNewParam =
userParams.AddByValue("Punctuation8",0,UnitsTypeEnum.kUnitlessUnits)

        oNewParam =
userParams.AddByValue("Punctuation9",0,UnitsTypeEnum.kUnitlessUnits)

        oNewParam =
userParams.AddByValue("Punctuation10",0,UnitsTypeEnum.kUnitlessUnits)

        oNewParam = userParams.AddByValue("GlobalGeometry", 0,
UnitsTypeEnum.kUnitlessUnits)
        oNewParam =
userParams.AddByValue("OtherFeatures",0,UnitsTypeEnum.kUnitlessUnits)
        oNewParam = userParams.AddByValue("CountFillet", 0,
UnitsTypeEnum.kUnitlessUnits)
        oNewParam = userParams.AddByValue("CountChamfer", 0,
UnitsTypeEnum.kUnitlessUnits)
        oNewParam = userParams.AddByValue("CountHole", 0,
UnitsTypeEnum.kUnitlessUnits)
        oNewParam =
userParams.AddByValue("CountRib",0,UnitsTypeEnum.kUnitlessUnits)
        oNewParam =
userParams.AddByValue("CountMirror",0,UnitsTypeEnum.kUnitlessUnits)
        oNewParam = userParams.AddByValue("CountCpattern", 0,
UnitsTypeEnum.kUnitlessUnits)
        oNewParam = userParams.AddByValue("CountRpattern", 0,
UnitsTypeEnum.kUnitlessUnits)
        oNewParam = userParams.AddByValue("CountShell", 0,
UnitsTypeEnum.kUnitlessUnits)
        oNewParam =
userParams.AddByValue("CountSweep",0,UnitsTypeEnum.kUnitlessUnits)

End Try
End Sub
```

FeatureEntitiesCount()

```
Function FeatureEntitiesCount()
        Dim asmDoc As AssemblyDocument
        asmDoc = ThisApplication.ActiveDocument
        Dim doc As Document
        For Each doc In asmDoc.AllReferencedDocuments
                Dim oFeature As PartFeature

                oFeatures = doc.ComponentDefinition.Features
        Next
        Dim oChamferCount = 0
        Dim oFilletCount = 0
        Dim oHoleCount = 0
        Dim oRibCount = 0
        Dim oSweepCount = 0
        Dim oCPatternCount = 0
        Dim oRPatternCount = 0
        Dim oMirrorCount = 0
        Dim oShellCount = 0
        Dim oTotalCount=0

        For Each oFeature In oFeatures
                Call oFeature.SetEndOfPart(True)
                If TypeOf (oFeature) Is RibFeature
                        oRibCount = oRibCount + 1

                End If
                If TypeOf (oFeature) Is SweepFeature
                        oSweepCount = oSweepCount + 1

                End If
                If TypeOf (oFeature) Is CircularPatternFeature

                        oCPatternCount = oCPatternCount + 1


                End If
                If TypeOf (oFeature) Is RectangularPatternFeature
                        oRPatternCount = oRPatternCount + 1

                End If
                If TypeOf (oFeature) Is MirrorFeature
                        oMirrorCount = oMirrorCount + 1

                End If
                If TypeOf (oFeature) Is ShellFeature
                         oShellCount = oShellCount + 1

                End If

                If TypeOf (oFeature) Is ChamferFeature

                        oChamferCount =oChamferCount+
oFeature.Definition.ChamferedEdges.Count

                End If
                If TypeOf (oFeature) Is FilletFeature
                        Call oFeature.SetEndOfPart(False)
```

```
                        oFilletCount =oFilletCount+
oFeature.Faces.Count

                End If
                If TypeOf (oFeature) Is HoleFeature
                                oHolesCount =oHolesCount+
oFeature.HoleCenterPoints.Count
                End If
                Call oFeature.SetEndOfPart(False)
        Next
        oTotalCount= oChamferCount+oFilletCount+oHolesCount+oRibCount
+oSweepCount+ oCPatternCount+oRPatternCount+oMirrorCount+oShellCount
        sharedvariable("OtherFeaturesCount")=oTotalCount
        Parameter("CountChamfer")= oChamferCount
        Parameter("CountFillet")= oFilletCount
        Parameter("CountHole")= oHolesCount
        Parameter("CountRib")= oRibCount
        Parameter("CountSweep") =oSweepCount
        Parameter("CountCpattern")= oCPatternCount
        Parameter("CountRpattern")= oRPatternCount
        Parameter("CountMirror") =oMirrorCount
        Parameter("CountShell")= oShellCount

End Function
```

Regla “Select file to evaluate”

Sub main()

```
Sub main()
Dim oFD As Inventor.FileDialog = Nothing
ThisApplication.CreateFileDialog(oFD)

oFD.CancelError = True
oFD.DialogTitle = "Seleccionar archivo a evaluar."
oFD.InitialDirectory =
ThisApplication.DesignProjectManager.ActiveDesignProject.WorkspacePath
' "C:\Temp\"
oFD.InsertMode = False
oFD.Filter = "Inventor Files (*.iam;*.ipt)|*.iam;*.ipt|All Files
(*.*)|*.*"
'oFD.FilterIndex = 1
oFD.MultiSelectEnabled = False
oFD.OptionsEnabled = True
'oFD.OptionValues
oFD.ShowQuickLaunch = True
oFD.ShowOpen

If Err.Number <> 0 Then
        MsgBox("Cancelar", vbOKOnly, " ")
        Return
End If
If oFD.FileName = "" Then
        MsgBox("No se ha seleccionado archivo", vbOKOnly, " ")
        Return
End If
If oFD.FileName <> "" Then

        oFFN = oFD.FileName
        oFN = IO.Path.GetFileName(oFFN)
        Dim pointA = ThisDoc.Geometry.Point(0,0,0)
        Dim componentB = Components.Add(oFN, oFD.FileName, position
:=pointA,grounded := False, visible := True, appearance := Nothing)
        SharedVariable("EvalPath")=IO.Path.GetDirectoryName(oFD.FileNam
e)
End If

End Sub
```

Esquema del programa