

# **Collaborative Computer Aided Design Application (CoCADA)**

*Desarrollo de una aplicación web colaborativa para revisiones de  
modelos sólidos paramétricos mediante el enfoque Lean UX*

**José María Guaimas, Marcos Daniel Henning**

**Director: Dr. Ing. Juan B. Cabral**  
**Co-Director: Ing. Héctor Ruidías**

## Agradecimientos

Antes que nada quisiéramos agradecer al director, Dr. Ing. J. B. Cabral y al co-director Ing. Héctor Ruidías; por el entusiasmo, la libertad y la paciencia que nos brindaron durante la planificación e implementación de este proyecto.

En segundo lugar a nuestras familias y amigos por la comprensión, consejos y compañía en estos años de estudio.

Y por último, y no menos importante, manifestar nuestro enorme respeto y aprecio por toda la comunidad FLOSS de las herramientas que utilizamos, por su invaluable generosidad a la hora de compartir conocimiento.

# Índice general

Resumen . . . . .	5
<b>1. Introducción</b>	<b>6</b>
1.1. Objetivos . . . . .	9
1.1.1. Objetivo general . . . . .	9
1.1.2. Objetivos Específicos . . . . .	9
<b>2. Sistemas CAD</b>	<b>10</b>
2.1. Introducción . . . . .	10
2.2. Diseño Paramétrico . . . . .	12
2.2.1. Diseño paramétrico mediante paquetes aplicativos . . . . .	13
2.2.2. Diseño paramétrico especificado en algoritmos . . . . .	14
2.3. Modelado Sólido . . . . .	15
2.3.1. Representación geométrica . . . . .	16
2.3.2. Visualización . . . . .	26
2.3.3. Diseño de Objetos . . . . .	28
2.4. Sistemas CAD . . . . .	32
<b>3. Desarrollo Colaborativo de Productos CAD con Lean UX</b>	<b>37</b>
3.1. Introducción . . . . .	37
3.2. Co-diseño . . . . .	38
3.3. Sistemas CAD colaborativos . . . . .	44
3.3.1. Soporte informático a la colaboración . . . . .	45
3.3.2. Sistemas de Modelado Sólido . . . . .	46
3.3.3. Gestión de Datos del Producto (PDM) . . . . .	48
3.3.4. Intercambio de datos CAD . . . . .	49
3.4. Lean UX . . . . .	54
3.5. Antecedentes . . . . .	61

<b>4. Resultados: CoCADA Un software para el diseño colaborativo con LeanUX</b>	<b>64</b>
4.1. Declaraciones . . . . .	64
4.1.1. Hipótesis . . . . .	65
4.1.2. Personas . . . . .	66
4.1.3. Funciones o funcionalidades . . . . .	67
4.2. Diseño Colaborativo . . . . .	68
4.3. PMV y experimentos . . . . .	69
4.4. Feedback . . . . .	70
4.4.1. Resultados y propuestas . . . . .	71
4.5. Sistema CoCADA . . . . .	73
4.5.1. Back-End . . . . .	73
4.5.2. Front-End . . . . .	78
4.5.3. Interacción entre Usuarios, Front-End y Back-End . . . . .	86
<b>5. Conclusiones</b>	<b>87</b>

## Resumen

El Desarrollo Colaborativo de Productos y el Co-Diseño son conceptos muy utilizados en las organizaciones, en especial, en las áreas que involucran el diseño y fabricación asistido por computadora *CAD/CAM*. La evolución de las tecnologías web ha permitido la colaboración entre personas dispersas geográficamente, de diferentes campos de especialización e incluso sin formación en diseño.

Al mismo tiempo, la diversidad de conocimientos trae como consecuencia algunos problemas en la gestión de los proyectos, entre ellos: la comunicación imprecisa, la múltiple interpretación de ideas y la complejidad para registrar el progreso o cambios en los diseños.

El presente trabajo propone el desarrollo de un prototipo de aplicación web que provee un marco para la colaboración multidisciplinaria mediante revisiones de modelos 3D. Estableciendo así, un tipo de comunicación entre los usuarios que va más allá de la geometría.

El software llamado *Colaborative CAD Application* (CoCADA) se desarrolla en base al enfoque *Lean UX* y utiliza tecnologías *Free Libre Open Source Software* (FLOSS).

**Palabras Claves:** Co-diseño, CAD/CAM, Diseño Paramétrico, Modelado Sólido, WPDM, FBDE, Javascript, Lean UX.

# Capítulo 1

## Introducción

Los sistemas de **diseño asistido por computadora** en inglés *Computer-aided design* (CAD) tienen una amplia trayectoria y han demostrado ser excelentes herramientas para el diseño de productos ([Chao and Wang, 2001](#)), como característica distintiva incorporan el paradigma de Diseño Paramétrico ([Davis, 2013](#)) que posibilita la modificación del diseño de manera sencilla y rápida mediante el ajuste de sus parámetros; sin necesidad de modelar todo nuevamente. En la industria de la manufactura estos sistemas son indispensables, ya que combinados con la **Fabricación Asistida por Computadora** en inglés *Computer-Aided Manufacturing* (CAM) y el **Control Numérico Computarizado** (CNC), permiten que la fabricación digital ([Chryssolouris et al., 2009](#)) se pueda aplicar a prácticamente cualquier producto. La innovación en este contexto es la vinculación directa entre el modelo CAD y su fabricación CAM.

En Argentina, tal es la interés en este campo, que desde el año 2013, el ministerio de ciencia, tecnología e innovación productiva ha impulsado diversas acciones para la difusión, capacitación y apoyo a proyectos de innovación, desarrollo y adopción de estas tecnologías. Sobre todo en el área de la impresión 3D ([Berman, 2012](#)), al ser una de las más difundidas y promisorias ([Ministerio de Ciencia, 2015](#)).

La masificación de la fabricación digital hace necesario el **desarrollo colaborativo de productos** en inglés *Collaborative Product Development* (CPD) ([Elfving, 2007](#)) entre grupos de expertos con diferentes competencias y muchas veces geográficamente dispersos. Este enfoque, también conocido como **co-diseño** ([Pérez García, 2014](#)) requiere de una comunicación efectiva en entornos distribuidos<sup>1</sup>; dado que los problemas en los procesos de diseño suelen ser ocasionados por errores en la

---

<sup>1</sup>Entorno en el que los sistemas informáticos en red colaboran aportando sus recursos.

interpretación de ideas entre los participantes.

Por otra parte, la mayoría de los sistemas CAD/CAM son aplicaciones de escritorio que necesitan ser instaladas obligatoriamente en cada computadora por separado. Las aplicaciones web son menos propensas a este hecho ya que el usuario accede a ellas sin necesidad de instalar el software en su dispositivo. El uso del CAD en la web se ha incrementado en los últimos años gracias a la estandarización de tecnologías como HTML5<sup>2</sup> y WebGL (Nyman, 2013) que permiten experiencias de usuario en inglés *User Experiencie* (UX) (Hartson and Pyla, 2012) similares a las de las aplicaciones de escritorio. «*Uno de los avances más recientes en este campo es la capacidad de las aplicaciones web para representar gráficos en 3D.*» (Waerner, 2012).

Recientemente han aparecido servicios que proporcionan funcionalidades de CAD y administración de proyectos de diseño en la nube (Mell et al., 2011). Un ejemplo es OnShape<sup>3</sup> que permite a los equipos colaborar mediante modelos compartidos desde la web. Sin embargo, esta plataforma está orientada a especialistas en el diseño de piezas mecánicas, siendo ideal para los ambientes de ingeniería y diseño industrial pero difícil de utilizar por personas sin formación en el área de modelado, en consecuencia, dificulta el co-diseño en términos de diversidad de competencias.

Esta situación provoca que los participantes opten por utilizar servicios colaborativos genéricos para comunicarse, como la mensajería instantánea, las redes sociales o videoconferencias mediante Skype<sup>4</sup> y comparten sus archivos utilizando Dropbox<sup>5</sup> o sistemas similares. La multiplicidad de medios produce sesgo en la comprensión de los proyectos a nivel general. «*Para un equipo es preferible contar con la colaboración integrada en una misma aplicación web*» (Alfaiate, 2017). Otras limitaciones de estos servicios son la falta de mecanismos para registrar el progreso o cambios de los diseños en todo momento y la imposibilidad de señalar o “marcar” los problemas detectados de forma precisa, de manera que se pueda informar sobre estos a los demás colaboradores.

---

<sup>2</sup><https://www.w3.org/TR/html52/>

<sup>3</sup><https://www.onshape.com/>

<sup>4</sup><https://www.skype.com/>

<sup>5</sup><https://www.dropbox.com/>

Ante los problemas planteados, una **aplicación web colaborativa y distribuida para el diseño de productos orientados a la fabricación digital** proporcionaría herramientas para facilitar el co-diseño entre participantes con diferentes competencias y mejorar la administración en términos del progreso o evolución de los diseños. La aplicación o prototipo de software lleva el nombre de **Aplicación de CAD Colaborativa** en inglés *Collaborative CAD Application* (CoCADA).

Este documento se encuentra organizado de las siguiente manera: En el **Capítulo 2** se explican las características de los sistemas CAD y los conceptos fundamentales de los modelos orientados a la fabricación digital. Luego se analizan las ventajas del uso de *LeanUX* ([Gotheff and Seiden, 2013](#)) para desarrollar un software colaborativo y distribuido (**Capítulo 3**), se enumera el problema y los detalles de la solución (CoCADA) explicando los componentes constitutivos y las principales funcionalidades implementadas (**Capítulo 4**). Finalmente, en el **Capítulo 5**, se realizan las conclusiones y se formulan directivas para trabajos futuros.

## **1.1. Objetivos**

### **1.1.1. Objetivo general**

Desarrollar un prototipo de sistema que facilite la colaboración en el proceso de diseño de productos entre personas con diferentes competencias.

### **1.1.2. Objetivos Específicos**

- Recabar bibliografía sobre sistemas CAD paramétricos, modelos 3D en la web, modelos orientados a la fabricación digital, sistemas colaborativos y distribuidos, metodología Lean UX.
- Analizar y describir soluciones de CAD existentes, sobre todo aquellas que se ofrecen como servicios en la nube.
- Indagar sobre tecnologías colaborativas para el diseño iterativo de modelos 3D.
- Investigar los requerimientos del sistema y los componentes de software a utilizar en el desarrollo.
- Diseñar la interfaz gráfica de usuario en inglés *Graphical User Interface* (GUI) de la aplicación utilizando el enfoque Lean UX.
- Diseñar un modelo de dominio de una capa de servicios.
- Desarrollar el prototipo de software con herramientas FLOSS ([Stallman et al., 2007](#))([Stallman](#)) según los requerimientos de la GUI y el Modelo de dominio.
- Realizar pruebas para experimentar en diferentes escenarios y evaluar los resultados.

# Capítulo 2

## Sistemas CAD

En este capítulo se describen los conceptos generales que involucran la representación de los modelos 3D, la visualización, la manipulación y las características para que puedan aplicarse en la fabricación digital. Al final del capítulo se describen algunos antecedentes de aplicaciones CAD de escritorio y web que incorporan estos conceptos.

### 2.1. Introducción

La interacción entre el diseñador y el modelo en un proceso de diseño no es una necesidad reciente, de hecho antecede la existencia de la computación. Desde principios del siglo XX se registran antecedentes como la maqueta que utilizó el arquitecto Antoni Gaudí<sup>1</sup> para representar el modelo de la cripta de la Colonia Guell<sup>2</sup>, esta se conformaba por cadenas que sostenían pesos y actuaban por la fuerza de la gravedad (Davis, 2013). El modelo fue realizado al revés y se sacó una fotografía para poder visualizarlo al derecho como se puede ver en la figura 2.1.

Una cadena que cuelga tiene por lo menos cuatro parámetros: su longitud, su peso y los dos puntos a los que está sujetada. Al estar colgando por la acción de la fuerza de gravedad adopta una forma curva, definida por la función explícita de los parámetros de la cadena. A pesar de ser análogo, este es un modelo paramétrico debido a la presencia de parámetros que controlan una forma derivada de una función (calculada por la acción de la gravedad). Así, el diseñador puede modificar un diseño

---

<sup>1</sup><http://www.antonigaudi.org/>

<sup>2</sup><http://www.gaudicoloniaguell.org/>

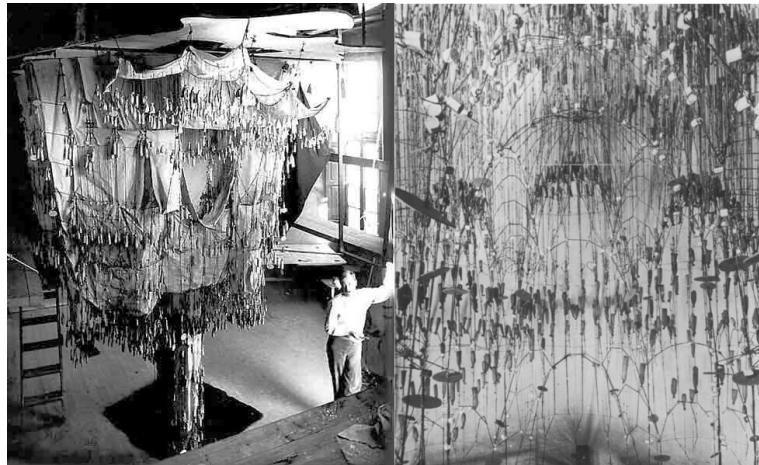


Figura 2.1: Maqueta gravitatoria de Gaudí para la cripta de la Colonia Guell. A la derecha una fotografía al revés donde se pueden apreciar los arcos catenarios utilizados para el modelo arquitectónico ([AA.VV, 2002](#)).

representado por un modelo visual de forma interactiva y en tiempo real, en este caso mediante arcos catenarios<sup>3</sup>.

No fue hasta la aparición de las computadoras y el primer programa CAD, **Sketchpad** ([Sutherland, 1963](#)) (ver figura 2.2) que se facilitó la interacción en tiempo real del diseñador y la computadora.



Figura 2.2: Ivan Sutherland utilizando sketchpad en 1963 ([Bethany, 2017](#)).

En este contexto, para comprender el uso de tecnologías CAD es fundamental

---

<sup>3</sup>Una catenaria es una curva ideal que representa físicamente la curva generada por una cadena, cuerda o cable sin rigidez flexional, suspendida de sus dos extremos y sometida a un campo gravitatorio uniforme

diferenciar los conceptos **computarización del diseño** y **diseño computacional**. El primero indica el uso de la computadora como herramienta de dibujo o representación formal orientado a disciplinas creativas, por ejemplo al realizar arte digital<sup>4</sup>, mientras que el diseño computacional aborda el diseño con bases en el pensamiento algorítmico, incluyendo el diseño paramétrico y el generativo (Kaled, 2016).

## 2.2. Diseño Paramétrico

El **Diseño Paramétrico** se entiende en términos generales como un proceso de descripción de una problemática utilizando variables. Actualmente para describir estas variables, los diseñadores introducen valores o algoritmos en un software especializado como AutoCAD<sup>5</sup>, al modificar las variables se generan una serie de alternativas de soluciones y según el criterio del diseñador, se obtiene la solución final. El diseño paramétrico en su definición contemporánea es únicamente posible creando un modelo paramétrico y se define como un conjunto de ecuaciones que expresan una geometría explícitamente por medio de funciones definidas por parámetros (Burry and Burry, 2012). Con la figura 2.3 se puede analizar el proceso general:

- Primero se realiza la abstracción de ideas y los conceptos del diseño.
- A partir de la abstracción se establecen las condiciones geométricas y matemáticas para dar soporte a las ideas iniciales.
- Del punto anterior derivan los parámetros y variables necesarios para programar el proceso.
- Finalmente, de la programación se obtiene la representación visual para explorar los resultados.

En todo momento el diseñador puede modificar las condiciones geométricas y matemáticas, los parámetros de la programación y explorar los resultados.

Mediante el **diseño iterativo** en inglés *iterative design* (Blokdyk, 2018) se analizan los resultados, se vuelve a trabajar y refinar el modelo diseñado hasta lograr una versión o solución aceptable.

---

<sup>4</sup>El arte digital engloba una serie de disciplinas creativas en las que se utilizan tecnologías digitales en el proceso de producción o en su exhibición

<sup>5</sup><https://latinoamerica.autodesk.com/products/autocad/overview>

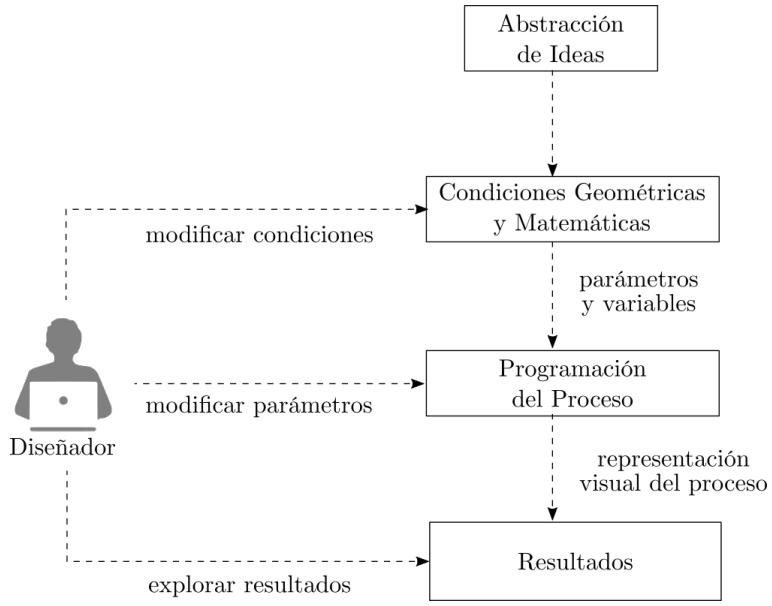


Figura 2.3: Proceso general del diseño paramétrico. A partir de la abstracción de ideas y conceptos del diseño se establecen las condiciones geométricas y matemáticas, de estas derivan los parámetros y variables que sirven para programar el proceso, y finalmente de la programación se obtiene la representación visual. En todo momento el diseñador puede modificar las condiciones geométricas y matemáticas, los parámetros de la programación y explorar los resultados. ([Bohnacker et al., 2012](#)).

Los modelos paramétricos permiten a los diseñadores alterar y modificar la geometría de manera eficiente sin tener que volver a crear el modelo. De esta manera, la parametrización puede elevar la calidad<sup>6</sup> y la reutilización de un modelo ([Alfaiate, 2017](#)). Este enfoque de diseño se lleva a la práctica utilizando **paquetes aplicativos** o bien mediante la **codificación por medio de algoritmos**.

### 2.2.1. Diseño paramétrico mediante paquetes aplicativos

Los paquetes aplicativos de CAD son aquellos que se instalan en el ordenador y no requieren del uso de internet, como AutoCAD. En este software, el diseño paramétrico se logra utilizando restricciones aplicadas a la geometría ([Autodesk, 2017](#)) y se clasifican en dos tipos: las **restricciones geométricas** que controlan las relaciones entre los objetos y las **restricciones por cota** que controlan los valores de distancia, longitud, ángulo y radio de los objetos. Proporcionan una manera de cumplir con ciertos requisitos que permiten experimentar con los diseños o hacer modificaciones. En el objeto ejemplo de la figura 2.4 las modificaciones en un parámetro pueden hacer cambios automáticamente a otros, a la vez se restringe las modifica-

<sup>6</sup>Capacidad que posee un objeto para satisfacer necesidades.

ciones de ciertos valores (como ser distancia y ángulo). Asimismo, las restricciones pueden aumentar su flexibilidad mediante el uso de fórmulas y ecuaciones.

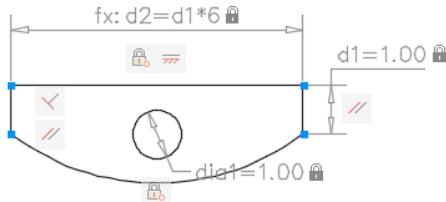


Figura 2.4: Restricciones en un diseño paramétrico hecho con AutoCAD. Se puede apreciar una restricción mediante la fórmula  $d2 = d1 * 6$  de manera que  $d2$  siempre será 6 veces el tamaño de  $d1$  (las variables están relacionadas). Por otro lado, el valor del diámetro  $dia1$  es independiente a cualquier otra variable. ([Autodesk, 2017](#)).

### 2.2.2. Diseño paramétrico especificado en algoritmos

Las **interfaces de secuencias de comandos** en inglés *scripting*<sup>7</sup> permiten a los diseñadores/programadores escribir código para automatizar partes del diseño.

El script, con sus parámetros de entrada, funciones explícitas y salidas es una realización arquetípica de la definición matemática de paramétrico ([Burry and Burry, 2012](#)). Los **sistemas paramétricos** se basan principalmente en **principios algorítmicos**, dado que al igual que los algoritmos, toman un valor o un conjunto de valores como entrada, ejecutan una serie de pasos computacionales que transforman la entrada y finalmente producen un valor o un conjunto de valores como salida ([Dino, 2012](#)). Por lo tanto, las interfaces de scripting disponibles en gran parte de los paquetes CAD están naturalmente predispostas para generar modelos paramétricos.

Independientemente de las aplicaciones que se utilicen para el diseño, los modelos resultantes son útiles para realizar estudios que de otra manera serían difíciles, por ejemplo al no contar con el objeto físico. En este aspecto, entra en juego el **Modelado Geométrico** ([Ramos, 2011](#)), haciendo referencia al conjunto de métodos utilizados para definir la forma y otras características de los objetos. Estos métodos son un compendio de las técnicas utilizadas en varias disciplinas, como la Geometría Analítica y Descriptiva, la Topología, la Teoría de Conjuntos, el Análisis Numérico,

<sup>7</sup>Un script es un programa informático usualmente simple, que por lo general se almacena en un archivo de texto plano.

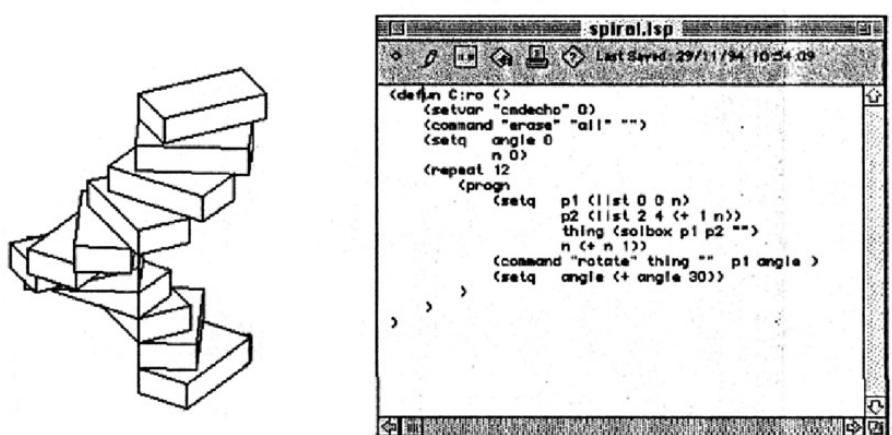


Figura 2.5: Programa en código AutoLisp para generar una espiral 3D con bloques simples mediante expresiones anidadas (derecha). A la izquierda se puede apreciar el modelo resultante del script ([Celani, 2008](#)).

las Estructuras de Datos, el Cálculo Vectorial y los Métodos Matriciales. Las aplicaciones de estas técnicas abarcan la **Representación, Visualización y Diseño** de los objetos.

### 2.3. Modelado Sólido

El **Modelado Sólido** ([Fol, 1993](#)) es una rama del modelado geométrico que hace hincapié en la aplicabilidad general de los modelos, e insiste en crear solamente modelos “completos” de los sólidos, es decir, modelos que son adecuados para responder algorítmicamente (sin la ayuda externa del usuario) a cualquier pregunta geométrica que se formule. Se espera que respondan preguntas geométricas típicas que aparecen en las aplicaciones de ingeniería. Por ejemplo: ¿Cuál es el aspecto del objeto?, ¿Cómo puede fabricarse con los procesos de manufacturación disponibles? La respuesta podría ser una imagen, un número o una constante booleana. De hecho, incluso podría ser otro modelo sólido ([Ramos, 2011](#)).

Las **ventajas prácticas del modelado sólido** son:

- Agiliza el desarrollo y los detalles del diseño.
- Mejora la visualización y la comunicación del diseño.
- Elimina los problemas de interferencias del diseño.
- Comprueba la funcionalidad y el rendimiento del diseño (sin la necesidad de prototipos físicos).

- Proporciona de forma automática las características topológicas para la fabricación digital, necesarias al programar maquinas herramientas de CNC, impresoras 3D, etc.

Un sistema de Modelado Sólido maneja dos tipos de información: los **datos geométricos** y los **datos topológicos**. Los datos geométricos son aquellos que representan geométricamente los objetos (coordenadas de vértices, ecuaciones de superficies, etc. En cambio, los topológicos se refieren a cómo conectar componentes geométricos para conseguir un modelo ([Ramos, 2011](#)).

### 2.3.1. Representación geométrica

Muchos esquemas de representación y particularmente los modelos sólidos utilizan **poliedros** ([Cromwell, 1999](#)) con caras, aristas y vértices. A bajo nivel, todos los algoritmos que se utilizan se basan en una única primitiva<sup>8</sup>: el **polígono**. Internamente los polígonos se dividen en elementos más simples: **triángulos**.

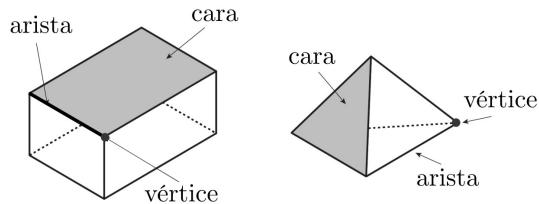


Figura 2.6: Caras, aristas y vértices en 2 poliedros diferentes: prisma rectangular (izquierda) y tetraedro (derecha).

## Transformaciones 3D

En términos matemáticos, es común encontrar la representación de los modelos por medio de **Matrices** ([Jeronimo, 2008](#)) (espacio vectorial 3D), estos deben someterse a ciertas transformaciones antes de que su imagen aparezca en la pantalla de un dispositivo. Las transformaciones geométricas se definen como la relaciones de los puntos entre dos imágenes, son operaciones matriciales sobre los puntos del objeto. Cada objeto se representa como una matriz constituida por las coordenadas (x, y, z) de los puntos que lo conforman ([Villamarin, 2015](#)).

---

<sup>8</sup>Las formas geométricas se denominan primitivas por su básica constitución en las partes que la conforman.

Las transformaciones básicas son: **Traslación 3D**, **Rotación 3D**, **Escalamiento 3D** y la composición de las mismas para lograr secuencias de transformaciones. La traslación mueve un objeto con una trayectoria en línea recta de una posición a otra. La rotación mueve un objeto de una posición a otra a lo largo de una trayectoria circular sobre un eje de rotación específico.

A modo de ejemplo se explica Escalamiento 3D:

El escalamiento permite cambiar el tamaño de un objeto expandiéndolo o contrayéndolo en sus dimensiones. Esto implica el cambio de tamaño de un poliedro, donde cada punto  $p = (x, y, z)$  es transformado por la multiplicación de tres factores de escalamiento:  $s_x, s_y$  y  $s_z$  a lo largo de los ejes  $X, Y, Z$  respectivamente, de esta forma, las coordenadas del nuevo punto  $p' = (x', y', z')$  se obtienen como:

$$\begin{aligned} x' &= x \cdot s_x \\ y' &= y \cdot s_y \\ z' &= z \cdot s_z \end{aligned}$$

Sea  $s = (s_x, s_y, s_z)$  el vector de factores de escalamiento, y  $S(s)$  la matriz de escalamiento, en coordenadas homogéneas ([Santaló, 1966](#)) el escalamiento de un punto  $p$  en 3D se puede expresar como el producto matricial  $p' = p \cdot S(s)$ , es decir:

$$\left[ \begin{array}{cccc} x' & y' & z' & 1 \end{array} \right] = \left[ \begin{array}{cccc} x & y & z & 1 \end{array} \right] \cdot \left[ \begin{array}{cccc} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{array} \right] \quad (\text{Ec.1})$$

Expresión matricial para el escalamiento 3D.

En la figura [2.7](#) se puede apreciar el resultado de un escalamiento con los valores  $s_x = 2$ ,  $s_y = 2,5$  y  $s_z = 1,5$ .

El **visor** o cámara puede ser considerado como un objeto más, respecto a las transformaciones lineales. Sin embargo, el movimiento de los visores tiene sus propias peculiaridades, por lo que conviene estudiar de modo independiente las transformaciones lineales que se aplican a estos objetos. Por ejemplo: El cambio de escala en el visor produce el efecto zoom (acercar/alejar) ([Ramos, 2011](#)).

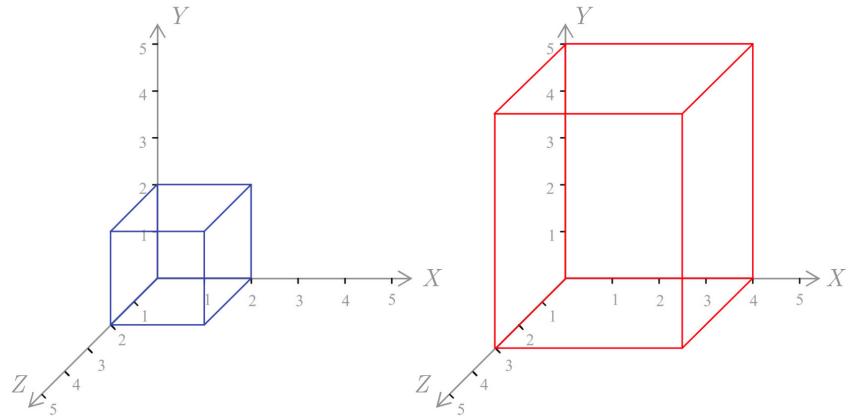


Figura 2.7: Escalamiento de cubo (izquierda): aplicando los factores  $s_x = 2$ ,  $s_y = 2,5$  y  $s_z = 1,5$  se obtiene el objeto de la derecha. Al aplicar diferentes factores se produce una variación en las proporciones del objeto original. ([Matías, 2007](#)).

### Topología de los modelos sólidos

Conocer las características topológicas de los poliedros es importante para la construcción de los modelos sólidos y especialmente para su validación.

Los sólidos están definidos en el espacio Euclídeo  $E^3$ , ocupando una determinada porción de ese espacio. Por lo tanto, se considera un sólido como un conjunto de  $E^3$ . Obviamente no todos los subconjuntos del espacio Euclídeo son sólidos (por ejemplo un punto aislado no es un sólido). Para que un conjunto de puntos represente un sólido a nivel abstracto, deberá satisfacer las siguientes condiciones ([Torres, 2014](#)):

- **Ser cerrado y acotado.** Esto significa que contiene a su frontera (o borde) y ocupa una porción finita del espacio.
- **Ser rígido.** Los sólidos no se modifican al trasladarlos o rotarlos. O lo que es lo mismo, dos sólidos que se diferencian tan solo en una transformación son el mismo sólido.
- **Tener Homogeneidad tridimensional.** El sólido debe ser tridimensional en todos sus puntos, es decir, en cualquier punto de él debe ser posible en tres direcciones ortogonales.

A continuación se estudia la topología de algunos poliedros que satisfacen las condiciones para el modelado sólido: Los poliedros simples, no simples y los objetos de Euler.

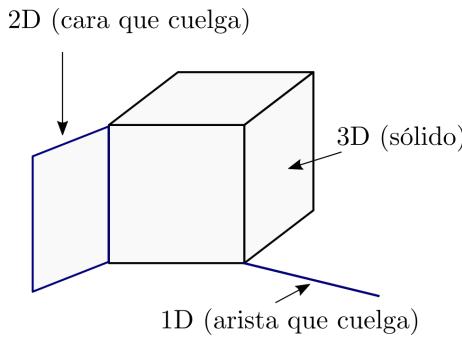


Figura 2.8: Objeto que no tiene homogeneidad tridimensional (no sólido). Contiene elementos 2D (cara que cuelga) y 1D (arista que cuelga) ([Torres, 2014](#)).

### Poliedros simples y no simples

Un **poliedro simple** es aquél que puede ser transformado de forma continua en una esfera, o sea, que es topológicamente equivalente a una esfera.

La **Fórmula de Euler** para los poliedros simples establece que «*para un poliedro simple se cumple que el número de vértices V, menos el número de aristas A, más el número de caras C*» [Ramos \(2011\)](#) es igual a 2, es decir:

$$V - A + C = 2 \quad (\text{Ec.2})$$

Los poliedros regulares forman un subconjunto de los poliedros simples, cuya característica principal es que todas sus caras (polígonos) son iguales, según la fórmula anterior se puede demostrar que sólo existen 5 poliedros regulares: tetraedro, hexaedro o cubo, octaedro, dodecaedro e icosaedro.

Los **poliedros no simples** son los equivalentes topológicos de cualquier objeto sólido con huecos, por lo que son muy útiles en el Modelado Sólido ([Ramos, 2011](#)).

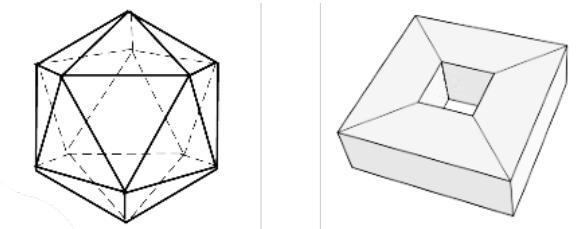


Figura 2.9: A la izquierda un poliedro simple regular (icosaedro): se puede transformar en una esfera. A la derecha un poliedro no simple (contiene un hueco): no se puede transformar en una esfera, pero sí en un toro.

Para clasificar los poliedros se recurre al concepto de **número de conectividad** ( $n$ ). Si la superficie de un poliedro se puede dividir en dos regiones separadas mediante un camino cerrado trazado a lo largo de sus aristas, se dice que tiene co-

nectividad  $n = 0$ . La esfera es un cuerpo que siempre cumple con esa condición, por ende, cualquier poliedro de conectividad 0 (simple) puede ser transformado en una esfera. Este fenómeno se aprecia con el icosaedro ilustrado a la izquierda de la figura 2.9. Por otro lado, el objeto de la derecha cuenta con caminos cerrados que no dividen a su superficie en dos partes separadas. A estos poliedros se les asigna un número de conectividad mayor que 0 y por ende, al contener huecos se los denomina **no simples**.

Si bien los poliedros simples y no simples son sólidos, la fórmula de Euler establece condiciones necesarias, pero no suficientes para que un objeto sea un poliedro. Se pueden construir objetos que satisfagan la fórmula, pero que no acoten un volumen, sin más que añadir una o más caras o aristas colgantes. **Se requieren de otras restricciones para garantizar que un objeto sea un sólido.**

### Objetos de Euler

$g$  se conoce como **orden o grado del objeto** y también como **orden de la superficie**. Así, una esfera equivale a  $g = 0$ , un toro, topológicamente hablando, puede considerarse como una esfera con 1 asa ( $g = 1$ ) y así sucesivamente según la cantidad de huecos que tenga el objeto. En la figura 2.10 se puede apreciar 3 objetos diferentes que se representan como **esferas con asas** ( $g = 0, g = 1$  y  $g = 2$ ) respectivamente.

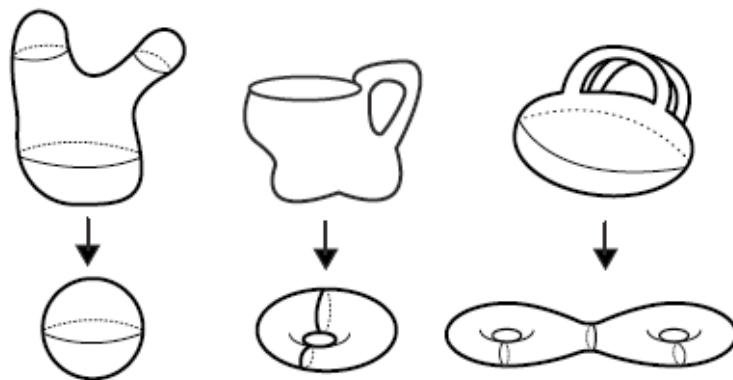


Figura 2.10: Objetos diferentes que se representan como esferas con  $g$  asas ( $g = 0, g = 1$  y  $g = 2$ ) respectivamente. Las asas representan la cantidad de huecos en la superficie de los objetos (Learner, 2017).

Los objetos cerrados tridimensionales de orden  $g \geq 0$  que verifican ciertos requisitos de construcción se denominan **Objetos de Euler**. Estos requisitos son:

- Todas sus caras (curvas o planas) han de ser discos topológicos<sup>9</sup>, es decir que no tienen huecos ni puntos aislados en ellas.
- Cada arista une sólo dos caras y todas finalizan en un vértice en cada extremo.
- Como mínimo tres aristas se unen en un vértice.

En todos los objetos de Euler se cumple que:

$$V - A + C = 2(S - P) \quad (\text{Ec.3})$$

$S$ : número de superficies inconexas del objeto.

$P$ : total de pasajes (túneles) en el objeto.

Como el número de pasajes en un objeto es igual al total de “asas” o agujeros que posea, ocurre que en la ecuación anterior  $P$  es igual al orden del objeto ( $P = g$ ). Para entender mejor el significado de  $S$ , la figura 2.11 muestra tres objetos eulerianos, con una, dos y tres superficies inconexas.

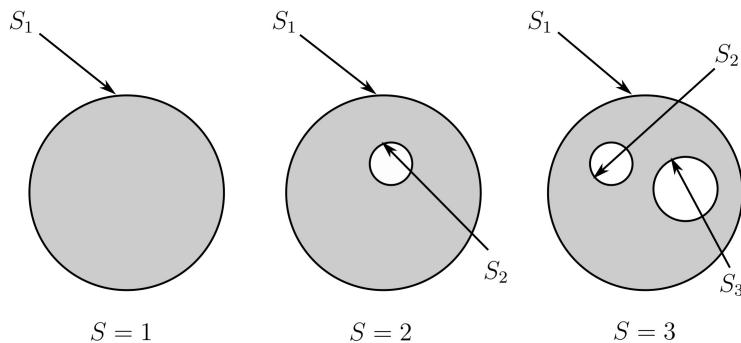


Figura 2.11: Objetos de Euler con diferentes valores de  $S$  (número de superficies inconexas):  $S = 1$ ,  $S = 2$  y  $S = 3$  respectivamente (Ramos, 2011)

En la figura 2.12 se pueden tres ejemplos de objetos eulerianos, a pesar de que cuentan son diferente número de vértices  $V$ , aristas  $A$  y caras  $C$ , cuando  $S = 1$  y  $P = 0$  (no poseen huecos) la Ec.3 se reduce a la ecuación de Euler para poliedros simples Ec.2. Por ende, estos objetos se representan como poliedros simples.

Por otro lado, en la figura 2.13 se ilustran dos objetos de Euler con diferentes  $P = 0$ . El objeto de la izquierda, aunque posee una cavidad, esta no llega a atravesar el modelo  $P = 0$ . Por lo tanto, se trata de un objeto sólido topológicamente

---

<sup>9</sup>Una región, plana o no, que pueda ser transformada en un cuadrado se llama disco topológico y se caracteriza porque no tiene huecos ni puntos aislados en él

The figure shows three objects: a tetrahedron, a cube, and a rectangular prism with a horizontal cut through its center. Below each object is a mathematical equation showing the result of applying Euler's formula.

 $V - A + C = 2$ $5 - 8 + 5 = 2$	 $V - A + C = 2$ $8 - 12 + 6 = 2$	 $V - A + C = 2$ $10 - 15 + 7 = 2$
--	---	--

Figura 2.12: Ejemplos de objetos de Euler con diferentes números de vértices  $V$ , aristas  $A$  y caras  $C$ . Se puede verificar que en los tres casos se cumple la igualdad. Al no contener huecos se consideran poliedros simples ([Ramos, 2011](#)).

equivalente a una esfera, es decir, de grado  $g = 0$ .

Sin embargo, el objeto de la derecha es de grado  $g = 1$ , ya que tiene un túnel que lo cruza  $P = 1$  y es un sólido topológicamente equivalente a un toro.

The figure shows two complex polyhedral structures. The left one is a cube with a central vertical tunnel, and the right one is a cube with a central cavity. Below each diagram is a mathematical equation showing the result of applying Euler's formula.

 $V - A + C = 2(S - P)$ $16 - 28 + 14 = 2(1 - 0)$	 $V - A + C = 2(S - P)$ $16 - 32 + 16 = 2(1 - 1)$
---	---

Figura 2.13: Objetos eulerianos de  $g = 0$  y  $g = 1$ , respectivamente. El sólido de la izquierda tiene un túnel que lo atraviesa  $P = 1$ , se puede representar topológicamente como un toro. El de la derecha posee una cavidad, pero esta no llega a atravesar el modelo  $P = 0$ , es un sólido que se puede representar topológicamente como una esfera [Ramos \(2011\)](#).

## Representación por atlas

Uno de los puntos clave en el modelado con poliedros es cómo se registra su información topológica, es decir, las relaciones entre los vértices, aristas y caras que los definen. La forma más simple y directa para representar los poliedros consiste en describir cada cara por separado, señalando explícitamente cómo han de unirse. Este método de representación se conoce como **atlas**. La figura 2.14 muestra el atlas de un cubo.

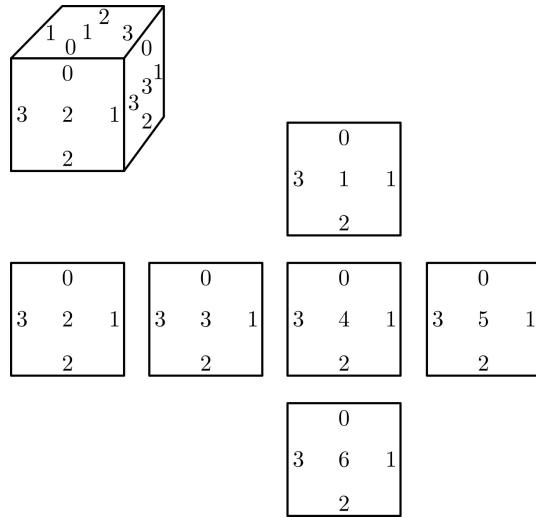


Figura 2.14: Atlas de un cubo. El valor del centro indica el número de cara. Los números cercanos al borde indican el número de arista correspondiente a cada cara. ([Ramos, 2011](#)).

Cada eje se etiqueta con un par de números indicando la cara y el número de la arista de la cara. Así el par (1, 0) se interpreta como la arista 0 de la cara 1. Cada unión de dos aristas se especificará por dos pares de números, identificando las aristas que se unen. De esta forma, el atlas formado queda como:

$$\begin{array}{llll} [(1, 0) \quad (2, 0)] & [(1, 1) \quad (5, 0)] & [(1, 2) \quad (4, 0)] & [(1, 3) \quad (3, 0)] \\ [(2, 1) \quad (3, 3)] & [(2, 3) \quad (6, 2)] & [(2, 3) \quad (5, 1)] & [(3, 1) \quad (4, 3)] \\ [(3, 2) \quad (6, 3)] & [(4, 1) \quad (5, 3)] & [(4, 2) \quad (6, 0)] & [(5, 2) \quad (6, 1)] \end{array}$$

Representación matricial del atlas

Además de especificar qué aristas están unidas, se debe decir cómo unirlas. La figura 2.15 muestra dos formas diferentes de unir un par de aristas. La primera, mostrada a la izquierda, es la unión con **orientación conservadora**, y la segunda tiene **orientación inversora**. Para unir esas dos aristas hay que hacer coincidir los números de cada una de ellas. Si se hace mediante la orientación conservadora se obtiene un cilindro y con la orientación inversora una **cinta de Möbius**<sup>10</sup>.

Para especificar en el atlas de qué forma se van a unir las aristas, se utiliza la **paridad de transición** que es número  $\pm 1$  (+ 1 si es conservadora, -1 si es inversora), unido al par de aristas. La figura 2.16 presenta tres ejemplos de esta notación.

---

<sup>10</sup>La cinta de Möbius o Moebius es una superficie con una sola cara y un solo borde. Tiene la propiedad matemática de ser un objeto no orientable.

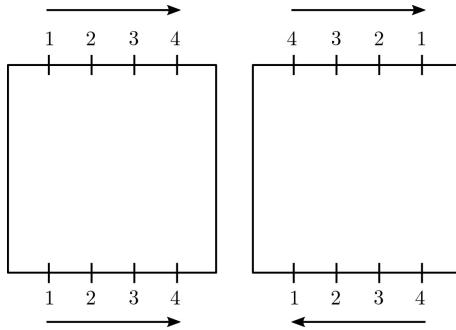


Figura 2.15: Orientación conservadora (izquierda), se obtiene como resultado un cilindro. Orientación inversora (derecha), se obtiene una cinta de moebius ([Ramos, 2011](#)).

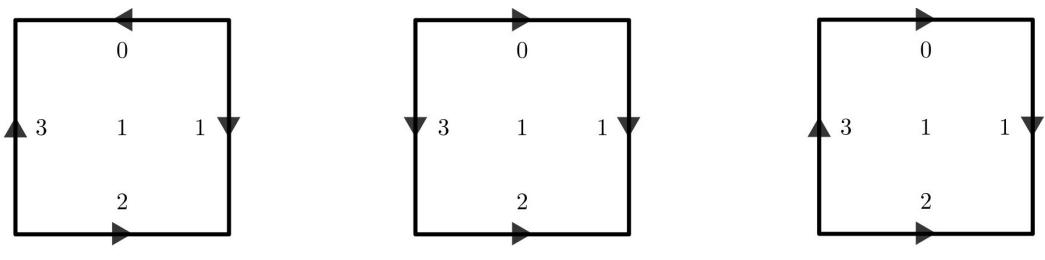


Figura 2.16: Paridad de transición de una. Esfera (izquierda). Toro (centro). Botella de Klein (izquierda), se puede observar la paridad de transición  $-1$  ([Ramos, 2011](#)).

Si se utilizan uniones no conservadoras en un atlas se consiguen algunas curiosidades matemáticas llamadas **superficies no orientables**, como la **cinta de Moebius** y la **botella de Klein**<sup>11</sup> (ver figura 2.17). La cinta de Moebius posee una curiosa particularidad. Si se recorre la cinta comenzando en un punto cualquiera, y se da una vuelta completa, como se trata de una cinta bidimensional, cuando se llegue al punto de partida se observará que la derecha e izquierda están intercambiadas. Si se parte de una cinta de Moebius y se cierra aplicando una orientación conservadora, se obtiene una botella de Klein (no orientable), que no se puede construir en un espacio tridimensional sin que se auto interseque. Este tipo de superficies se llaman **no orientables** y se producen al efectuar uniones no conservadoras entre sus aristas. Por el contrario, si en la superficie nunca se intercambian la izquierda y derecha, entonces se dice que la superficie es **orientable**.

«Para que cualquier superficie cerrada tridimensional sea consistente, debe ser topológicamente equivalente a una esfera con “g” asas. Ahora se añade otra condi-

---

<sup>11</sup>En topología, una botella de Klein es una superficie no orientable abierta cuya característica de Euler es igual a 0; no tiene interior ni exterior.

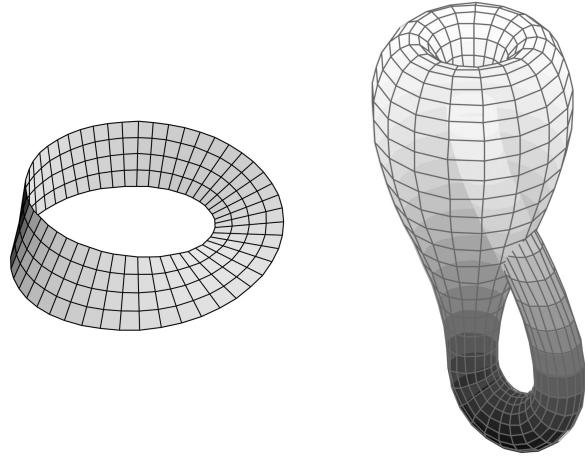


Figura 2.17: **Superficies no orientables.** Cinta de Moebius (izquierda) y Botella de Klein (derecha) ([Ramos, 2011](#)).

ción más: *debe ser orientable»* ([Ramos, 2011](#)).

### Representación mediante operadores Booleanos.

Para construir objetos sólidos complejos se pueden utilizar operaciones entre conjuntos, tales como unión ( $\cup$ ), intersección ( $\cap$ ) y diferencia ( $-$ ). A estos operadores se les conoce en modelado como **Operadores Booleanos** ([Ramos, 2011](#)). Al aplicarlos sobre objetos simples se pueden construir modelos más complejos. Es fundamental que los modelos obtenidos sean íntegros (topológicamente válidos), y que además sean **dimensionalmente homogéneos**, es decir, que su dimensión sea igual a la de los objetos que se combinan. En la figura 2.18 se pueden apreciar los resultados de las operaciones booleanas entre una esfera y un cubo.

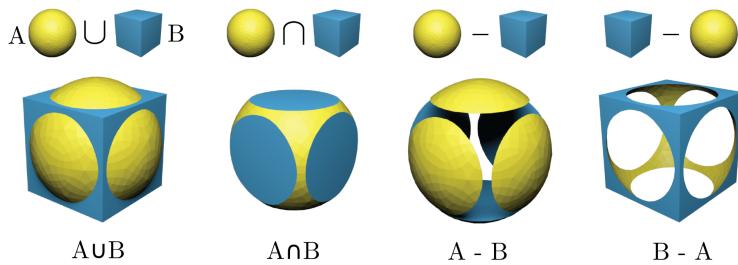


Figura 2.18: Operaciones booleanas entre objetos A(esfera) y B(cubo). Resultados de  $A \cup B$ ,  $A \cap B$ ,  $A - B$  y  $B - A$  respectivamente ([Zhou, 2018](#)).

Sin embargo, la aplicación de una operación booleana a objetos sólidos no necesariamente produce otro objeto sólido. En la figura 2.19 se muestra un ejemplo

donde se pierde la homogeneidad dimensional. Para evitar los resultados no sólidos se utiliza el conjunto de **operadores regularizados** (Ramos, 2011), que preservan la homogeneidad dimensional. Los operadores regularizados se representan por  $\cup^*$ ,  $\cap^*$  y  $-^*$  y se definen de manera que las operaciones con sólidos siempre generen sólidos.

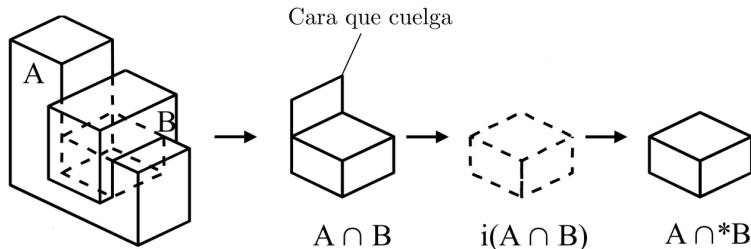


Figura 2.19: Intersección no regularizada ( $\cap$ ) y regularizada ( $\cap^*$ ) de dos sólidos. La intersección  $A(\cap)B$  entre dos objetos válidos 3D obtiene como resultado un objeto con una cara que cuelga 2D. Luego se efectúan una serie de operaciones matemáticas  $i(A(\cap)B)$  de manera que eliminan los componentes 2D, 1D. La intersección ordinaria se reemplaza por la intersección regularizada  $(A \cap^* B)$  (Torres, 2014) asegurando que siempre se generen objetos sólidos.

### 2.3.2. Visualización

Un elemento indispensable para visualizar modelos 3D en una pantalla es la **Síntesis de imágenes**, definida como un «*campo de la informática gráfica que se dedica principalmente al estudio y desarrollo de procesos para sintetizar imágenes raster*» Ramos (2011). Las computadoras deben realizar un conjunto de operaciones elementales para conseguir pasar de la **representación de un modelo geométrico** tridimensional a su imagen plana en pantalla (2D), pero que para la impresión del observador parezca estar contemplando un sistema de visualización en el mundo real en 3D. Esquemáticamente, el objetivo principal de un sistema gráfico<sup>12</sup> se resume en la figura 2.20: la visualización o **rendering** se produce en función de las condiciones establecidas por el observador, cámara o visor  $f(\text{visor})$ . Es necesario el visor para generar una imagen bidimensional discreta (raster) a partir de un espacio vectorial 3D, lo implica que no se puede mostrar simultáneamente en pantalla toda la información del modelo sino solamente lo que alcanza a ver el observador.

---

<sup>12</sup>Sistema gráfico se refiere al conjunto de programas y herramientas auxiliares que permiten la síntesis de imágenes raster, a partir de un modelo vectorial 3D.

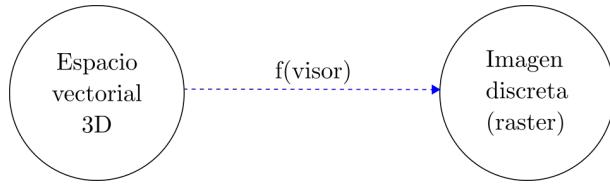


Figura 2.20: Esquema de visualización, se genera una imagen 2D (raster) a partir de un espacio vectorial 3D en función del visor  $f(\text{visor})$  (Ramos, 2011).

La visualización se logra mediante una secuencia de operaciones conocida como *viewing pipeline* (Marsh, 2005) ilustrada en la figura 2.21.

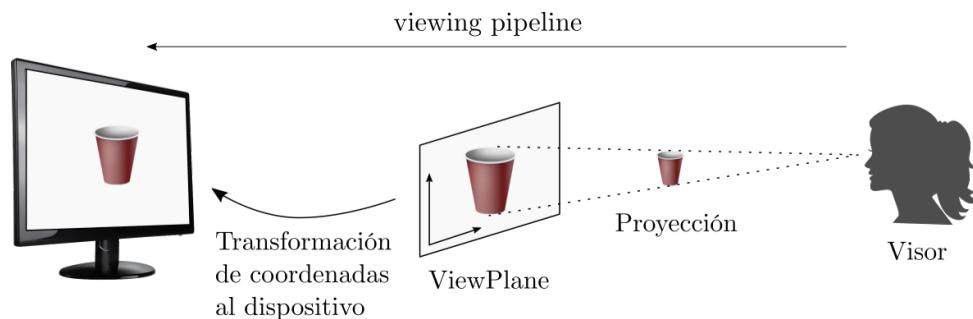


Figura 2.21: Secuencia de operaciones para la visualización (viewing pipeline). En primer lugar, se aplica una proyección que mapea el objeto 3D a un nuevo objeto “plano” en un **plano de visión** en inglés *Viewplain*. Luego, se define un sistema de coordenadas en el viewplane especificando un punto como origen, y dos vectores perpendiculares que dan las direcciones de los ejes de coordenadas. Se aplica una asignación de coordenadas del viewplane para expresar el objeto “plano” en términos del sistema de coordenadas del plano bidimensional elegido. Finalmente, el objeto “plano” se asigna a la pantalla del dispositivo por medio de una transformación de coordenadas bidimensional (Marsh, 2005).

Básicamente hay dos métodos para proyectar objetos tridimensionales sobre una superficie bidimensional: **proyección en perspectiva** y **proyección en paralelo**.

Sea  $n$  el vector plano de un viewplane, y sea  $V$  (punto de visión o cámara) un punto que no está sobre el viewplane. La proyección en perspectiva de  $V$  sobre  $n$  es la transformación que mapea cualquier punto  $P$  del objeto, distinto de  $V$ , en el punto  $P'$  que es la intersección de la línea  $\overline{VP}$  y el plano  $n$ , como se ilustra en figura 2.22 (a). La proyección se efectúa mediante líneas que convergen en una posición determinada. Si  $V$  es un punto en el infinito, la proyección es paralela, todos los puntos del objeto se proyectan sobre la superficie bidimensional mediante líneas paralelas como se ilustra en la derecha de la figura figura 2.22 (b).

La proyección desde el punto de visión  $V$  al viewplane  $n$  es una transformación tridimensional dada por la matriz (Marsh, 2005):

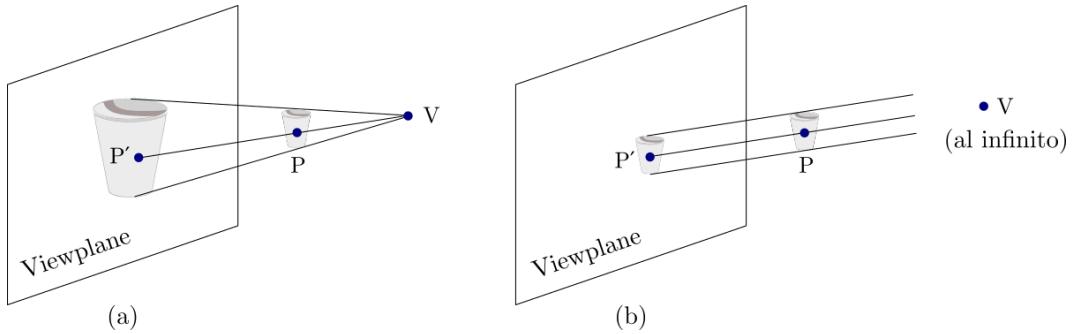


Figura 2.22: a) proyección en perspectiva: La proyección del punto de visión  $V$  es la transformación que mapea cualquier punto  $P$  del objeto observado al punto  $P'$  sobre el *viewplain*. La proyección se efectúa mediante líneas que convergen en una posición determinada. b) proyección en paralelo:  $V$  es un punto de visión en el infinito. Todos los puntos del objeto  $P$  se proyectan mediante líneas paralelas a los respectivos en  $P'$  sobre la superficie bidimensional *Viewplain* (Marsh, 2005).

$$M = n^T V - (nV)I_4 \quad (\text{Ec.4})$$

Siendo  $I_4$  una matriz identidad 4x4.

Las tecnologías de visualización implementan estos conceptos de diferentes maneras dependiendo del contexto en que se utilizan, por ejemplo, en OpenGL<sup>13</sup> se implementa la visualización mediante las matrices Modelo, Vista y Proyección en inglés *model-view-projection matrix* (Kessenich et al., 2016).

En el diseño CAD es fundamental que los modelos estén bien definidos (consistentes) y la visualización es auxiliar (aunque necesaria), por lo que no tiene demasiada importancia la resolución<sup>14</sup> de las imágenes, al menos en las primeras fases de diseño.

### 2.3.3. Diseño de Objetos

Los modelos sólidos se pueden diseñar mediante el **modelado poligonal** (Russo, 2010) que consta en la manipulación de vértices, aristas o caras de los poliedros. Este enfoque es muy flexible porque los ordenadores pueden renderizar los modelos muy rápido, sin embargo, al contener polígonos planos se dificulta la cobertura geométrica al construir modelos de geometría compleja, por ejemplo: para generar superficies

---

<sup>13</sup><https://www.opengl.org/>

<sup>14</sup>La resolución de una imagen indica la cantidad de detalles que puede observarse en esta.

curvas precisas se necesitan grandes cantidades de polígonos. Para solventar ese problema se utilizan otras técnicas de modelado, incluyendo:

### Modelado Booleano (CSG)

Al conjunto de procesos que permiten el diseño de sólidos complejos mediante la combinación booleana de objetos simples (primitivas) se conoce como **modelado booleano**, o también **Geometría Constructiva de Sólidos** en inglés *Constructive Solid Geometry* (CSG) ([Foley et al., 1996](#)).

En el modelado booleano, luego de escalar, trasladar y rotar convenientemente las primitivas, se aplican los operadores booleanos regularizados ( $\cup^*$ ,  $\cap^*$  y  $-^*$ ) para combinarlas. Un sólido construido mediante un modelador booleano (CSG) queda descrito mediante un árbol binario en el que:

- El nodo raíz es el sólido resultante.
- Los nodos internos son los operadores booleanos.
- Los nodos hoja son las primitivas.

En los nodos terminales del árbol no sólo se representan las primitivas, sino que también se indican las transformaciones lineales que se han de efectuar sobre ellas. En el árbol, tanto las operaciones como los nodos hoja deberán encontrarse ordenadas, debido a que las operaciones booleanas no son, en general, conmutativas. El proceso de seguimiento (recorrido) del árbol se hará partiendo de los nodos terminales.

En el ejemplo de la figura [2.23](#) se puede ver la descripción de un modelo booleano, llamado R3.

Partiendo de las primitivas  $P_0$  (cilindro),  $P_1$  (cilindro),  $P_2$  (cilindro),  $P_3$  (esfera) y  $P_4$  (cubo) se obtiene el objeto final ( $R_3$ ) después de tres niveles de operaciones booleanas.

En primer lugar se efectúa la unión de  $P_0$  y  $P_1$ , después de haber transformado y posicionado adecuadamente las primitivas en el espacio, mediante las matrices netas  $T_0$  y  $T_1$ .

$$R_0 = P_0 \cdot T_0 \cup P_1 \cdot T_1 \quad (\text{Ec.5})$$

A continuación el resultado intermedio  $R_0$  se suma con la primitiva  $P_2$  (cilindro), después de haber sido escalada y posicionada por  $T_2$ .

$$R_1 = R_0 \cup P_2 \cdot T_2 \quad (\text{Ec.6})$$

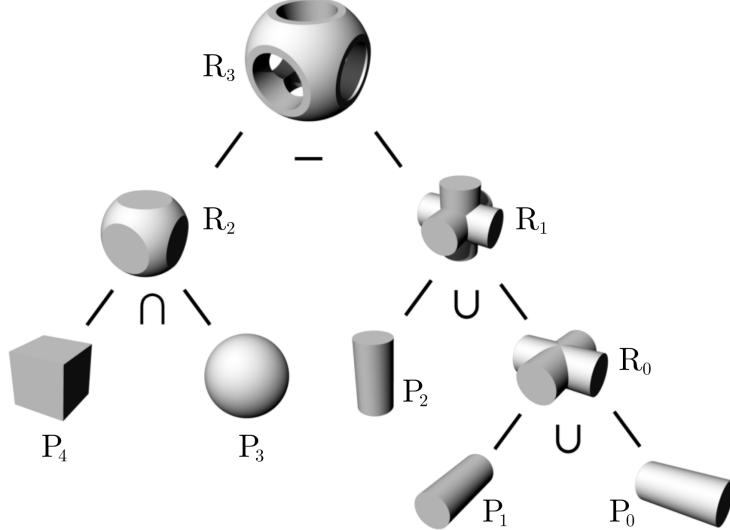


Figura 2.23: Árbol CSG utilizando las operaciones booleanas: unión ( $\cup$ ), intersección ( $\cap$ ) y diferencia ( $-$ ). En la derecha se observa que  $R_0$  es la unión de los cilindros  $P_0$  y  $P_1$  mediante la ecuación ( $R_0 = P_0 \cdot T_0 \cup P_1 \cdot T_1$ ), luego se efectúa la unión entre  $R_0$  y el cilindro  $P_2$  dando como resultado  $R_1$  mediante ( $R_1 = R_0 \cup P_2 \cdot T_2$ ). Por su parte en la izquierda  $R_2$  es la intersección entre la esfera  $P_3$  y el cubo  $P_4$  ( $R_2 = P_3 \cdot T_3 \cap P_4 \cdot T_4$ ). Finalmente el nodo raíz resultante  $R_3$  es la diferencia entre  $R_2$  y  $R_1$  mediante ( $R_3 = R_2 - R_1$ ). También se aprecia un transformación de escala en la figura resultante, aumentando su tamaño, esto se debe a la aplicación de las matrices de transformación  $T$  en cada operación (Wassermann et al., 2016).

Para producir el resultado  $R_2$  se realiza la intersección entre  $P_3$  y  $P_4$ , después de haber transformado y posicionado adecuadamente las primitivas en el espacio, mediante las matrices netas  $T_3$  y  $T_4$

$$R_2 = P_3 \cdot T_3 \cap P_4 \cdot T_4 \quad (\text{Ec.7})$$

Finalmente, a  $R_2$  se le resta  $R_1$ , obteniendo así el objeto raíz del árbol  $R_3$ .

$$R_3 = R_2 - R_1 = (P_3 \cdot T_3 \cap P_4 \cdot T_4) - ((P_0 \cdot T_0 \cup P_1 \cdot T_1) \cup (P_2 \cdot T_2)) \quad (\text{Ec.8})$$

En el ejemplo se puede apreciar que las transformaciones han incrementado el tamaño del sólido resultante  $R_3$ . Por lo general, el escalado de las primitivas no es uniforme, es decir, los factores de escala  $s_x$ ,  $s_y$ , y  $s_z$  son diferentes. De este modo, una sola primitiva puede proporcionar una gran variedad de copias diferentes simplemente usando transformaciones.

Es importante indicar que el modelado booleano es un proceso descriptivo, es decir, sólo se limita a especificar qué primitivas se utilizan y cómo se combinan. De manera

que sólo se dispone de la información geométrica y topológica de las primitivas pero no la de los objetos resultantes. Para poder combinar las primitivas en un esquema de modelado, éstas han de ser definidas previamente. Una forma frecuente de crearlas es estableciendo los parámetros de las ecuaciones que las definen.

### Diseño de nuevas primitivas

La figura 2.24 muestra dos primitivas con los parámetros de control que cada una requiere (excluidos los de posición y orientación). Cuanto más flexible es esta definición, mayor es la capacidad para modelar los diferentes objetos y por consecuencia mayores son las posibilidades de modelado.

Se llama dominio o **poder expresivo** de un modelador, a la capacidad que posee para modelar los diferentes objetos. Cuanto mayor sea el poder expresivo de un modelador, mayores son las posibilidades de modelado.

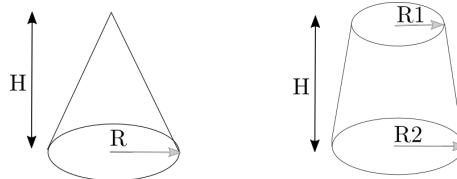
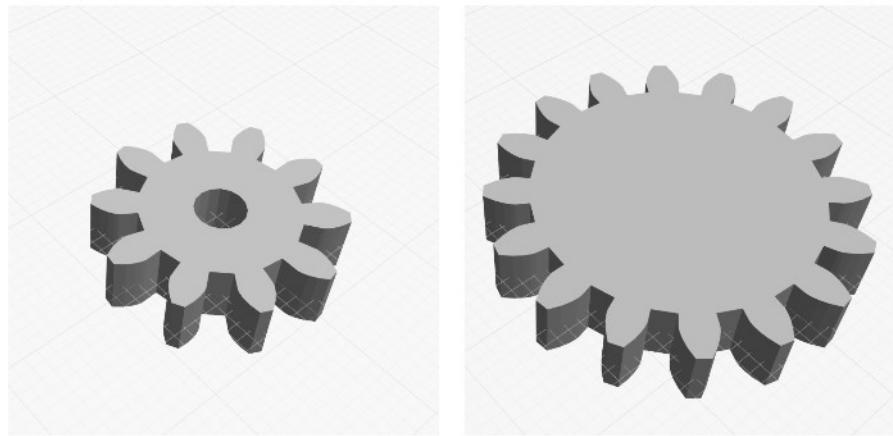


Figura 2.24: Primitivas con sus parámetros de control. A la izquierda un cono con 2 parámetros: altura ( $H$ ) y radio ( $R$ ). A la derecha un cono truncado con 3 parámetros: altura ( $H$ ), radio superior ( $R_1$ ) y radio inferior ( $R_2$ ). La primitiva de la izquierda en 3D solo puede generar conos. La primitiva de la derecha por su parte puede generar cilindros, conos y conos truncados. Por ende, el poder expresivo de esta última es superior. ([Ramos, 2011](#)).

Un sistema de modelado debería permitir al usuario definir sus propias primitivas y así aumentar el dominio y/o la potencia del modelador. En la generación de nuevos ejemplares, el sistema define un conjunto objetos sólidos que son relevantes para el área de aplicación. Estas primitivas suelen parametrizarse no sólo en función de las transformaciones, sino también con base a otras propiedades. Por ejemplo, objetos como engranajes o tuercas son tediosos de definir utilizando combinaciones booleanas de objetos más sencillos pero pueden ser formulados fácilmente con parámetros de alto nivel como diámetro, espesor, número de dientes, etc. (Ver figura 2.25).

La única forma de modelar un nuevo tipo de objeto es describiendo el mecanismo que lo define, es decir, es necesario escribir las rutinas o programas que lo dibujan o determinan sus propiedades.



Radio Total = 7

Num. de Dientes = 10

Radio de orificio = 1

Grosor = 2.5

Radio Total = 14

Num. de Dientes = 16

Radio de orificio = 0

Grosor = 2.5

Figura 2.25: Dos engranajes generados a partir de la misma primitiva. El cambio en los parámetros genera objetos con diferentes características. El engranaje de la derecha no posee ningún orificio, producto del cambio de valor *Radio de orificio* = 0.

## 2.4. Sistemas CAD

Para comprender el estado actual del CAD primero se deben estudiar algunas aplicaciones disponibles. Las soluciones presentadas incluyen herramientas basadas en el escritorio y basadas en la web. A continuación se describen las características más relevantes para este trabajo: Permitir el diseño paramétrico, contar con interfaz de scripting y proveer mecanismos para generar modelos sólidos.

### Blender

**Blender** ([Blender Foundation, 2018](#)) es una suite de creación 3D FLOSS compatible con la totalidad del trabajo en 3D: modelado, rigging, animación, simulación, renderizado, composición y tracking de movimiento, incluso edición de video y creación de videojuegos. Esta aplicación está principalmente orientada a producciones artísticas, sin embargo, dispone de herramientas y primitivas para generar sólidos orientados a la fabricación digital. También permite el uso de modificadores que son operaciones automáticas que afectan a un objeto de una manera no destructiva<sup>15</sup>. Con los modificadores, se pueden realizar tareas que manualmente serían

---

<sup>15</sup> No destructivo se refiere a hecho de realizar cambios sin sobrescribir los datos del modelo original, de manera que se puedan revertir.

demasiado tediosas. Entre muchos otros, se encuentra el modificador booleano que valga la redundancia se utiliza para operaciones booleanas entre objetos. Otra capacidad de blender es el acceso a su API<sup>16</sup> mediante scripting, utilizando el lenguaje de programación python<sup>17</sup>.

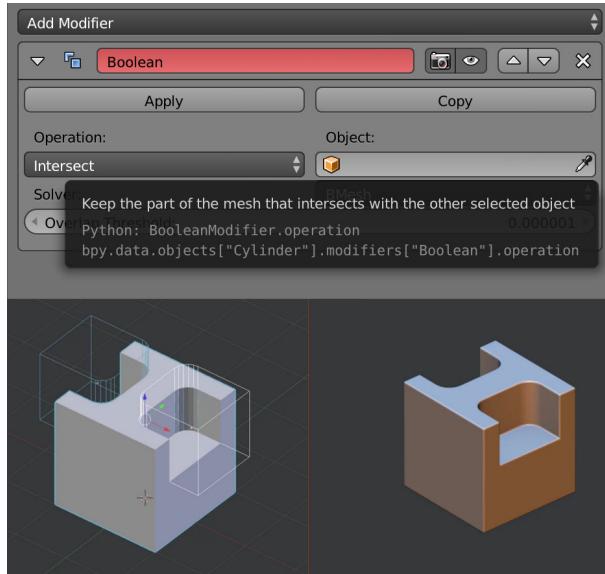


Figura 2.26: Interfaz de Blender. Aplicación de intersecciones mediante el Modificador booleano. Se puede observar el código en python para realizar la operación mediante la API de Blender. (Matthias, 2018).

## Grasshopper

En la última década se ha visto la aparición de un nuevo tipo de interfaz visual para scripting. Este tipo de programación visual implica representar programas no como texto, sino como diagramas. Un ejemplo de estas aplicaciones es **Grasshopper** (Bob McNeel, 2018), un editor de algoritmos gráficos estrechamente integrado con las herramientas de modelado 3D de Rhino<sup>18</sup>. Se basa principalmente en nodos que mapean el flujo de relaciones desde parámetros, a través de funciones definidas por el usuario, concluyendo normalmente con la generación de geometrías. Las modificaciones en los parámetros o las relaciones del modelo hacen que los cambios se propaguen a través de las funciones explícitas para volver a dibujar la geometría automáticamente. Esta es otra forma de crear modelos paramétricos.

<sup>16</sup><https://docs.blender.org/api/current/>

<sup>17</sup><https://www.python.org/>

<sup>18</sup><https://www.rhino3d.com/>

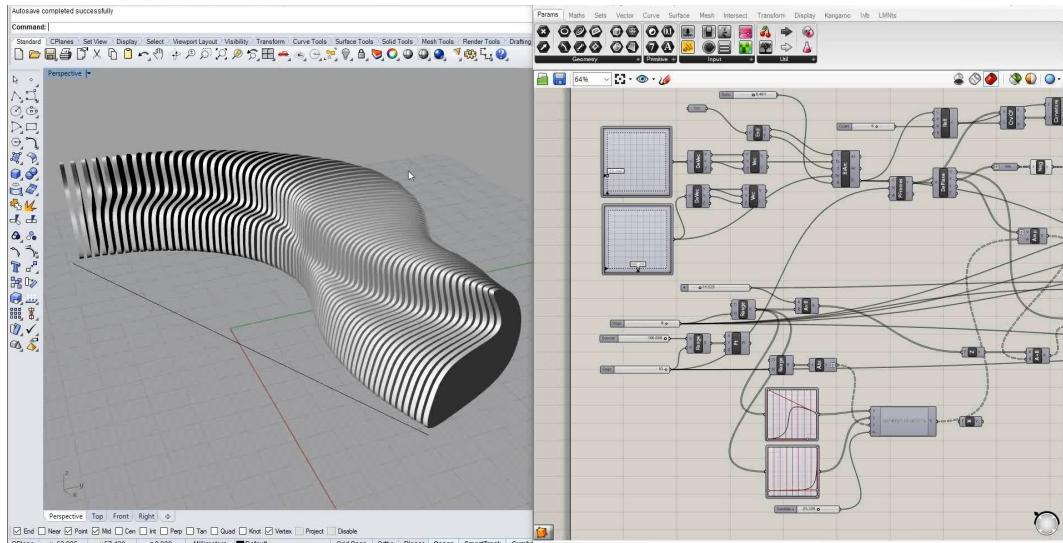


Figura 2.27: Programa en Grasshopper en forma de diagrama y las relaciones entre los nodos (derecha). Modelo resultante en la interfaz de Rhinoceros (derecha). ([Christev, 2016](#)).

## OpenJSCAD

**OpenJSCAD**<sup>19</sup> está inspirado en OpenSCAD<sup>20</sup> y esencialmente proporciona un enfoque a programadores para desarrollar modelos mediante una interfaz web. El diseño paramétrico se realiza mediante scripts en código javascript ([Flanagan, 2007](#)) que utilizan funciones **CSG** y otras un tanto especiales, proporcionadas por el mismo entorno. Permite incorporar parámetros en la interfaz gráfica, posibilitando al usuario cambiar los valores de forma interactiva. OpenJSCAD utiliza otras librerías como base para cumplir con sus funcionalidades. Para modelado booleano requiere de **CSG.js**<sup>21</sup> y para la visualización de los modelos 3D utiliza **lightgl.js**<sup>22</sup> que facilita el desarrollo de aplicaciones WebGL, re-implementando la matriz modelo-vista/proyección<sup>23</sup> (analizada en la sección 2.3.2) de OpenGL<sup>24</sup> para proporcionar una funcionalidad similar.

En la figura 2.28 se puede ver a la izquierda el modelo 3D de un engranaje y los campos con sus parámetros. A la derecha se sitúa el editor de texto con el programa

<sup>19</sup><http://openjscad.org/>

<sup>20</sup><http://openscad.org/>

<sup>21</sup><https://github.com/jscad/csg.js>

<sup>22</sup><https://github.com/evanw/lightgl.js/>

<sup>23</sup><http://www.opengl-tutorial.org/es/beginners-tutorials/tutorial-3-matrices/>

<sup>24</sup><https://www.opengl.org/>

correspondiente escrito en javascript.

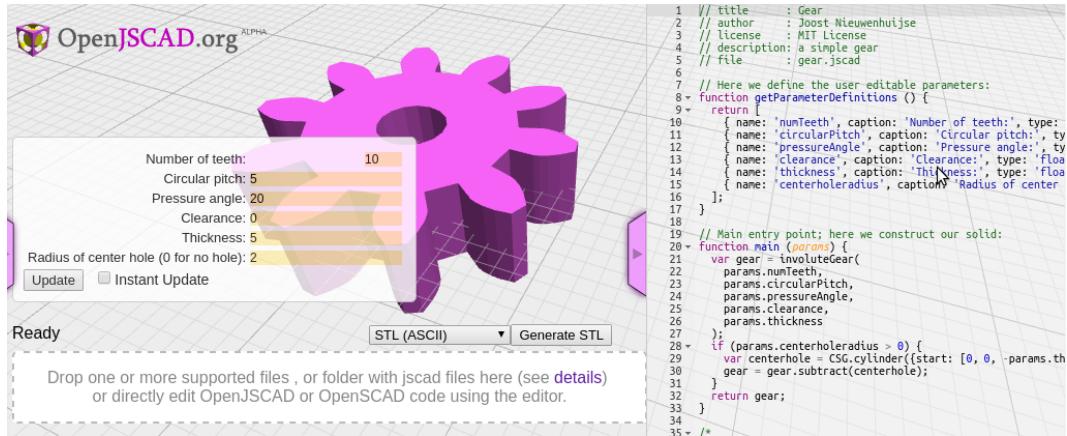


Figura 2.28: Interfaz de OpenJSCAD. Modelo 3D de un engranaje y los campos con sus parámetros (izquierda) y la especificación del modelo mediante código javascript (derecha). ([Nieuwenhuijse et al., 2018](#)).

## OnShape

**OnShape**<sup>25</sup> es una aplicación CAD basada en la nube que puede ser accedida desde cualquier navegador o mediante su aplicación para móviles. Su dominio está fuertemente orientado al **diseño mecánico** ([Lardiés and Cuello, 2012](#)) con características similares a programas como SolidWorks<sup>26</sup>. Cuenta con un sistema de control de versiones<sup>27</sup> para los documentos del proyecto (dibujos, modelos 3d y documentos) y soporta la colaboración en tiempo real. La aproximación de modelado es de manipulación directa, sin embargo, incluye un lenguaje de scripting que se denomina FeatureScript<sup>28</sup> y permite la definición de la geometría, relaciones y parámetros para la construcción de modelos 3D Paramétricos ([Alfaiate, 2017](#)).

<sup>25</sup><https://www.onshape.com/>

<sup>26</sup><https://www.solidworks.com/es>

<sup>27</sup>Se llama control de versiones a la gestión de los diversos cambios que se realizan sobre los elementos de algún producto o una configuración del mismo

<sup>28</sup><https://cad.onshape.com/FsDoc/>

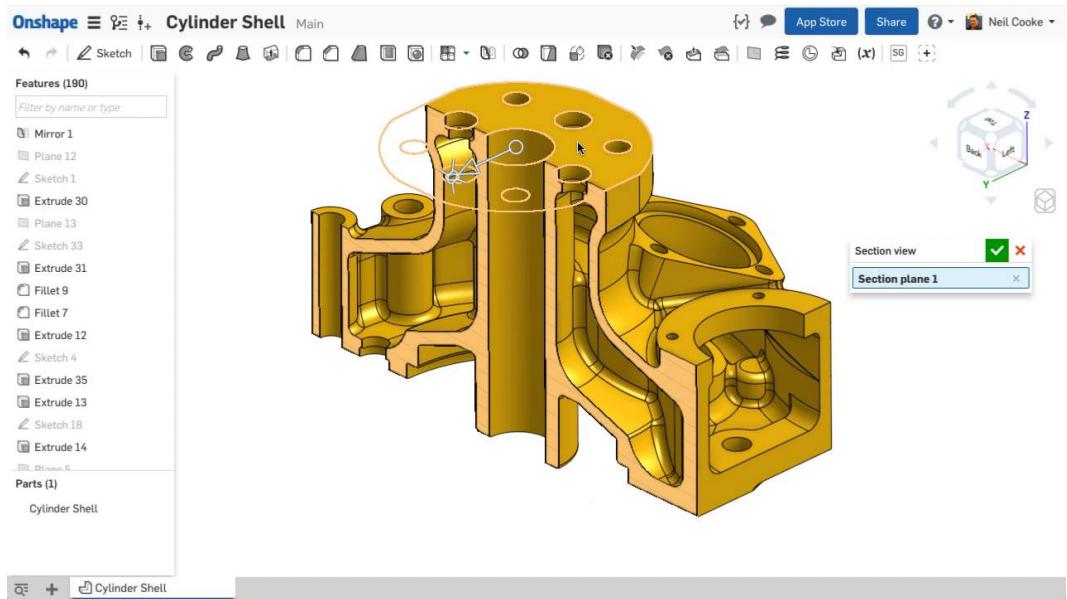


Figura 2.29: Interfaz de OnShape con un modelo mecánico ([Onshape, 2016](#)).

# Capítulo 3

## Desarrollo Colaborativo de Productos CAD con Lean UX

En este capítulo se explican los conceptos que involucran el desarrollo de los sistemas colaborativos y distribuidos para CAD y la metodología *LeanUX* como una herramienta de soporte para desarrollar un software con tales características. Finalmente se describen algunos ejemplos de aplicaciones CAD colaborativas y distribuidas disponibles en la web.

### 3.1. Introducción

La complejidad creciente de los productos hace necesaria la colaboración entre diferentes personas, habitualmente se generalizan en “cliente” y “diseñador” a las partes que interactúan, pero no necesariamente son una única persona por cada parte, sino equipos conformados por múltiples roles. Sus diferentes habilidades pueden abarcar el modelado 3D, la producción física con diferentes tecnologías (impresión 3D, corte a láser, control numérico, etc.), el marketing, etc.

Tradicionalmente, la interacción se producía entregando los planos a “contratistas” para que construyan las distintas partes que integran un proyecto, los contactos se producían de forma presencial y obligaban a frecuentes viajes para mantener el proyecto bajo control ([Ruiz, 2009](#)). Esta modalidad de trabajo se caracteriza por utilizar un **modelo en cascada**, que ordena rigurosamente las etapas del ciclo de vida de un proyecto, de tal forma que el inicio de cada etapa debe esperar a la finalización de la anterior ([de Areba, 2001](#)). Este modelo tiene la ventaja de ser sencillo de comprender e implementar, ya que se sabe de antemano que esperar del proyecto, con una estimación precisa del costo y duración. Sin embargo, una vez que

un producto se encuentra en la etapa de prueba o revisión, es muy difícil realizar algún cambio por algo que no fue definido adecuadamente. De modo que el riesgo y la incertidumbre son elevados en los proyectos donde los requerimientos cambian constantemente. Por otro lado, no se distingue en involucrar al usuario final o cliente durante el proceso, si se solicita algún cambio no previsto, el proyecto se puede retrasar y salirse del presupuesto.

El avance de las tecnologías web ha democratizado la participación en los procesos de diseño en muchos niveles. Los teléfonos inteligentes son un ejemplo de la tecnología al servicio de las personas, porque ofrecen oportunidades para la co-creación mediante sus aplicaciones ([Huerta, 2013](#)). En consecuencia, el rol del contratista tradicional ha cambiado y esto implica que realice su trabajo en estrecha relación con otros actores, en colaboración y sin necesidad del contacto presencial. En este nuevo escenario, la aparición de conceptos como **Desarrollo Colaborativo de Productos** en inglés *Collaborative Product Development* (CPD) ([Elfving, 2007](#)) impulsan la participación de ingenieros, diseñadores industriales, arquitectos, programadores, artistas e incluso personas sin formación específica en procesos conocidos como **co-diseño** o **diseño colaborativo**.

### 3.2. Co-diseño

Se entiende como **co-diseño** al proceso de colaboración creativa entre diseñadores y otras personas, las cuales no poseen una formación previa en diseño, con el propósito de resolver problemas ([Pérez García, 2014](#)). En esta definición se plantean dos elementos fundamentales que dan soporte al paradigma colaborativo:

1. **Nuevos perfiles:** Al grupo habitual de trabajo se suman la iniciativa y la creatividad de otros perfiles que antes aportaban ideas generalmente como agentes externos: el investigador y el usuario final. Estos perfiles tienden a mezclarse: El usuario pasa a jugar un rol de “experto en su experiencia” y puede aportar elementos de valor en la generación de conceptos e ideas en la etapa inicial del proyecto. El investigador, a partir de la experiencia del usuario, puede recoger todos los datos y, a la vez, desempeñar un papel fundamental en formalizar las ideas. Lógicamente, una misma persona puede cumplir más de un rol y el diseñador debe cambiar al incluir nuevos “socios creativos” en un entorno que tradicionalmente le pertenecía.
2. **Objetivo en común:** La idea del **objetivo compartido** se plantea como

una de las principales diferencias con respecto a los métodos tradicionales. Los métodos de diseño generalmente se planteaban para ser llevados a cabo solamente por expertos que realizaban tareas individuales y no era necesario compartir la visión del objetivo general del proceso. En cambio, en un proyecto colaborativo este punto es fundamental: Para que el equipo funcione correctamente necesita tener la visión del objetivo en común ([Huerta, 2013](#)).

En la figura [3.1](#) se pueden apreciar las diferencias entre un esquema de **diseño tradicional vs co-diseño**. En el esquema de la izquierda (tradicional) los roles trabajan por separado, el investigador aporta conocimientos para realizar la lista de requerimientos de diseño, la interacción del diseñador (contratista) con el usuario es a través de dicha lista. De esa manera el diseñador trabaja en base a los requerimientos y una vez finalizado todo el trabajo recién se efectúa la entrega. Por otra parte, en el esquema de la derecha (co-diseño) el usuario participa durante todo el proceso de diseño a través de la interacción directa con todo el equipo, en este ejemplo, el contratista cumple el rol de diseñador e investigador. La interacción se realiza mediante las herramientas inherentes a cada proyecto, que pueden ser diagramas, referencias, artefactos, métodos y/o técnicas. En lugar de establecer los requerimientos, se plantean las necesidades de diseño y las entregas del trabajo se realizan progresivamente mediante la revisión continua del producto por parte de todos los participantes.

La diversidad de profesiones en el co-diseño trae como consecuencia que los actores difieran tanto en la forma de interpretar el diseño como en la forma de comunicarlo. Generalmente la comunicación está cargada de jerga propia cada especialidad y por lo tanto, es difícil de comprender para los demás participantes. Aun así, si se comprendieran las palabras, el significado de las mismas puede diferir según la disciplina. Un ejemplo de palabra con muchos significados es “*concepto*”, la figura [3.2](#) muestra cómo los actores de diferentes especialidades interpretan de diferente manera la misma palabra. En consecuencia, si no se crea un entendimiento compartido sobre el contenido del diseño, se dificulta la colaboración.

## Diseño Iterativo

La etapa de **revisión de los diseños** es fundamental porque permite evaluar la capacidad de los resultados para cumplir los requisitos del cliente, identificar cualquier problema y proponer soluciones ([Pereiro, 2005](#)).

Lo más habitual es registrar los resultados de las revisiones en actas de reunión o

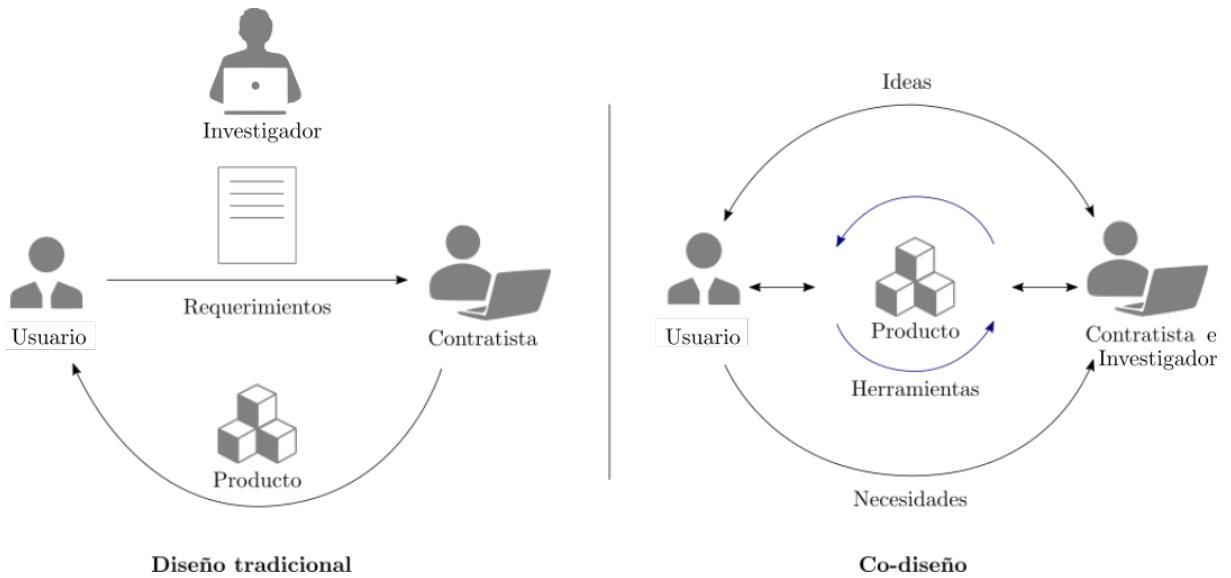


Figura 3.1: Diseño tradicional vs co-diseño. En el diseño tradicional (izquierda) la interacción se realiza mediante la lista de requerimientos hacia el contratista. A la derecha, en el co-diseño se aprecia la interacción de los participantes mediante la compartición de herramientas y la revisión del producto durante todo el proceso ([Sanders and Stappers, 2008](#)).

bien en algún formato que se haya establecido específicamente para el control. Estas tareas parecieran ser anticuadas en términos administrativos, si no se considera que diseñar un objeto tangible implica tener un entendimiento completo de todo lo que se debe hacer antes de iniciar la producción física, porque esta generalmente es compleja comparada con la producción de bienes intangibles. Por lo que la necesidad de la revisión constante y la tendencia creciente a requerimientos que pueden cambiar, ponen en evidencia las limitaciones del modelo en cascada. Como solución surgieron los **modelos iterativos** que promueven una iteración continua sobre el producto y pruebas a lo largo de todo el proyecto ([Laurel and Lunenfeld, 2003](#)).

El **diseño iterativo** en inglés *Iterative Design* es una metodología de diseño basada en un proceso cíclico de creación de prototipos<sup>1</sup> y pruebas, para analizar y refinrar un trabajo que se encuentra en progreso. La interacción con el diseño se utiliza como una herramienta de investigación para recolectar información y desarrollar el proyecto, implementando iteraciones o versiones sucesivas. La prueba, el análisis, el refinamiento y la repetición son necesarios porque la experiencia del usuario no se puede predecir por completo. Las decisiones se basan en la experimentación con el prototipo, de esta manera el proyecto se desarrolla a través de un continuo diálogo

---

<sup>1</sup>Un prototipo es un ejemplar o primer molde en que se fabrica una figura u otra cosa.



a) Concepto para un diseñador de automóviles



b) Concepto para un fabricante de prototipos



c) Concepto para un publicista



d) Concepto para un ingeniero mecánico

Figura 3.2: Interpretación de *concepto* según especialistas de la industria automotriz. a) Para un diseñador son los dibujos rápidos, bosquejos o *sketches*. b) Un fabricante de prototipos lo materializa en modelos de arcilla u otro material que permita visualizar rápidamente los detalles del producto, incluso en escala real. c) Para un publicista es la representación digital o render del producto final. d) Un ingeniero mecánico lo entiende en los documentos técnicos, planos y especificaciones. ([Kleinsmann, 2006](#))

entre los diseñadores, el diseño y el usuario.

En un nivel superficial, el diseño iterativo difiere del modelo de cascada de una sola manera: en lugar de especificar todo el sistema completo antes de desarrollarlo, se diseña y construye completamente una parte del mismo, y luego se utiliza esa parte y las unidades completadas previamente como base para más diseños y más producción futura. En otras palabras, **iterar es diseñar** y, específicamente, es comprender el diseño en el momento que se construye el diseño ([Chronicles, 2009](#)).

En este punto es necesario hacer una aclaración respecto a otro concepto con el que se suele confundir: **el diseño incremental** que tiene como objetivo un crecimiento progresivo de la funcionalidad. Es decir, el producto va evolucionando con cada una de las entregas hasta que se adapta a lo requerido por el cliente. Alistair Cockburn describe al enfoque iterativo como “aprender al completar” y lo distingue del diseño incremental en el sentido que éste consiste en agregar nuevos elementos, incluso de forma iterativa, mientras que iterar trata sobre volver a trabajar y re-

finar (Cockburn, 2002). En ocasiones el concepto de iteración se puede perder al reemplazarlo por incrementos, lo que provoca que el proyecto tenga las mismas consecuencias que los en cascada. Jeff Patton, utilizando visualmente el cuadro de la Mona Lisa (ver figura 3.3) para explicar la diferencia entre los dos métodos (Patton, 2007). Lo más destacado del método iterativo es el siguiente principio subyacente: Hasta que se no se haya construido realmente lo que se está diseñando, no se podrá comprender en su totalidad.

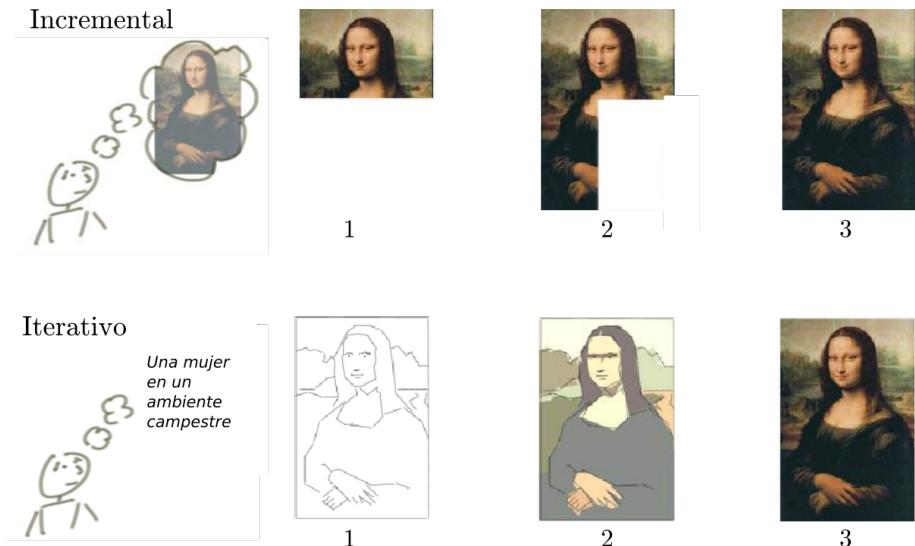


Figura 3.3: Diseño incremental vs iterativo. En el incremental (arriba) se agregan nuevos elementos a partir de una idea completa preconcebida del producto. En el iterativo (abajo) se parte de una idea vaga, ejemplo: “Una mujer en un ambiente campestre”. De forma iterativa se comienza a trabajar sobre una versión mínima del producto, refinando en cada ciclo hasta llegar a la solución (Patton, 2007).

El concepto de iterativo se ha extendido con éxito a otras áreas como las metodologías *Agile* (Cockburn, 2002). Por ejemplo, el marco de desarrollo *SCRUM* (Schwaber, 1995) introduce el concepto de “*sprint*”<sup>2</sup> para referirse a la iteración.

### Riesgo de la metodología en cascada frente a las iterativas

En este contexto, el riesgo se refiere a los factores que contribuyen al éxito o fracaso de un proyecto. Todos los proyectos tienen algún riesgo, independientemente de su enfoque. En algunas situaciones, los factores inesperados que impactan positivamente el proyecto también son riesgos, pero se consideran oportunidades. Una

---

<sup>2</sup>Un Sprint, es un intervalo prefijado durante el cual se crea un incremento de producto “Hecho o Terminado” utilizable, potencialmente entregable

regla que se aplica siempre es que **los riesgos negativos deben detectarse y mitigarse** ([Layton, 2015](#)).

En la metodología en cascada existen una serie de factores negativos, propios de su estructura secuencial. Un caso se puede analizar en la parte superior de la figura 3.4 un modelo en cascada con las etapas *definir, construir, testear y lanzar* para el desarrollo de un producto. Testear (probar) el producto justo antes del lanzamiento significa que si se presentan problemas en esa instancia se pone en riesgo todo el proyecto, la curva de riesgo incrementa notablemente justo al final de proyecto, en el lanzamiento. Por otro lado, en la parte inferior de la 3.4 se aprecia un modelo iterativo (Agile) con las mismas etapas definidas en todas las iteraciones. Si una decisión técnica, un requisito o incluso un producto completo no es factible, el equipo descubre esto en poco tiempo, por ende se cuenta con más tiempo para realizar correcciones. Si la corrección no es posible, de todas maneras el riesgo por el proyecto fallido es menor que con el enfoque en cascada.

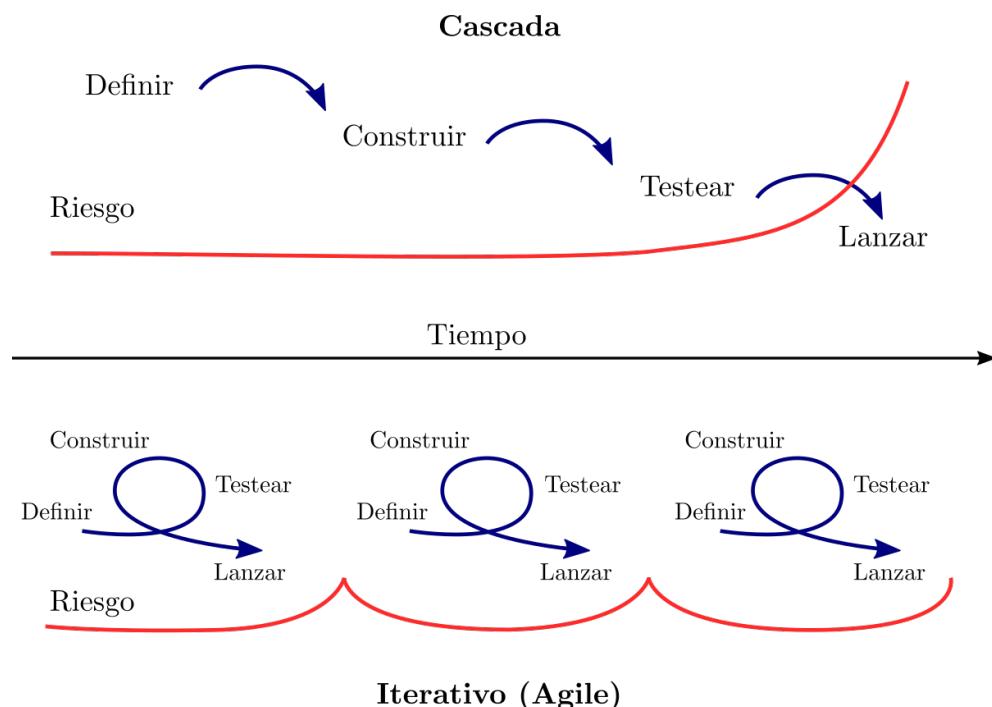


Figura 3.4: Riesgo de un proyecto en cascada vs iterativo (Agile). En cascada (arriba) en un ciclo definir, construir, testear y lanzar el riesgo de fracaso del proyecto se incrementa justo a final. En iterativo (abajo) se definen las etapas definir, construir, testear en cada iteración, por ende, el riesgo es menor y se posibilitan las correcciones durante todo el proceso. ([Kukhnavets, 2016](#)).

Son evidentes las ventajas de la iteración y esto ha promovido que los nuevos productos se lancen a un ritmo creciente. Adicionalmente, tanto el hardware como el software evolucionaron para admitir la comunicación y el acceso multiusuario a los

datos de diseño, la forma de acceso varía desde archivos compartidos hasta accesos compartidos mediante **Sistemas de Gestión de Bases de Datos** en inglés *Data Base Management System* (DBMS) ([Silberschatz et al., 2006](#)).

Las aplicaciones CAD colaborativas se pueden identificar mediante una matriz (ver [3.5](#)) según su forma de uso en el tiempo y espacio ([Maher and Rutherford, 1997](#)). A continuación se analiza las diferentes configuraciones de la matriz.

1. En el **mismo sitio y al mismo tiempo** es posible con una interfaz para un solo usuario, los diseñadores pueden compartir la misma computadora si necesitan colaborar.
2. En el **mismo sitio pero en tiempos diferentes** es posible gracias a la gestión de datos técnicos, de manera que estén disponibles para la misma persona u otros miembros del equipo luego de que el diseño se completa.
3. Al **mismo tiempo en sitios diferentes** se denomina **CAD colaborativo** o bien **Co-Diseño**, diferentes diseñadores pueden ver y modificar el diseño en diferentes ubicaciones, viendo la misma imagen en la pantalla y comunicándose entre sí.
4. En **sitios diferentes en tiempos diferentes** por consecuencia de la distribución de datos del diseño a través de una red (**CAD distribuido**), permitiendo a los actores acceder a los diseños independientemente de su ubicación y disponibilidad.

La colaboración implica que el soporte informático (sistemas CAD) debe proporcionar flexibilidad en la comunicación de datos e ideas, considerando que el diseño colaborativo involucra muchos tipos de conocimiento de diferentes dominios. A continuación se analiza la forma en que los sistemas soportan estos paradigmas.

### 3.3. Sistemas CAD colaborativos

El contenido compartido en los ambientes de **diseño colaborativo y distribuido** por lo general implica el uso de **modelos 3D** como medio de comunicación entre los participantes con el fin de visualizar ideas abstractas y se usan iterativamente durante todo el proceso de diseño ([Tek-Jin Nam, 2009](#)). Los participantes requieren diferentes vistas del diseño, pueden tener intereses diferentes respecto a las soluciones y su representación asociada. De manera que, se necesitan múltiples niveles

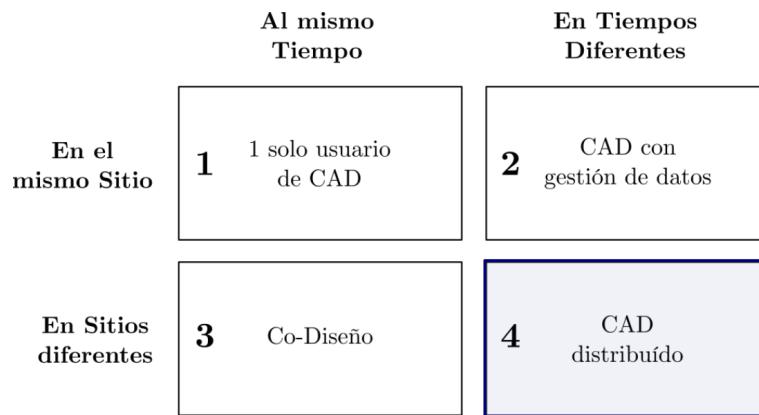


Figura 3.5: El uso del CAD en el espacio y tiempo. ([Maher and Rutherford, 1997](#)) .

de abstracción en términos de soporte informático para gestionar la diversidad de conocimiento.

### 3.3.1. Soporte informático a la colaboración

La colaboración se puede lograr mediante un **Espacio de Trabajo Compartido** en inglés *Shared Workspace*. Este proporciona una comunicación visual y funciona como un medio en el que un actor puede comprender el modelo/diseño de otro sin necesidad de tener el mismo vocabulario ([Maher and Gero, 2006](#)). Un espacio de trabajo compartido CAD se puede dividir en dos significados: El espacio de trabajo con el que los participantes humanos visualizan el diseño e interactúan y la representación compartida del problema de diseño que utiliza la propia computadora para la persistencia y la comunicación entre procesos. Por ende se consideran dos categorías de representaciones en el espacio de trabajo: **Representación visual** compartida y **Representación subyacente** compartida. Esto proviene de los requisitos de los sistemas multiusuario, en el que los usuarios puedan ver el trabajo de los demás. La parte visual es proporcionada por los modelos 3D y además con la posibilidad que el sistema mantenga una o más representaciones de la solución de diseño (versiones). Cualquier otro conocimiento de dominio relevante es proporcionado por el contenido subyacente, por ejemplo: referencias, documentación técnica, archivos extra para interactuar con otros sistemas, etc. Ver figura [3.6](#).

A parte de la implementación de un espacio compartido, resolver problemas como el modelado no es una tarea trivial para los sistemas y se necesita de una infraestructura y componentes con determinadas características. A continuación se explican los sistemas de modelado sólido.

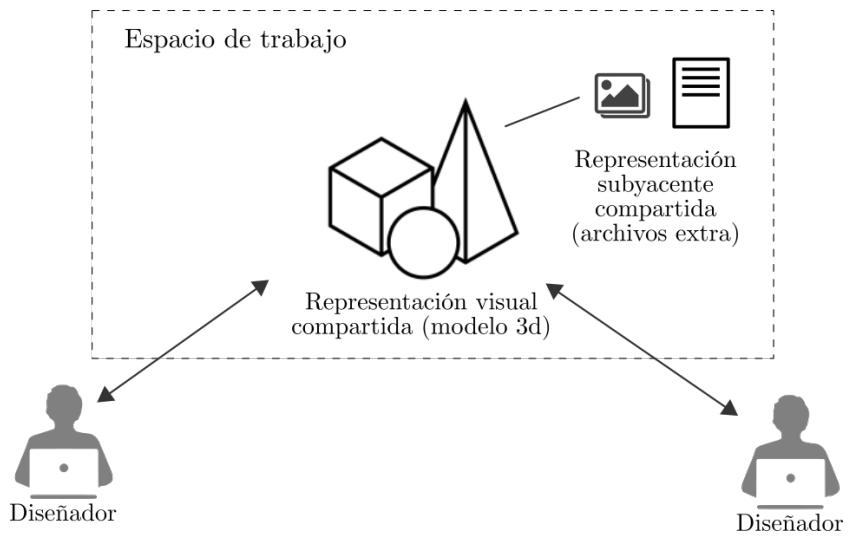


Figura 3.6: Representaciones en un espacio de trabajo compartido por varios usuarios. El modelo 3D y sus posibles versiones son parte de la representación visual. Los archivos extras (referencias, documentación técnica, manuales, etc) se utilizan para la representación subyacente compartida (Maher and Gero, 2006).

### 3.3.2. Sistemas de Modelado Sólido

En la creación de un sistema de modelado de sólidos o “modelador sólido” se deben considerar las condiciones geométricas y topológicas explicadas en la sección 2.3. Mäntylä expone que la idea fundamental es separar el modelo geométrico de la aplicación y desarrollar técnicas de modelado que sean independientes de los objetos (Mantyla, 1988). Los pasos a dar se ilustran en la figura 3.7 y pueden resumir de la siguiente manera:

- Inicialmente, los objetos son descritos por el usuario mediante un **lenguaje de descripción**, basados en los conceptos de modelado (poligonal, booleano, etc) aplicados al sistema. Se introducen utilizando *scripting*, con lenguajes como OpenSCAD o python<sup>3</sup>, o bien mediante una **interfaz de usuario** para interactuar gráficamente. Un mismo sistema puede incluir varios lenguajes de descripción, atendiendo a diferentes usuarios y aplicaciones.
- Una vez introducidos, los objetos son traducidos para crear la **representación interna** almacenada por el modelador. La relación entre el lenguaje de descripción y la representación interna no necesariamente debe de ser directa: las representaciones internas pueden emplear conceptos de modelado distintos a la descripción original. La transformación del lenguaje de descripción a la

---

<sup>3</sup><http://python.org>

representación interna es necesaria para poder encontrar las respuestas a las preguntas geométricas. De hecho, para que un sistema de modelado sea eficiente, debe soportar múltiples representaciones internas de los objetos, por ejemplo con diferentes tecnologías para geometrías como CGCAL<sup>4</sup> u Open-Cascade<sup>5</sup>. Por lo tanto, se deben incluir algoritmos de conversión que puedan modificar una representación en otra.

- Para comunicarse con otros sistemas de modelado (importación/exportación) el modelador debe proveer interfaces de comunicación. Estas interfaces son utilizadas para recibir o transmitir modelos hacia o desde otros sistemas de modelado. Necesariamente debe manejar información geométrica utilizando los diferentes formatos existentes como por ejemplo STEP acrónimo de *Standard for the Exchange of Product Data* ([Wilson, 1998](#)) o STL acrónimo de *Stereolithography* ([Grimm, 2004](#)).
- Finalmente, el modelador también debe incluir facilidades para almacenar las descripción de objetos y demás datos, en bases de datos permanentes.

En estos sistemas se pueden distinguir tres niveles de abstracción:

1. **Interfaz de usuario.** Mediante el lenguaje de descripción, el usuario utiliza las operaciones disponibles en cualquier aplicación CAD (crear, modificar, guardar, borrar y analizar diseños).
2. **Infraestructura matemática y algorítmica.** Implementa las operaciones que proporciona el nivel anterior (por ejemplo, los algoritmos CSG para trabajar con objetos mediante operaciones booleanas).
3. **Primitivas.** A nivel de sistema, son operaciones aritméticas y lógicas que describen los objetos primitivos. Se encuentran disponibles de forma permanente en bases de datos para que puedan ser utilizadas por el nivel anterior.

Dentro de la complejidad de un modelador es indispensable que los datos de los productos sean accesibles y carezcan de errores. Para lograrlo se utilizan los sistemas de **Gestión de Datos del Producto** en inglés *Product Data Management* (PDM) ([Ruiz, 2009](#)) los cuáles se explican a continuación.

---

<sup>4</sup><https://www.cgal.org/>

<sup>5</sup><https://www.opencascade.com/content/latest-release>

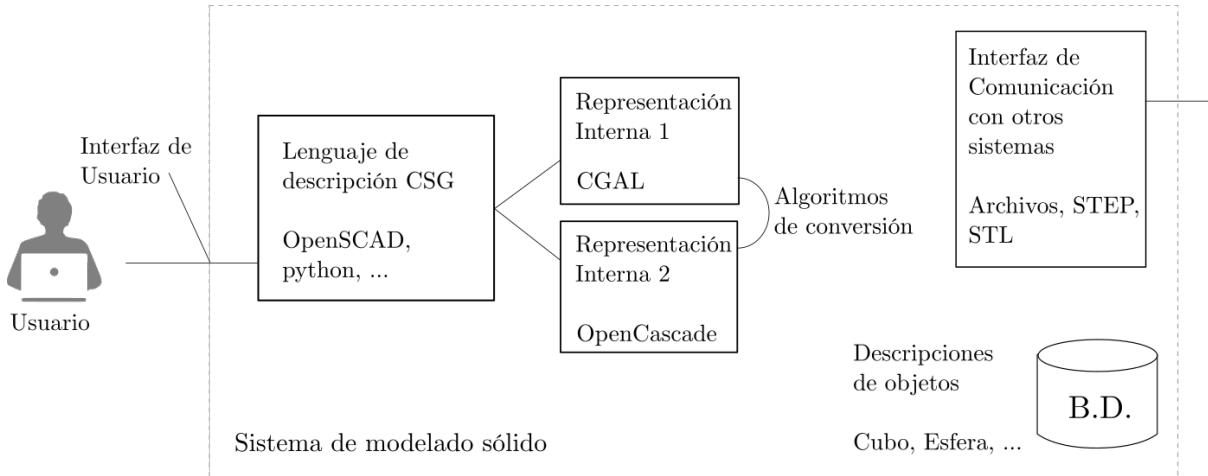


Figura 3.7: Sistema de Modelado Sólido. El usuario (izquierda) ingresa las descripciones de los objetos a través de la interfaz de usuario mediante un lenguaje de *scripting* (OpenSCAD, python, etc). Luego, el lenguaje de descripción es traducido a la representación interna (GCAL, OpenCascade, etc) para encontrar las respuestas a las preguntas geométricas (centro). Si existen varias representaciones internas se utilizan los algoritmos de conversión. En la derecha se puede apreciar la Interfaz de Comunicación con otros sistemas y la Base de Datos permanente con las descripciones de los objetos (primitivas). ([Ramos, 2011](#))

### 3.3.3. Gestión de Datos del Producto (PDM)

Max Ungerer define que «*un sistema PDM es “algo” que maneja datos sobre productos*». En el núcleo central de la información está la identificación del producto y este es representado conceptualmente como un elemento o ítem dentro del sistema ([Ungerer and Buchanan, 2002](#)).

Para garantizar la colaboración distribuida, los PDM tradicionales tienen los mismos problemas que las aplicaciones de escritorio:

1. Es difícil proporcionar acceso a los usuarios desde diferentes ubicaciones, especialmente aquellos que se encuentran en diferentes redes. En cada implementación del PDM, las configuraciones de red deben ser homogéneas.
2. Las aplicaciones cliente<sup>6</sup> son dependientes de la plataforma, todos los usuarios deben usar la misma plataforma informática o bien se debe proporcionar una aplicación específica para cada plataforma de usuario. Actualmente, es casi imposible este mandato por la diversidad de sistemas y dispositivos.
3. Las tareas de ampliación y actualización no son sencillas, cuando se requieren

---

<sup>6</sup>El cliente es una aplicación informática que consume un servicio remoto

nuevas funciones, los usuarios deben volver a instalar o actualizar la aplicación completamente.

En consecuencia, es lógico utilizar un **PDM basado en la web** en inglés *Web based Product Data Management* (WPDM) (Huang et al., 2004) porque pueden ofrecer soluciones centradas en la lógica de negocio de las empresas y proporcionar una comunicación global sin mucho esfuerzo, brindando funciones independientes de las redes y la plataforma. Esto se traduce en una reducción del costo general para la implementación. Un WPDM eficiente requiere:

- Ser totalmente escalable para proporcionar flexibilidad, porque cada organización tiene diferentes prioridades y diferentes flujos de trabajo.
- Ser sencillo de usar por todos los participantes.
- Tener una arquitectura abierta para que se permite añadir, modernizar y cambiar sus componentes sin depender de un proveedor.
- Estar disponible en una amplia variedad de plataformas y proveer funciones en redes heterogéneas.

La mayoría de los sistemas WPDM que se han implementado con éxito hacen uso de **Servicios Web** en inglés *Web Services* (Richardson and Ruby, 2008).

### 3.3.4. Intercambio de datos CAD

En el pasado, el proceso de intercambio de datos estaba relacionado con los planos y los documentos técnicos, hoy se requieren archivos digitales compartidos entre múltiples aplicaciones. Los esfuerzos hacia la integración del CAD generaron el desarrollo de varios formatos estándares para el **Intercambio de Datos de CAD** en inglés *CAD Data Exchange* (Randjelovic and Zivanovic, 2007), así como muchos “no estándares”. Los formatos han variado desde archivos para datos de dibujo técnico como DXF<sup>7</sup> hasta representaciones de modelos 3D como STEP, acrónimo de *Standard for the Exchange of Product Data* (Wilson, 1998).

En lo ámbitos de fabricación digital el CAD data exchange es la norma, a pesar de que cada sistema tiene su propio formatos de datos. Así, la misma información puede ser ingresada varias veces en diferente sistemas, provocando redundancia. Además, el uso de modelos 3D de geometría compleja puede provocar errores en los

---

<sup>7</sup><https://es.wikipedia.org/wiki/DXF>

datos de diseño, con representaciones incorrectas y falta de entendimiento entre los actores. Respecto a esta problemática, desde el año 1999 el Instituto Nacional de Estándares de EE.UU (NIST)<sup>8</sup> estima que la incompatibilidad de datos se traduce en costos que alcanzan los 90 millones de dólares por año ([Tassey et al., 1999](#)).

En conclusión, el CAD Data Exchange es fundamental en el contexto actual del modelado geométrico, ya que es el mecanismo principal para lograr interoperabilidad la entre las diferentes plataformas.

## Data Exchange Geométrico

El enfoque establecido para el intercambio de datos se denomina **Data Exchange de la geometría** o **DE geométrico** ([Spitz and Rappoport, 2004](#)), en este la representación del objeto se transfiere de un sistema origen a un sistema de destino. En las aplicaciones comerciales se implementan a través de los **formatos neutrales** que pueden ser generados y leídos en la mayoría de los sistema CAD. Algunos ejemplos de ellos son STEP y STL, acrónimo de *Stereolithography* ([Grimm, 2004](#)).

STEP interpreta un producto como uno o varios documentos. Utiliza **grupos de conceptos** para organizar los elementos de manera lógica y así generar una estructura clara y comprensible para todas las plataformas. Su estructura depende de la dirección que tome el sistema PDM, sus protocolos de aplicación y su alcance.

Por su parte, los archivos STL describen la geometría de un objeto 3D mediante triángulos, sin considerar la representación de color, textura u otros atributos. Se pueden especificar tanto en ASCII como en binario y se caracterizan por ser ampliamente utilizados para fabricación digital.

El uso del DE geométrico con formatos neutrales o nativos es bastante confiable, aunque muchas veces es posible obtener resultados de modelos no sólidos debido a la perdida de información de sus estructuras. No obstante, su principal inconveniente no es la inconsistencia geométrica, sino el hecho de que no es compatible con el paradigma de diseño más común de la actualidad: el diseño **basado en características** en inglés *Feature Based* (FB), también llamado **diseño paramétrico** o **diseño basado en la historia** como se explica en la sección [2.2](#). Por esa razón es imposible realizar modificaciones de los modelos en el lado del sistema destino. Esta situación se puede ver ilustrada en el ejemplo de la figura [3.8](#).

De esta limitación surge el **Intercambio de Datos Basado en Características** en inglés *Feature Based Data Exchange* (FBDE) ([Spitz and Rappoport, 2004](#)).

---

<sup>8</sup><https://www.nist.gov/>

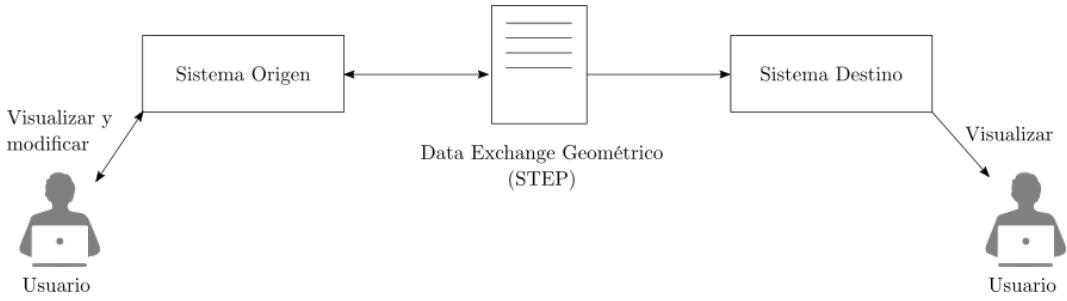


Figura 3.8: Esquema de Data Exchange Geométrico. El usuario en el sistema origen (izquierda) puede visualizar y modificar el modelo para compartir con el sistema destino mediante un archivo en formato STEP que incluye los datos geométricos. Por otra parte el usuario en el sistema destino (derecha) puede visualizar el modelo pero no modificarlo, puesto que solamente tiene acceso a los datos geométricos, pero no a los parámetros necesarios para alterar la representación.

### Data Exchange basado en características (FBDE)

En el FBDE, dado un grafo o estructura basada en el historial paramétrico de un modelo (características) en un sistema origen, el objetivo es construir un grafo en un sistema destino con una geometría similar, conservando al mismo tiempo la mayor información paramétrica posible. En muchos casos, la geometría puede no ser idéntica debido a diferentes políticas de tolerancia entre los sistemas CAD. Siempre que la aproximación esté controlada es totalmente aceptable en la práctica.

El FBDE conserva la inteligencia del diseño, permitiendo que se realicen modificaciones en el lado del sistema destino. Ver figura 3.9.

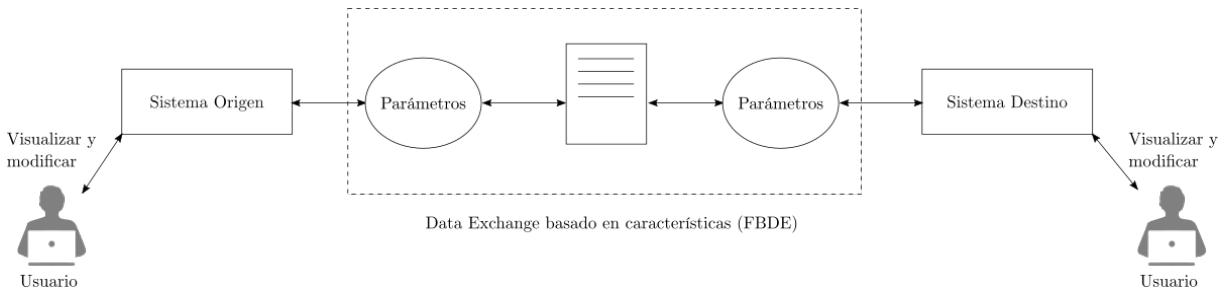


Figura 3.9: Esquema de Data Exchange basado en características (FBDE). El usuario en el sistema origen (izquierda) puede visualizar y modificar el modelo. La compartición del modelo se realiza mediante un formato o tipo de fichero que contiene los datos geométricos y también expone las características del modelo mediante los parámetros. En el sistema destino (derecha) se puede visualizar y manipular el modelo debido a que posee los datos geométricos y paramétricos para modificar la representación.

Adicionalmente, el sistema debe contar con mecanismos para evitar o reparar problemas de inconsistencias geométricas en los archivos de intercambio.

El modelo se suele representar como un árbol o lista de operaciones, comúnmente llamado **árbol de historia**. Las operaciones crean una nueva geometría o modifican una existente. Esta estructura se puede considerar una extensión de la geometría constructiva de sólidos (CSG) explicada en la sección 2.3.3. El punto principal de este paradigma es que las operaciones son siempre de naturaleza paramétrica.

En la figura 3.10 se ilustra un ejemplo con las diferentes versiones de un modelo mecánico. Las versiones se producen a partir de las operaciones o modificaciones de los parámetros, desde un **modelo original** hasta un **modelo objetivo**.

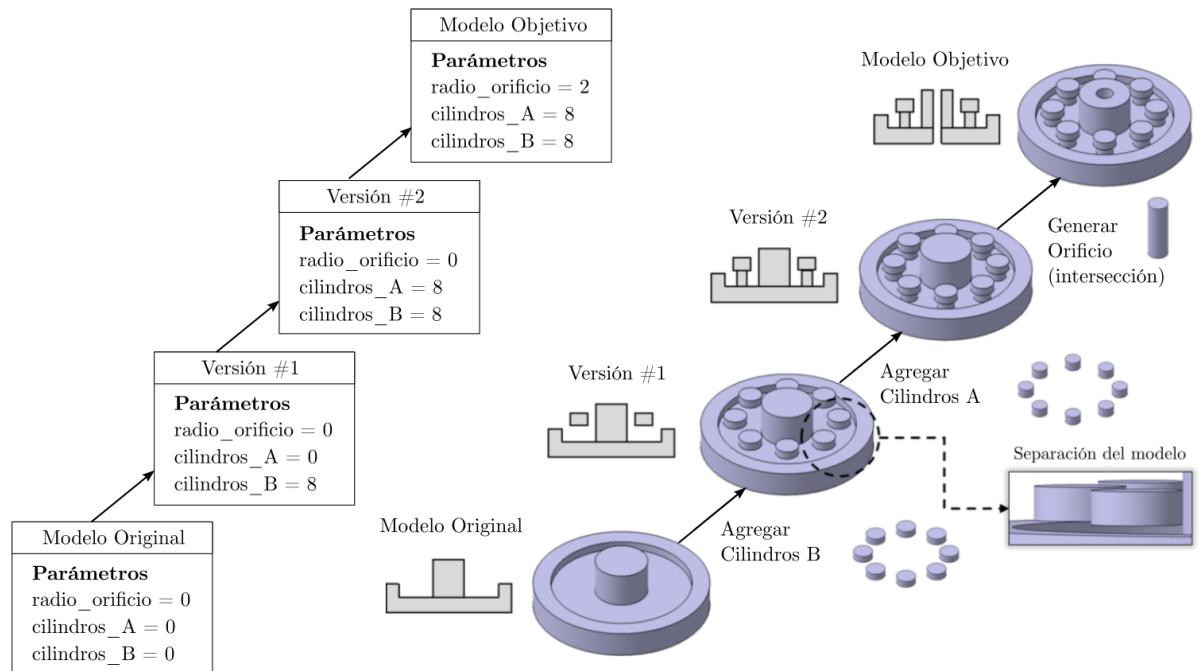


Figura 3.10: Modelo mecánico (derecha) y su representación como árbol de historia (FBDE) (izquierda). Las versiones se producen a partir de las operaciones o modificaciones de los parámetros, desde un **modelo original** hasta un **modelo objetivo**. En cada nivel del árbol se pueden ver las operaciones realizadas. Partiendo del modelo original, para obtener la *Versión #1* se modifica el parámetro *cilindros\_B = 0* a *cilindros\_B = 8* de manera que se agregan 8 cilindros. Esta operación genera una separación entre la pieza y los cilindros, produciendo un objeto “no sólido”. La *Versión #2* soluciona este problema, convirtiendo al objeto nuevamente en sólido con la modificación de la variable *cilindros\_A = 8* y su correspondiente incorporación de 8 cilindros de base. Finalmente a la *Versión #2* se aplica un operador booleano de intersección entre el modelo y un cilindro, generando un orificio en el modelo objetivo ([Kwon et al., 2015](#)).

Por lo anteriormente expuesto, es deseable un sistema de **CAD distribuído** que permita el **co-diseño** de productos orientados a la **fabricación digital**, este debe tener capacidades para la gestión de datos de productos basados en la web

**(WPDM)** y soporte al intercambio de datos basados en características **(FBDE)**.

Para que el desarrollo del sistema esté en sintonía con los principios colaborativos planteados, se pueden utilizar metodologías o enfoques orientados a la experiencia de usuario (UX) como *Lean UX*.

### 3.4. Lean UX

Los equipos de desarrollo de software son un ejemplo de eficiencia en la colaboración, utilizan técnicas de desarrollo ágil (Cockburn, 2002), reduciendo drásticamente el tiempo que requiere modificar una aplicación, realizan cambios en el código y lo llevan a producción<sup>9</sup> a una velocidad similar a la de guardar un archivo en un ordenador. Además, utilizan mecanismos para iterar incorporando lo que han aprendido y tal vez, sin advertirlo, aumentan las expectativas de los usuarios. GitHub<sup>10</sup> es un caso de éxito como plataforma de trabajo colaborativo, convirtiéndose en la herramienta de gestión de código más utilizada en la actualidad (GitHub, 2018). En este nuevo contexto, las prácticas de analizar todo el proyecto al inicio quedaron obsoletas.

*Lean User Experience* (Lean UX) (Gothelf and Seiden, 2013) es una metodología que utiliza las herramientas disponibles y las combina de forma diferente para adecuarlas a esta nueva realidad.

Jeff Gothelf y Josh Seiden la describen como «una nueva etapa evolutiva en el diseño de productos». Se considera profundamente colaborativa y multidisciplinaria, en gran medida porque permite implementar técnicas para construir una comprensión compartida del proyecto, logra un ambiente propicio para el feedback con los usuarios finales, replantea las conversaciones de diseño en términos objetivos y cambia la forma en que se comunica el diseño del producto: en lugar de hablar de funciones y documentos, se habla de lo que efectivamente funciona y de lo que no.

Los 3 pilares principales de Lean UX son:

#### 1. Design Thinking o Pensamiento de Diseño

El *Design Thinking* (Brown, 2009) logra obtener soluciones involucrando a los usuarios para convertirlos en actores activos en todo el proceso de la construcción del producto. Es una manera de trabajar que alienta la colaboración del equipo, independientemente del rol que cada actor desempeñe. Centra su eficacia en entender y dar solución a las necesidades reales de los usuarios.

#### 2. Metodologías de desarrollo ágil de software.

Los desarrolladores de software las han usado durante mucho tiempo. No obs-

---

<sup>9</sup>Producción es la instancia del software cuando se encuentra a disposición de los usuarios finales.

<sup>10</sup><https://github.com/>

tante, estas constituyen un reto para los diseñadores. Lean UX aplica los 4 principios básicos del desarrollo ágil al diseño de productos ([Gothelf and Seiden, 2013](#)) :

- a) **Los individuos y las interacciones son más importantes que los procesos y las herramientas:** Para generar rápidamente las mejores soluciones, se debe implicar a todo el equipo. La comunicación fluida debe primar por encima de las restricciones propias de las herramientas.
- b) **El software funcional es más importante que la documentación exhaustiva:** Un software que funcione es más importante que preocuparse por una documentación exhaustiva. De esta manera, se puede encontrar de antemano la solución que mejor se adapte a las necesidades.
- c) **La colaboración con los clientes es más importante que la negociación de contratos con ellos:** Si el equipo colabora con los usuarios/-clientes, hay un entendimiento común sobre los problemas y las posibles soluciones. Cualquier decisión se debe tomar por consenso, esto se traduce en iteraciones más rápidas y una verdadera implicación de todos los actores con la ventaja de trabajar siempre con soluciones validadas.
- d) **La respuesta a los cambios es más importante que la planificación:** Se asume que el equipo no encontrará la solución la primera vez, por lo que el objetivo consiste en averiguar en qué se ha fallado. Luego se pueden ajustar las propuestas y volver a probarlas iterativamente.

### 3. Método Lean Startup

Los proyectos han estado enmarcados, tradicionalmente, por los requerimientos y las entregas. A los equipos se les suministraban requerimientos para que produjeran entregas. No se inicia a partir de requerimientos, sino de suposiciones. A partir de ellas, se crean y prueban **hipótesis** o *una manera de expresar las suposiciones que se tienen del proyecto de una forma comprobable* ([Gothelf and Seiden, 2013](#)).

*Lean Startup* ([Ries and Julián, 2012](#)) utiliza un ciclo de feedback denominado “**construir-medir-aprender**” que minimiza el riesgo de los proyectos y consigue que el equipo pueda desarrollar software y aprenda de él en muy poco tiempo (ver figura [3.11](#)).

De esta manera, los equipos pueden desarrollar de inmediato los denominados

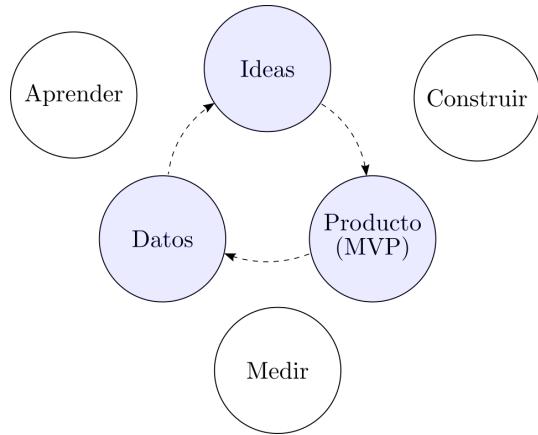


Figura 3.11: Ciclo Construir-Medir-Aprender. Consiste en poner en marcha (construir) las ideas que se suponen van a tener éxito, medir y recabar los datos para confirmar o desmentir las hipótesis que se plantearon al principio y aprender del fracaso o el éxito resultante. En el centro de la figura se aprecia la interacción entre las ideas, el producto construido (MVP) y los datos recabados. ([Borodin, 2018](#)).

**Productos Mínimos Viables** en inglés *Minimum Viable Products* (MVP) ([Olsen, 2015](#)). El ciclo se desarrolla de la siguiente manera:

- En primer lugar, se construye un MVP, es decir, **el desarrollo más pequeño que pueda construirse para probar cada hipótesis**. Se debe tener en cuenta que tanto el MPV inicial como las siguientes iteraciones deben tener valor por sí mismas. Es decir, debes poder obtener un feedback del usuario final. Para esto, el producto debe tener el mismo tratamiento utilizado en el diseño iterativo. Para un mejor entendimiento de este punto, se puede analizar la figura 3.3 explicada en la sección 3.2.
- El siguiente paso es entregar el MVP a los usuarios y realizar experimentos que puedan ser medidos para obtener datos.
- Después, se utiliza lo aprendido con ellos para evaluar las hipótesis y hacer las modificaciones pertinentes.
- Y finalmente se itera, se comienza todo el proceso nuevamente.

## Proceso Lean UX

El objetivo final es trasladar todos los valores heredados de los 3 pilares al trabajo de experiencia de usuario mediante un conjunto de pasos definidos, métodos y herramientas que se analizan a continuación.

### Declaración de suposiciones

Se compone de los siguientes elementos:

- **Suposiciones:** una declaración de alto nivel que se considera cierta. El primer paso en Lean UX es declarar las suposiciones (ver figura 3.4), este ejercicio se realiza en equipo, asegurándose de que todas las disciplinas estén representadas.

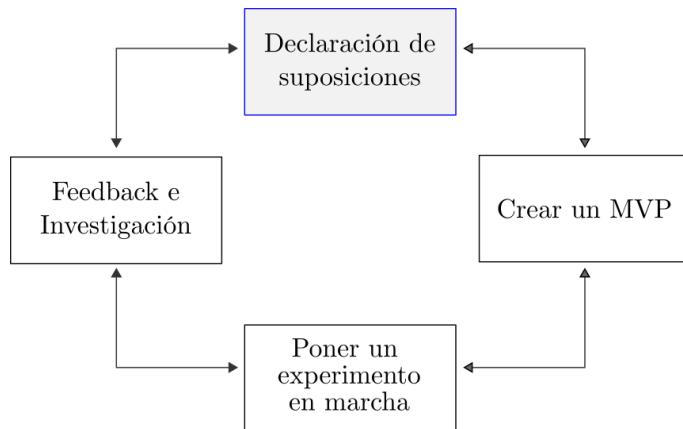


Figura 3.12: Proceso LEAN UX.

Antes de cualquier declaración, se realiza una exploración recurriendo a técnicas de investigación en inglés *research* ([Sharon and Gadbaw, 2016](#)) sobre los usuarios y sus necesidades. Con toda la información recolectada se continúa con la declaración del problema.

### Método: Declaración de problema

Permite centrar correctamente el trabajo de todo el equipo y define las restricciones y límites necesarios para que no se pierda de vista el objetivo. Se puede utilizar la siguiente plantilla:

«[El servicio/producto] debe cumplir con [estos objetivos]. Sin embargo, se ha observado que no se están alcanzando [estos objetivos], lo que está causando

[este efecto adverso] para los usuarios. ¿Cómo se podría mejorar el [servicio/- producto] de modo que los usuarios consigan mejorar sus resultado según [estos criterios cuantificables o cualificables]?»

Generalmente las declaraciones está repletas de suposiciones, para extraerlas se utiliza una lista u **hoja de suposiciones**.

En base a las siguientes preguntas, se recopilan las declaraciones que reflejen lo que el equipo de desarrollo considere cierto respecto a la solución o producto. **¿Quiénes son los usuarios del producto?. ¿Cómo encaja en su trabajo?. ¿Qué problemas soluciona?. ¿Cuáles son sus funciones más importantes? ¿Qué aspecto debe tener y cómo debe comportarse?**

Si existen diferencias en un punto, lo mejor es reflejar esto de inmediato para facilitar la discusión entre los miembros del equipo de desarrollo. Una vez obtenida la lista de suposiciones priorizada, es necesario probarlas.

- **Hipótesis:** descripciones más detalladas de las suposiciones, dirigidas a áreas específicas del producto o flujos de trabajo con las que se puede experimentar. Es decir, se transforman las suposiciones a un formato más sencillo de probar. Sin embargo, las hipótesis suelen ser demasiado extensas para que, con una única prueba, se pueda determinarse su validez. Contiene demasiadas partes, es decir, demasiadas “sub-hipótesis”. Para registrar estas partes más pequeñas y específicas se utiliza la siguiente plantilla:

«Se considera que [haciendo esto, desarrollando esta función, creando esta experiencia de usuario] para [estas personas] se conseguirá [este resultado]. Se sabrá si esto es correcto cuando se obtenga [esta medida cuantitativa, o este conocimiento cualitativo]».

El primer campo se completa con la función o mejora para el producto. El segundo describe exactamente qué usuarios objetivo se beneficiarán de la función. El último, especifica los beneficios que esos usuarios obtendrán de ella. La frase final lo une todo. Esta frase determina si la hipótesis es cierta o no.

- **Resultados:** los datos de entrada, proveniente de la experiencia de los usuarios, que ayudan a validar o invalidar las hipótesis. Pueden ser cuantitativos o cualitativos. En principio se encuentran con resultados de alto nivel o generales. Se debe considerar cómo dividir esos resultados en componentes más pequeños, específicos y cercanos a la realidad del proyecto. **¿Qué funciones en la UI podrían generar un mayor uso?. ¿Un visor 3D totalmente funcional o**

simplemente compartiendo imágenes?. Mediante estas preguntas, se enuncian resultados que se acercan a las necesidades reales del proyecto. Ejemplo: **Dar soporte a la visualización de modelos 3D en la web.**

- **Personas:** son modelos o arquetipos ([Gothelf and Seiden, 2013](#)) de los potenciales usuarios para las que se considera estar resolviendo el problema. Detallan quién utilizará el producto y por qué lo hará. Se inicia con esquema muy sencillo, en cuya creación participa todo el equipo. A medida que se avanza en la investigación se pueden realizar ajustes.
- **Funciones o funcionalidades:** son las técnicas, las funciones, los productos y los servicios a desarrollar para conseguir los resultados. Normalmente, en este punto todos los miembros del equipo tienen su propia opinión, ya que, después de todo, las funciones son lo más concreto con lo que trabajan y les resulta más sencillo expresar sus ideas en términos de funciones. Sin embargo, un error común es hacer que el proceso de diseño parta de las funciones.

Con todo el material obtenido se organizan las sub-hipótesis que deben probarse en una **tabla de sub-hipótesis**. La tabla se organiza con el siguiente formato: Función / Persona / Resultado.

## Diseño Colaborativo

Con la **tabla de sub-hipótesis** como guía se reúne a diseñadores y no diseñadores (programadores, administradores de proyectos) para crear los conceptos del producto , y un entendimiento común sobre el problema y las soluciones de diseño. Asimismo, permite decidir qué elementos de la interfaz gráfica implementan mejor las funciones recogidas en las hipótesis. A continuación se describen dos herramientas utilizadas para el diseño de interfaces.

- **Estudio de diseño:** «*Un equipo interdisciplinario se reúne en una sesión de trabajo o Estudio de Diseño (a veces también llamado Charrette de diseño)*»([Gothelf and Seiden, 2013](#)). Permite explorar y analizar qué elementos de la interfaz gráfica pueden implementar mejor las funciones recogidas en las hipótesis. La documentación de salida de las sesiones constan normalmente de **esquemas de baja fidelidad** en inglés *sketch* (ver figura 4.2). Es esencial que la documentación no sea muy elaborada para que el trabajo continúe siendo maleable. De esta manera, el equipo puede cambiar de dirección con facilidad si las pruebas demuestran que el enfoque adoptado no es el correcto.

- **Guía de estilo:** En base a la documentación resultante del estudio de diseño se desarrolla la **guía de estilo**, una biblioteca de patrones aceptada por todo el equipo, para codificar los elementos gráficos e interactivos de la interfaz de usuario. También sirve para recopilar los potenciales componentes web en inglés *web components*<sup>11</sup> del producto, todo lo que forme parte de la experiencia de usuario (UX) aparece en la guía de estilo. No solo se define el aspecto y el comportamiento del producto, sino que también se proporciona la codificación de los componentes y las hojas de estilo en inglés *Cascading Style Sheet* (CSS)<sup>12</sup>. De esta manera, al modificar la guía de estilo también se modifica el producto.

## PMV y experimentos

Con la lista de sub-hipótesis se crean los PMV, explicado en la sección 3.4. Se utilizan para hacer experimentos y determinar qué ideas sobre el producto son válidas y cuáles deben descartarse. Una de las maneras más efectivas de crear los PMV es mediante un prototipo de la experiencia. «*Un prototipo es una aproximación de la experiencia de usuario que permite simular cómo será el uso de un producto o servicio*» (Gothelf and Seiden, 2013). Es recomendable probar el PMV en primera instancia con todo el equipo de desarrollo, y luego con otras personas. Cuanto más se exponga a las miradas ajenas, más conocimiento se tendrá para validarlos. El siguiente paso es la experimentación de los usuarios finales. La idea es dejar que lo utilicen libremente, así se puede obtener todo el *feedback* posible sobre la experiencia.

## Feedback

Hasta ahora, todo el trabajo está basado en las suposiciones; a partir de este punto se comienza con el proceso de validación. Un error común es realizar este proceso pocas veces, normalmente al principio del proyecto o al final. La investigación es continua, lo que significa que se realizan en cada *sprint* o iteración del producto.

Como consecuencia de usar este enfoque se obtiene: un equipo que trabaja de forma colaborativa, iterativamente, reduciendo al mínimo los documentos entregables, enfocándose en el software funcional y en el *feedback* con el usuario final.

---

<sup>11</sup><https://www.webcomponents.org/introduction>

<sup>12</sup><https://www.w3.org/Style/CSS/>

### 3.5. Antecedentes

#### Spekle

Spekle ([The Bartlett, 2015](#)) comenzó como una investigación de diseño colaborativo para arquitectura en *The Bartlett School of Architecture* (UCL)<sup>13</sup>. El proyecto permite que los usuarios puedan compartir diseños paramétricos en la web. En la figura 3.13 se puede ver un modelo paramétrico en la plataforma web Spekle. El proyecto consta de:

- Un plugin denominado *Spekle Streams*<sup>14</sup> para el ecosistema del software Rhino<sup>15</sup>, este permite establecer los parámetros de los modelos y exportar a archivos en un formato particular SPK.
- Una plataforma web que permite subir los archivos SPK para visualizar modelos 3D paramétricos, generar versiones modificadas, registrar un historial de los cambios y colaborar con otros usuarios.

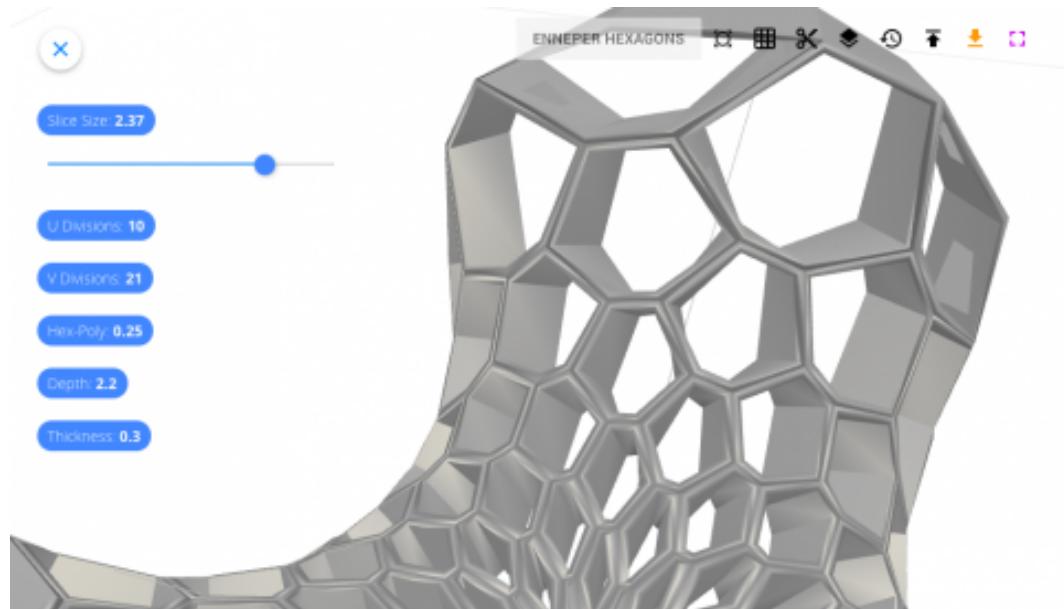


Figura 3.13: Interfaz web de [spekle.works](#), a la izquierda se pueden apreciar los parámetros del modelo. ([Dimitrie, 2017](#)).

Lo destacado de esta plataforma es la manera intuitiva de visualizar y modificar los modelos mediante su interfaz con componentes visuales como campos de texto,

<sup>13</sup><https://www.ucl.ac.uk/bartlett/architecture/>

<sup>14</sup><https://www.food4rhino.com/app/spekle-streams>

<sup>15</sup><https://www.rhino3d.com/>

deslizadores, etc. **Speckle Works**<sup>16</sup> es el repositorio FLOSS del proyecto que permite contribuciones de la comunidad. Actualmente se desarrollan diversas herramientas para la integración con otros softwares como Blender.

## Modelo.io

Es una plataforma web creada por Qi Su y Tian Deng ([Modelo.io, 2018](#)) orientada a profesionales que utilizan Rhino y SketchUp<sup>17</sup>. La característica de esta herramienta es que permite una colaboración efectiva entre los miembros de un equipo para perfeccionar los diseños y presentarlos de forma interactiva.

Los usuarios pueden explorar los modelos 3D, realizar comentarios, capturas de pantalla, anotaciones sobre los modelos, almacenar archivos extras asociados a los proyectos y otras características avanzadas como crear presentaciones 3D interactivas para los clientes. Se utiliza principalmente en proyectos de arquitectura y diseño de interiores (ver figura 3.14).

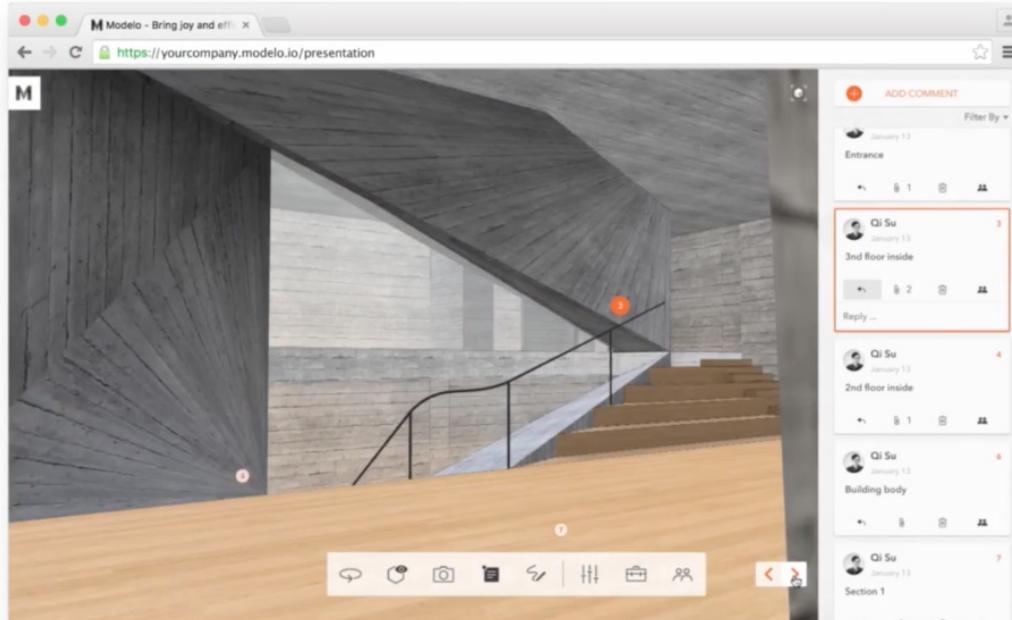


Figura 3.14: Interfaz web de Modelo.io, a la derecha se puede apreciar la interacción entre los usuarios ([Anthony Frausto-Robledo AIA, 2016](#)) en un proyecto de arquitectura.

---

<sup>16</sup><https://github.com/speckleworks>

<sup>17</sup><https://www.sketchup.com/es>

## **Problemas encontrados**

En este trabajo se han presentado herramientas que soportan características relacionadas con el modelado especificado en algoritmos y la colaboración mediante modelos 3D. Los entornos analizados que permiten el modelado en la web son: OpenJSCAD y OnShape. Por otro lado, los entornos colaborativos son OnShape, Spekle.works y Modelo.io. En cierta medida, OnShape es la herramienta que cumple con los requisitos deseables para un sistema CAD distribuido vistos en la sección [3.3](#), sin embargo, esta herramienta presenta el problema de no ser intuitiva para los usuarios sin conocimientos en modelado mecánico. Dado que uno de los objetivos de este trabajo es lograr la colaboración entre personas con diferentes competencias, se debe abordar la solución teniendo presente esta necesidad.

# Capítulo 4

## Resultados: CoCADA Un software para el diseño colaborativo con LeanUX

En este capítulo se implementa el proceso de *Lean UX* en el desarrollo de CoCADA. El mismo se divide en cinco secciones, las cuatro primeras ([4.1](#), [4.2](#), [4.3](#) y [4.4](#)) corresponden al uso de la metodología la cual describe las declaraciones, el diseño colaborativo, el desarrollo de los PMVs y experimentos y el *feedback* con los usuarios. La última sección ([4.5](#)) se enfoca exclusivamente a los aspectos técnicos del sistema tales como la arquitectura de software y las tecnologías involucradas.

### 4.1. Declaraciones

#### Declaración del problema para CoCADA:

«Los proyectos de diseños de productos son desarrollados por equipos de trabajo generalmente conformados por personas con diferentes competencias (profesiones o grados de conocimiento). Los participantes requieren comunicar sus ideas, proponer cambios y comprender la evolución de los productos, para ello es fundamental una comunicación eficiente. Se ha observado que las herramientas de comunicación más utilizadas en la actualidad (email, redes sociales, etc.) **dificultan la organización de los mensajes y la gestión de información sobre los proyectos**. Esto genera problemas de interpretación de las características de los productos y dificulta el control sobre el trabajo realizado. En consecuencia, los participantes manifiestan una experiencia negativa durante el proceso, reduciendo su participación.

»¿Cómo se podría mejorar la comunicación de modo que los que usuarios aumenten la colaboración de una manera ordenada y precisa?»

Las **suposiciones** del problema son entonces las siguientes:

**1. ¿Quiénes son los usuarios del producto?**

- a) **Personas sin conocimientos específicos.** Interesados en participar del proceso de diseño sin tener conocimientos sobre diseño 3D. Generalmente solicitan modelos 3D para la visualización o para la fabricación digital. Estas personas pueden ser emprendedores, artistas, docentes, etc.
- b) **Profesionales encargados de crear diseños 3D.** Poseen las capacidades técnicas y la experiencia sobre procesos y metodologías para llevar a cabo proyectos de diseño de productos.

**2. ¿Cómo cambiaría el producto su trabajo?**

- a) Ahorraría mucho tiempo en el *feedback*.
- b) De forma positiva, ya que es indispensable la organización en los cambios de los diseños.

**3. ¿Qué problemas soluciona el producto?**

- a) La imposibilidad de visualizar el estado actual de un diseño.
- b) La comunicación inexacta a la hora de discutir sobre un producto.

**4. ¿Cuáles serían las funciones más importantes?**

- a) Visualizar el modelo 3D de un producto.
- b) Comunicarse con el Diseñador de forma similar a un chat.

**5. ¿Qué aspectos debe tener el producto y cómo debe comportarse?**

- a) Debe ser agradable a la vista, intuitivo y fácil de usar como una red social.

#### **4.1.1. Hipótesis**

Desarrollando una herramienta de comunicación para proyectos de diseño de productos, se logrará una mayor colaboración en los equipos de trabajo. Se sabrá si el desarrollo es correcto cuando aumente la participación entre las personas con diferentes competencias y se generen las condiciones necesarias para el co-diseño.

#### 4.1.2. Personas

En la figura 4.1 se analizan dos arquetipos que representan los dos tipos de usuarios del sistema. El usuario sin conocimientos de CAD (**Persona A**). El usuario profesional (**Persona B**).

 Valeria D.	35 años Lic. en Artes plásticas Docente Investigadora Artista	 Pedro C.	37 años Diseñador Industrial Docente Investigador Modelador 3D indepte Maker de impresión 3D
<b>Frustraciones</b> <ul style="list-style-type: none"> <li>- Hice el pedido de una escultura y no comprendo el lenguaje del diseñador</li> <li>- No puedo ver el avance porque no utilizo SolidWorks o Cinema4D.</li> <li>- Los emails son un caos no puedo seguir el hilo</li> <li>- Es difícil poder opinar sobre el diseño.</li> </ul>	<b>Soluciones potenciales</b> <ul style="list-style-type: none"> <li>- Sería útil algo como Google Docs pero con modelos 3D.</li> <li>- Algo simple de usar para comentar como las redes sociales</li> </ul>	<b>Frustraciones</b> <ul style="list-style-type: none"> <li>- Lo principal es la gestión del feedback, no queda nada asentado, lo que produce problemas de confianza.</li> <li>- Problemas por falta de conocimiento del cliente sobre 3D.</li> <li>- Problemas para gestionar en el email, las referencias, avances.</li> <li>- Es muy difícil que el cliente vea todo el proceso complejo del trabajo de diseño</li> </ul>	<b>Soluciones potenciales</b> <ul style="list-style-type: none"> <li>- Algo como la herramienta de Thingverse pero con más control.</li> <li>- Algo simple de usar para el cliente y que pueda comentar sobre los modelos</li> <li>- Algo que sirva para tener idea de todos los cambios que se realizaron en los modelos.</li> </ul>

Figura 4.1: Personas utilizadas en CoCADA. El usuario sin conocimientos de CAD (Persona A) (izquierda). Usuario profesional (Persona B) (derecha). En la parte superior se especifica la información general del usuario. El cuadrante inferior izquierdo debe contener las necesidades y frustraciones respecto la solución actual, los puntos de conflicto específicos que el producto intenta resolver y/o la oportunidad que se trata de capturar con él. El cuadrante inferior derecho contiene las soluciones potenciales, sugeridas por el usuario.

#### 4.1.3. Funciones o funcionalidades

Con todo el material obtenido se organizan a continuación las sub-hipótesis que deben probarse.

Núm.	Se desarrolla (función)	Para (persona)	Para (solución)
1	Un visor de modelos 3D en la web	Para (Persona A) y (Persona B)	Solucionar de forma eficiente la visualización de los productos
2	Un sistema de comentarios asociado a los modelos 3D	Para (Persona A) y (Persona B)	Discutir sobre los avances del proyecto.
3	Un mecanismo para anotaciones sobre los modelos 3D	Para (Persona A) y (Persona B)	Poder comunicar aspectos de diseño entre personas de diferentes disciplinas
4	Un sistema para dar de Alta proyectos	Para (Persona B)	Gestionar nuevos proyectos de diseño.
5	Un mecanismo para escribir código de modelado	Para (Persona B)	Incorporar diseños y modificarlos mediante scripting
6	Un mecanismo para ver los cambios o “versiones” de los diseños	Para (Persona A) y (Persona B)	Visualizar la evolución de los modelos y su estado en una fecha determinada
7	Componentes visuales para modificar los parámetros del modelo 3D	Para (Persona A) y (Persona B)	Modificar geometrías de forma intuitiva, sin necesidad de tener conocimientos de programación o de modelado avanzado
8	Un mecanismo para inicio de sesión con usuario y contraseña	Para (Persona A) y (Persona B)	El ingreso privado

Tabla de sub-hipótesis para CoCADA

## 4.2. Diseño Colaborativo

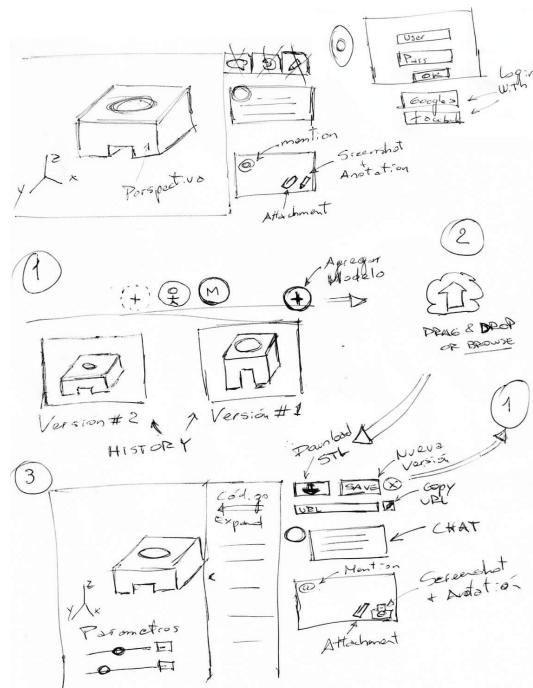


Figura 4.2: Resultado del estudio de diseño explicado en la sección 3.4. Sketches o bocetos de interfaces de usuario para CoCADA.

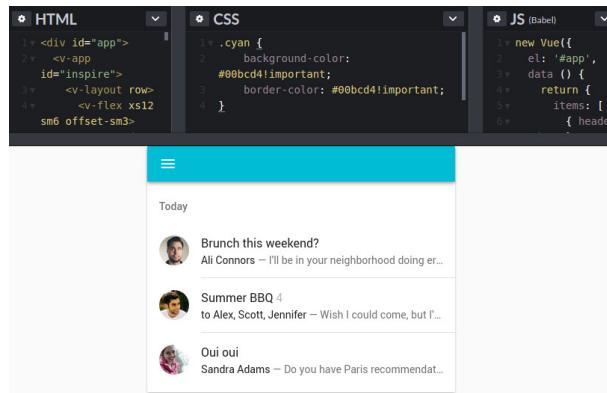


Figura 4.3: Guía de estilo utilizada en CoCADA. Ejemplo de componente web para conversaciones y su forma de uso mediante código HTML, CSS y javascript.

### 4.3. PMV y experimentos

El primer PMV o **Demo #1** evaluó la **sub-hipótesis 1** de la tabla de sub-hipótesis: “**Un visor de modelos 3D en la web para el usuario (Persona A) y (Persona B) para solucionar de forma eficiente la visualización de los productos**”.

Se utilizaron tres preguntas para orientar el experimento:

1. **¿Quién interactuará efectivamente con el prototipo?**

El usuario sin conocimientos de CAD (Persona A).

2. **¿Qué se espera aprender de él?**

Aspectos de funcionalidad y usabilidad de la interfaz UI al momento de interactuar con un modelo 3D.

3. **¿Cuánto tiempo se tiene para desarrollar el prototipo?**

Aproximadamente 4 días.

Se utilizaron **prototipos de alta fidelidad** ([Gamble, 2016](#)) para lograr un aspecto y comportamiento similar al de la interfaz definitiva.

En el primer experimento se presentó un prototipo de visor con un modelo 3D (ver figura 4.4), programado en base al código de OpenJSCAD en un tiempo aproximado de 36 horas.

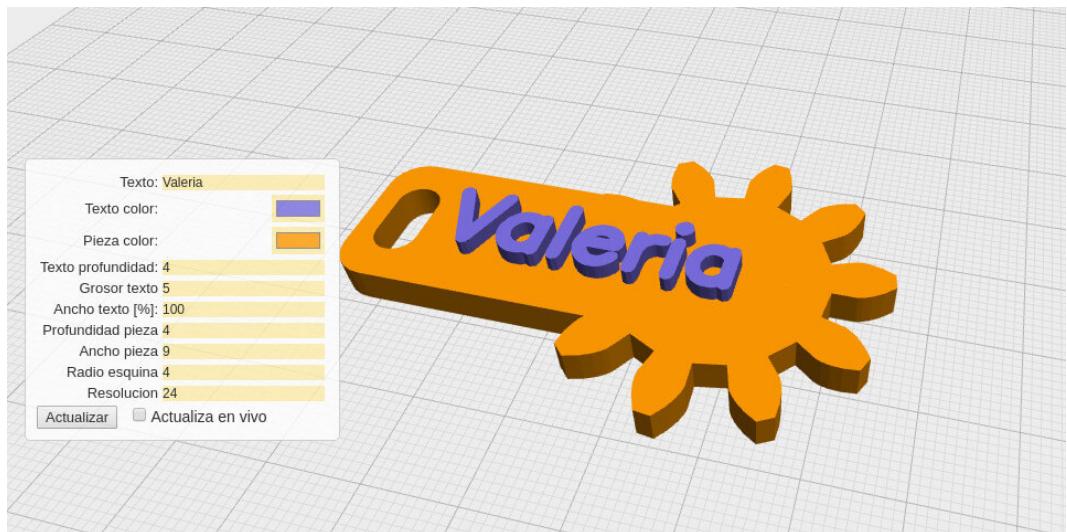


Figura 4.4: Pantalla de Demo #1. Visor con un modelo 3D (centro) y funcionalidades de zoom (alejar, acercar), rotación y traslación. Parámetros para modificar la geometría y el color (izquierda).

### Características del experimento:

Se utilizó una computadora con conexión a internet y un navegador web mostrando el modelo por defecto en la pantalla. El usuario no recibió instrucciones sobre el uso y no tiene limitaciones de tiempo. El evaluador se limitó a observar y registrar la actividad.

## 4.4. Feedback

Al final de la sesión de prueba se realizaron una serie de preguntas sobre la experiencia de usuario, divididas en tópicos.

Las respuestas posibles fueron: **Buena, Regular o Mala**. En caso de responder Regular o Mala se realizaron otras 2 preguntas: **¿Cuál es el inconveniente?** y **¿Qué sugiere para resolver ese inconveniente?**

En la siguiente tabla se detallan los tópicos y sus respuestas:

Ítem	Tópico	Experiencia	Inconveniente	Cómo mejorar
1	Visualización del Modelo	Buena	-	-
2	Zoom	Regular	Al hacer demasiado zoom sobre el modelo ocurre que no hay un límite, lo que produce que se rompa la geometría.	Limitar el zoom o ver la posibilidad de volver la visualización a su estado original.
2	Rotación	Buena	-	-
3	Translación	Mala	Al mover demasiado el modelo, ocurre que se pierde la visualización y parece que la escena está vacía.	Limitar la opción de mover o ver la posibilidad de volver la visualización a su estado original.
4	Modificación de parámetros	Regular	Es incómodo ingresar números utilizando el teclado	Agregar un elemento parecido a los que tienen las apps.

#### 4.4.1. Resultados y propuestas

Luego de replicar y analizar las **experiencias de usuario negativas**, se construyó un prototipo **Demo #2** en base al feedback y las recomendaciones del usuario en el experimento con Demo #1.

- **Zoom** (alejar, acercar). En la figura 4.5 se puede apreciar como se “rompe” la geometría. **Solución:** Se agrega un botón de “Resetear Vista” (izquierda) para establecer la cámara en la posición original, con el objeto centrado en la pantalla (ver figura 4.7 , izquierda).

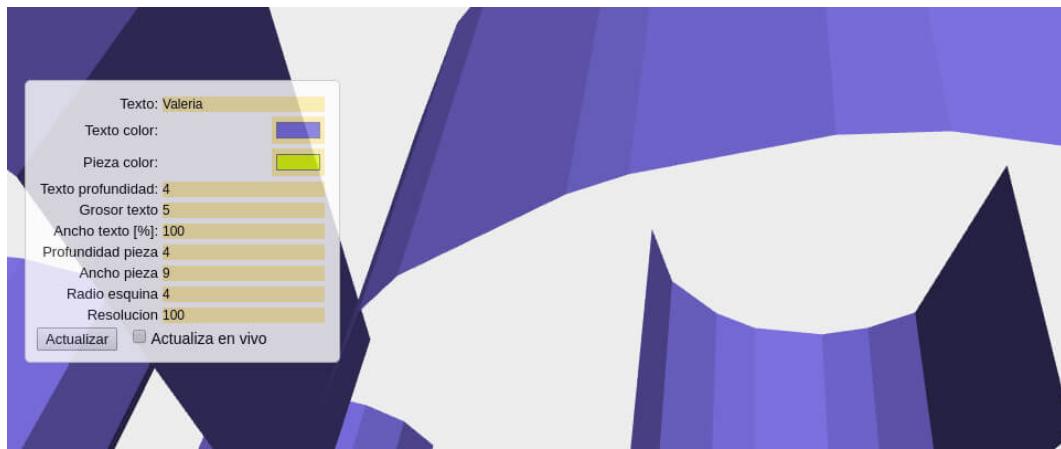


Figura 4.5: Inconvenientes con el zoom en Demo #1. Se “rompe” la geometría.

- **Traslación** (Mover). En la Figura 4.6 el modelo prácticamente desaparece de la pantalla. **Solución:** Se utiliza la misma funcionalidad de el punto anterior.
- **Modificación de parámetros.** La mayoría de las personas están acostumbrados a las interfaces con componente gráficos como deslizadores en los que no es necesario escribir valores numéricos. Tanto en la figura 4.5 como en la figura 4.6(izquierda) no se visualizan este tipo de elementos.

**Solución:** Se agrega un componente *slider* como parámetro “Texto profundidad”(ver figura 4.7 , izquierda) para evitar la carga de valores numéricos por teclado. El resto de los campos de texto no se han modificado, intencionalmente, para que el usuario pueda evaluar las diferencias en el uso.

Al finalizar cada experimento se vuelve al ciclo *construir-medir-aprender* visto en la sección 3.4, hasta obtener un prototipo que satisfaga al usuario. De esa manera se obtuvieron recomendaciones interesantes para ser implementadas en el futuro: en

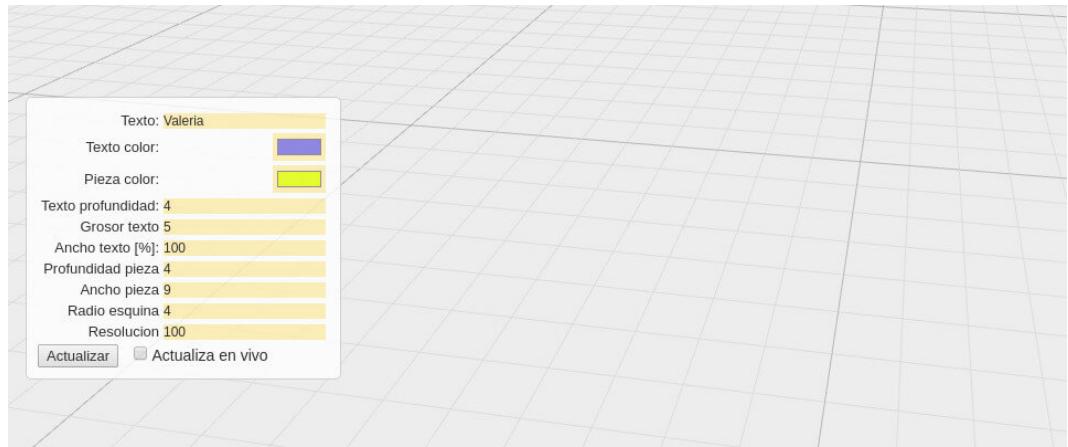


Figura 4.6: Inconvenientes con la traslación en Demo #1. El objeto se sitúa fuera del campo de visión. También se observan problemas con la modificación de parámetros (izquierda), el uso de campos de texto no es adecuado para valores numéricos.



Figura 4.7: Demo #2. Corrección de inconvenientes hallados en Demo #1. Se agrega un botón “Resetear vista” para solucionar el problema del zoom. Se agrega el componente “slider” como parámetro “Texto profundidad” para mejorar la experiencia de usuario al modificar los valores numéricos.

el experimento con Demo #2, el usuario sugirió utilizar la funcionalidad de *reset*<sup>1</sup> pero en los parámetros y de forma individual, un requerimiento de la vista general adaptado a los componentes.

Con la tabla de sub-hipótesis y lo aprendido con el usuario en los experimentos, finalmente se estableció la arquitectura de software necesaria para dar soporte a todas las funcionalidades de CoCADA.

---

<sup>1</sup>Reiniciar, en inglés *reset* se conoce como la puesta en condiciones iniciales de un sistema.

## 4.5. Sistema CoCADA

En la sección 4.1.3 se identificaron las funcionalidades que la aplicación debe proveer a los usuarios. Adicionalmente, para una colaboración distribuida, se requiere de la **Gestión de datos de productos basado en la web** (WPDM), explicada en la sección 3.3.3. CoCADA se divide en dos áreas fundamentales: *Back-End* y *Front-End* (Mardan, 2015).

### 4.5.1. Back-End

La aplicación de *Back-End* o capa de servicios cuenta con tecnologías para tareas que no pueden ser resueltas directamente por los usuarios como la persistencia de datos y el almacenamiento de archivos. Las tecnologías utilizadas son:

- **Node.js**<sup>2</sup> como web server y entorno de ejecución para JavaScript en el lado del servidor, de la misma manera que lo hacen otros lenguajes como PHP<sup>3</sup> o Python.
- **Nuxt.js**<sup>4</sup>. Es un framework para crear aplicaciones isomórficas o universales con Vue.js<sup>5</sup>. Una aplicación universal es aquella que su código puede ser ejecutado tanto en el cliente (navegador web) como en el servidor. Nuxt.js incorpora el concepto *Server Side Rendering* (SSR)<sup>6</sup>. El SSR en CoCADA brinda la posibilidad de convertir los componentes web en cadenas de HTML en el servidor, luego son enviadas al navegador web y se genera la aplicación en el lado del cliente.
- **LoopBack**<sup>7</sup>. Es un conjunto de módulos de Node.js que permiten la implementación de APIs (Masse, 2011) altamente flexibles. Está basado en el framework **Express.js**<sup>8</sup> y proporciona funcionalidades para:

---

<sup>2</sup><https://nodejs.org/es/>

<sup>3</sup><http://php.net/manual/es/intro-whatis.php>

<sup>4</sup><https://nuxtjs.org/>

<sup>5</sup><https://vuejs.org/>

<sup>6</sup><https://ssr.vuejs.org/#what-is-server-side-rendering-ssr>

<sup>7</sup><https://loopback.io>

<sup>8</sup><https://expressjs.com/es/>

- Crear APIs REST dinámicas end-to-end<sup>9</sup> con poca codificación.
  - Acceso a datos para los principales motores de bases de datos, servicios SOAP y otras APIs.
  - Almacenamiento de archivos o *file storage*.
  - Inicio de sesión mediante protocolos de autorización OAuth<sup>10</sup>.
- Nuxt.js se comunica con LoopBack a través de su API REST , obteniendo como respuesta es un documento JSON.

- **MongoDB**<sup>11</sup>. El enfoque de CoCADA es agnóstico<sup>12</sup> respecto al motor de base de datos, sin embargo, se utiliza MongoDB (NoSQL) para la persistencia de datos. A diferencia de las bases de datos relacionales, los datos se almacenan en tablas, sino mediante documentos en formato JSON. Esto permite que la integración entre MongoDB y javascript sea mucho más directa.

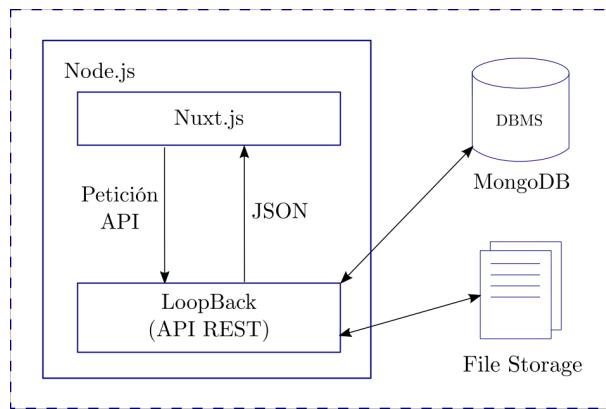


Figura 4.8: Backend de CoCADA. Node.js se utiliza como web server y entorno de ejecución para JavaScript. Nuxt.js provee el Server Side Rendering (SSR). Loopback implementa la API REST y gestiona los archivos (File Storage). MongoDB es el motor de bases de datos. Nuxt.js se comunica con LoopBack a través de su API, obteniendo como respuesta un documento JSON.

---

<sup>9</sup>End-to-end es un enfoque que involucra la visión global del encadenamiento de procesos y/o actividades, desde que surge una necesidad a satisfacer, hasta que esta es satisfecha

<sup>10</sup><https://oauth.net/2/>

<sup>11</sup><https://www.mongodb.com/es>

<sup>12</sup>Se refiere a la capacidad de interoperabilidad y compatibilidad de un componente de cómputo entre diversos sistemas y ambientes, sin requerir una adaptación especial.

## CoCADA API REST

En esta sección se desarrolla el modelo lógico del sistema, incluyendo el soporte para **Intercambio de datos Basado en Características** (FBDE) explicado en la sección 3.3.4. El diagrama de la figura 4.9 ilustra el esquema de datos y las relaciones entre las entidades. Las especificaciones del modelo y la fuentes de datos (base de datos) se realizan con LoopBack mediante archivos JSON.

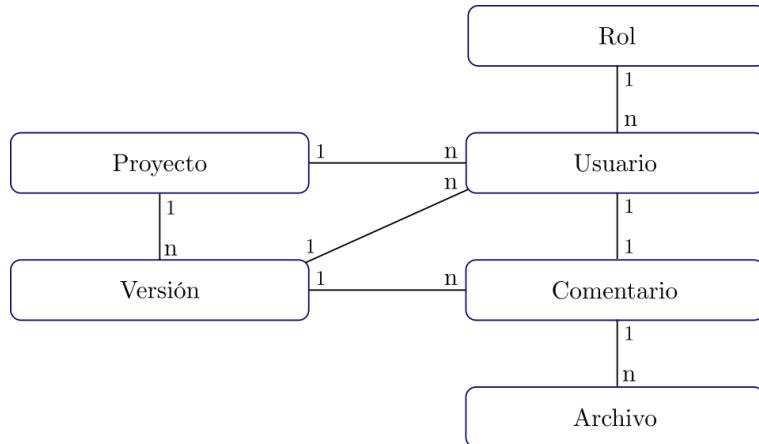


Figura 4.9: Modelo de datos para CoCADA. Un Proyecto puede contener versiones de un producto (árbol de historias). Una Versión puede ser utilizada por uno o más usuarios en el marco de trabajo, también soporta conversaciones de estos usuarios a través de Comentarios. Un Usuario puede asumir un rol específico. Un Comentario puede contener uno o más archivos.

- **Proyecto**: Es la entidad que gestiona la evolución de los diseños. Puede contener una o más versiones de un producto, también llamado árbol de historias (ver figura 4.9).

En la figura 4.10 se puede analizar un proyecto con su respectivas versiones y la evolución de los diseños.

- **Versión**: Es la representación de un diseño (producto) individual y su información relacionada. También se puede entender como una iteración en el proceso de diseño.

- **Usuario**. La entidad representa a los usuarios del sistema.

- **Comentario**. Es la entidad que gestiona los comentarios asociados a una Versión.

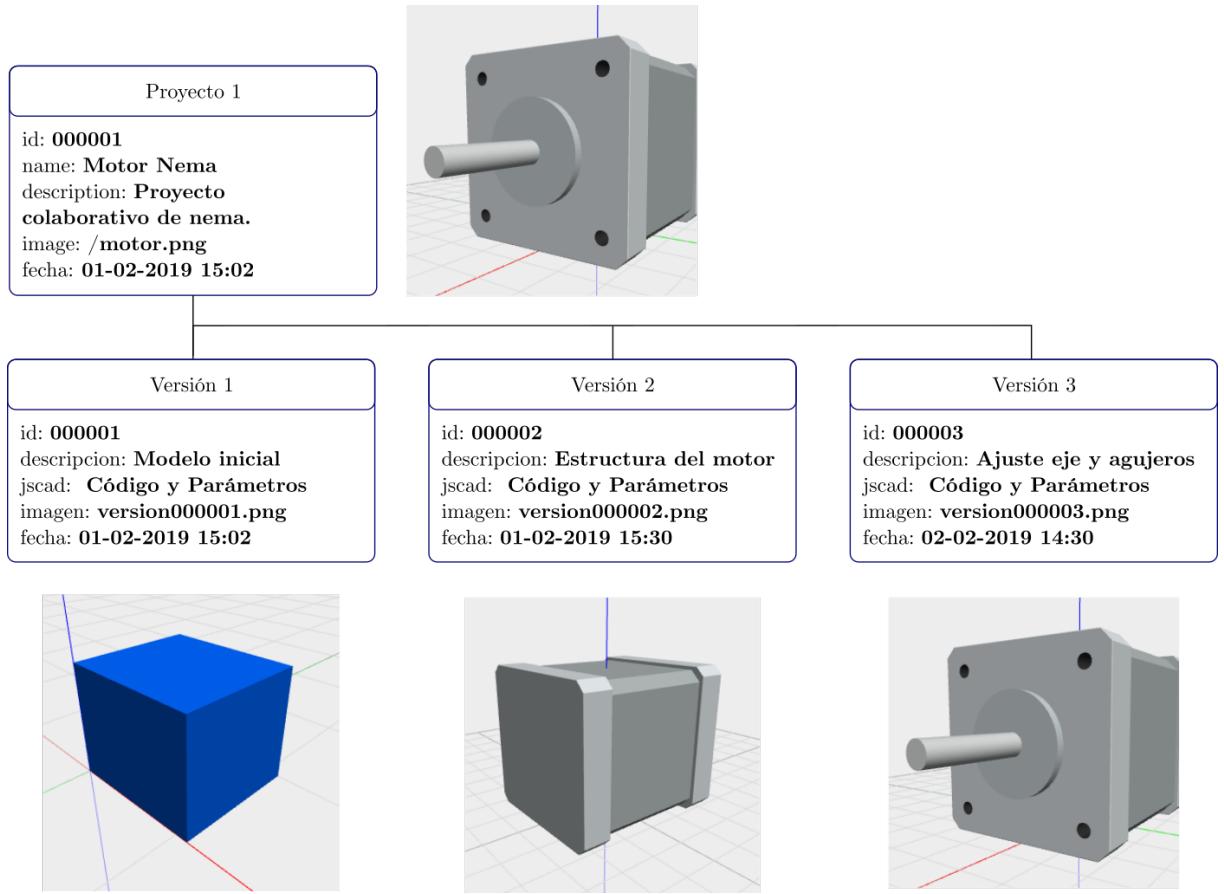


Figura 4.10: Un proyecto con su árbol de historia o versiones. Se puede apreciar la evolución del producto ordenada por fecha ascendente. La imagen que identifica el *Proyecto 1* es la misma que la última versión del producto. La *Versión 1* se inicia al momento de generar el proyecto, con un modelo de ejemplo (cubo). La *Versión 2* se genera a partir de la anterior, con los cambios efectuados en el código y así sucesivamente.

- **Rol.** Especifica los permisos de los usuarios dentro del sistema, un mismo rol puede ser asignado a diferentes usuarios.
- **Archivo.** Esta entidad contiene la dirección web o URL de un archivo, en CoCADA estos archivos se asocian a los comentarios.

Para facilitar el desarrollo, LoopBack provee una herramienta (*API Explorer*<sup>13</sup>) para explorar las entidades, los métodos HTTP disponibles y los *EndPoints*<sup>14</sup> de la API. En la figura 4.11 se puede ver un modelo “Versión” utilizado como prueba.

<sup>13</sup><https://loopback.io/doc/en/lb3/Use-API-Explorer.html>

<sup>14</sup>Un Endpoint en la API de CoCADA es una URL única que representa un objeto o una colección de objetos.

The screenshot shows the LoopBack API Explorer interface for the 'version' model. At the top, there are tabs for 'Show/Hide', 'List Operations', and 'Expand Operations'. Below this, there are two main sections: 'PATCH /versions' and 'GET /versions'. The 'PATCH' section is described as 'Patch an existing model instance or insert a new one into the data source.' The 'GET' section is described as 'Find all instances of the model matched by filter from the data source.' Under the 'GET' section, there is a 'Response Class (Status 200)' tab and a 'Model Schema' tab. The 'Model Schema' tab displays the JSON schema for the 'version' model:

```
{
  "projectId": "string",
  "parentVersionId": "string",
  "description": "string",
  "image": "/containers/projects/download/default-project.png",
  "jscadCode": "function getParameterDefinitions() { \n\n\treturn [ \n\n\t\t{ name: 'sizeX', caption: 'Tamaño X', type: 'slider' } ]\n  }",
  "authorId": "string",
  "id": "string",
  "version_id": "string",
  "createdAt": "2019-01-30T22:16:28.727Z",
  "updatedAt": "2019-01-30T22:16:28.727Z"
}
```

Below the schema, there is a 'Response Content Type' dropdown set to 'application/json'. A 'Parameters' table follows, with a single row for 'filter':

Parameter	Value	Description	Parameter Type	Data Type
filter	<input type="text"/>	Filter defining fields, where, include, order, offset, and limit - must be a JSON-encoded string ({"something":"value"})	query	string

A 'Try it out!' button is located at the bottom left of the parameters section.

Figura 4.11: Herramienta API Explorer de LoopBack. Se muestra el contenido de un modelo “Versión”, se pueden apreciar los datos del producto.

#### 4.5.2. Front-End

En las aplicaciones web, el *Front-End* o capa de presentación implica el uso de las tecnologías con las que interactúa directamente el usuario. Normalmente estas tecnologías son desarrolladas en los lenguajes HTML, CSS, javascript y otros recursos como imágenes y gráficos vectoriales (ver figura 4.12). Al ingresar al sistema, se muestra la interfaz de usuario UI también llamada CoCADA APP.

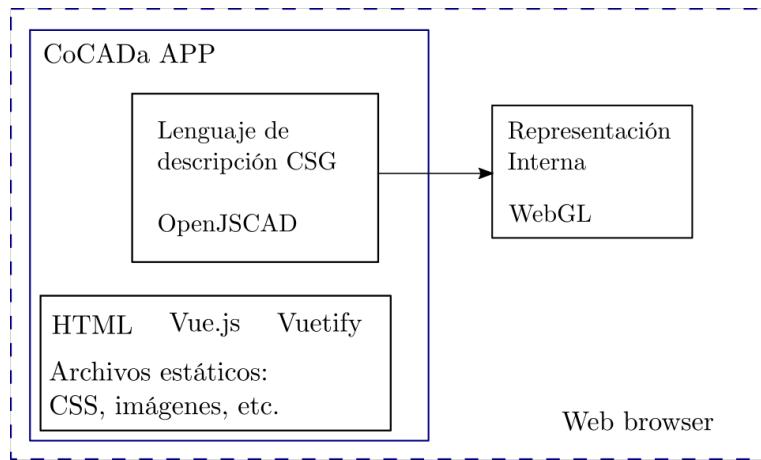


Figura 4.12: Frontend de CoCADA. CoCADA APP está compuesta por OpenJSCAD (Lenguaje de descripción) y Vue.js como sistema reactivo + Vuetify (Material Design). WebGL resuelve la representación interna (matemática) en el navegador web.

Para la implementación se utilizan las siguientes tecnologías:

- **OpenJSCAD** (javascript) como **lenguaje de descripción CSG** para trabajar en alto nivel y de forma intuitiva con las primitivas, transformaciones, operaciones booleanas, etc. Además provee mecanismos para la manipulación visual de parámetros y exportación de los modelos en formato STL. Una vez procesadas las sentencias del lenguaje de descripción, se traducen a su **representación interna** (matemática) vista en la sección 2.3.1 mediante la API **WebGL** del navegador web (ver figura 4.12).
- **Vue.js**<sup>15</sup>. Es un framework progresivo para interfaces de usuario, lo que significa que se pueden incorporar herramientas incrementalmente, a medida que aumenta la complejidad de la aplicación. Una ventaja de usar esta tecnología es su característica de sistema **reactivo** (Mezzalira, 2018), manteniendo una interacción constante con su entorno y permitiendo el cambio de estado interno por medio de eventos. Cuando los datos de la interfaz son modificados

<sup>15</sup><https://vuejs.org/>

por alguna acción del sistema o del usuario, por ejemplo: cada vez que se hace una petición al *Back-End* (ver figura 4.19), tiene la capacidad de modificar solamente los componentes necesarios, sin necesidad de actualizar toda la aplicación en el navegador web.

- **Vuetify<sup>16</sup>**. Es un framework progresivo orientado a componentes visuales, se basa en el concepto *Material Design*<sup>17</sup>, con elementos reconocibles por los usuarios en la mayoría de las apps. Se utiliza para lograr una experiencia de usuario satisfactoria, en base al diseño establecido en la sección 4.2.

Para una mejor comprensión de la UI, se hace una distinción entre pantallas y componentes.

## Pantalla de Proyectos

Luego del *login* mediante un usuario y contraseña, se muestra una pantalla con el listado de proyectos (ver figura 4.13) (izquierda).

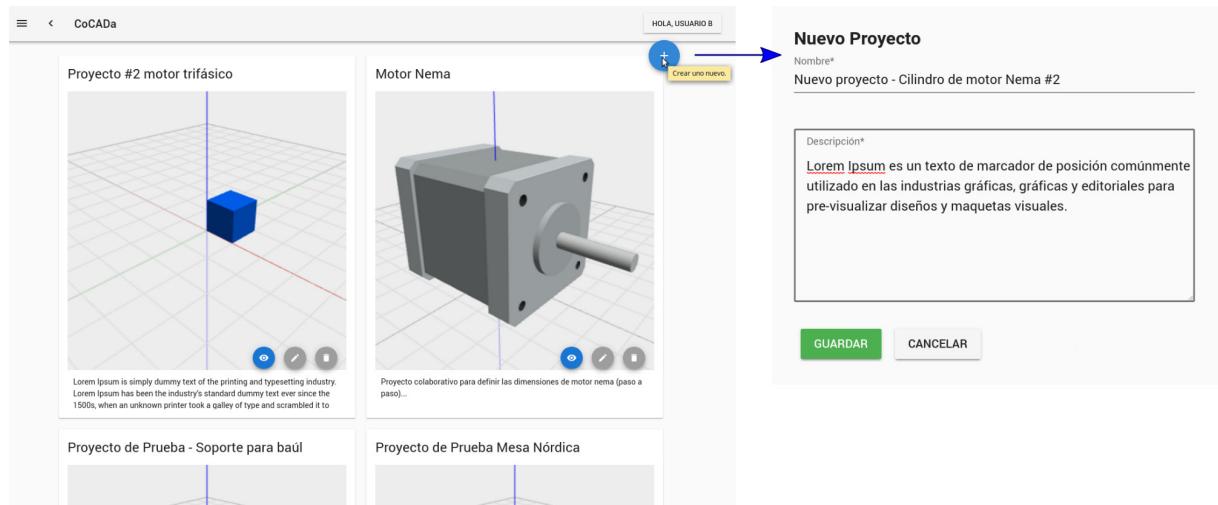


Figura 4.13: Pantalla de proyectos (izquierda), en primera instancia se muestra un listado de proyectos con la posibilidad de ver, editar y eliminar los mismos. Al hacer click en la llamada a la acción (botón) en la parte superior derecha de la ventana se agrega un nuevo proyecto. A la derecha se puede observar el formulario de alta para un nuevo proyecto. Los usuarios Persona A y Persona B pueden agregar proyectos.

<sup>16</sup><https://vuetifyjs.com/en/>

<sup>17</sup><https://material.io/design/>

## Pantalla de Producto

Un producto se refiere a una versión específica del mismo dentro del árbol de historia. Al agregar un nuevo producto o al “ver” un proyecto del listado, se muestra la pantalla de la figura 4.14, correspondiente a la vista de producto.

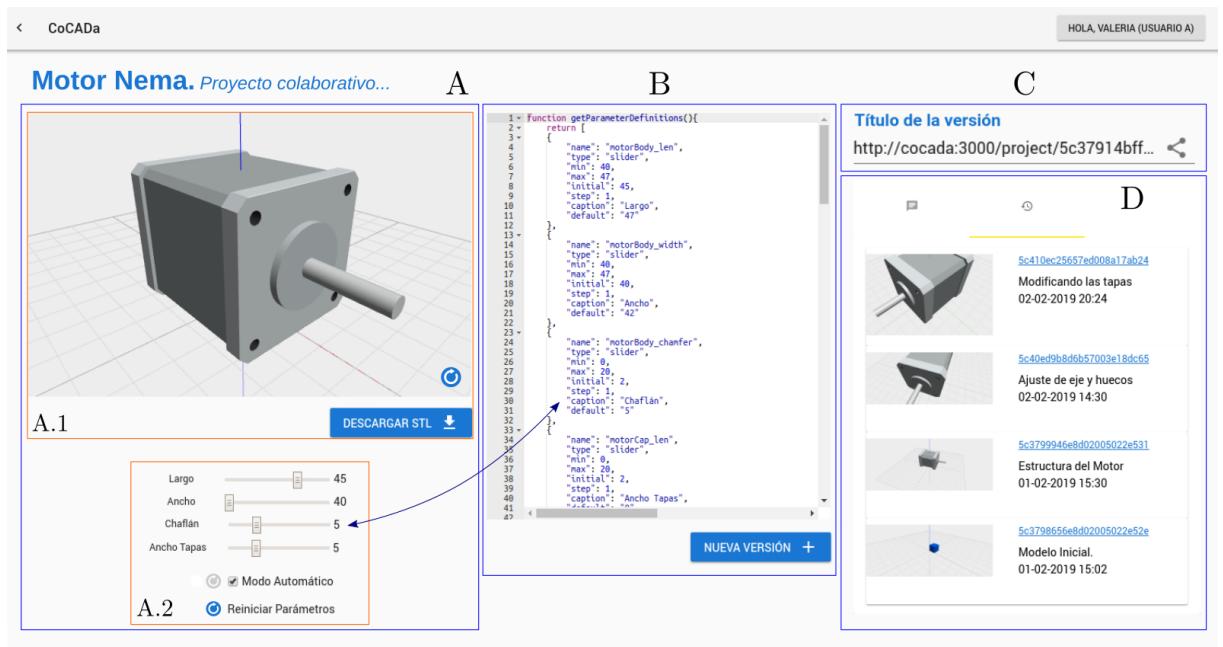


Figura 4.14: Pantalla de Producto. Visor 3D (A) (izquierda). Editor de código (B) (centro). Componente para compartir el modelo (C) (derecha y arriba). Componente de comentarios e historial de modelos (D) (derecha y abajo).

Los componentes web de la pantalla son:

- **A. Componente del Visor 3D** (izquierda).

**A1. Modelo 3D.** Se visualiza el modelo mediante OpenJSCAD y WebGL.

Funcionalidades:

- **Zoom** (in - out) mediante el scroll con botón del medio del mouse.
- **Rotación** mediante click con botón izquierdo y moviendo el mouse.
- **Traslación** mediante la tecla shift + botón izquierdo presionados y moviendo el mouse.
- **Reiniciar Vista** mediante el botón ubicado abajo y a la derecha en el visor (sitúa la cámara en su posición original).
- **Descargar STL.** Se genera y descarga el modelo sólido en formato STL (preparado para la fabricación digital)(ver figura 4.15).



Figura 4.15: Ejemplo de fabricación digital con impresora 3D.

**A2. Parámetros.** Se sitúa en la parte inferior izquierda. Tiene los siguientes componentes:

- **Parámetros de control.** Estos campos se configuran previamente mediante código en el componente B (la flecha en la figura 4.14 indica la relación con el código que genera el campo).
- **Modo Automático.** Permite visualizar automáticamente los cambios, ya sea modificando los parámetros o editando el código. En caso de deshabilitar la opción, los cambios son manuales mediante el botón de la izquierda (útil para modelos con gran cantidad de polígonos).
- **Reiniciar parámetros.** Permite volver los campos de los parámetros al valor por defecto especificado en el código.

#### • B. Componente Editor de código.

Se sitúa en la parte central del layout y es visible únicamente por el usuario Persona B (diseñador, establecido en la sección 4.1.2).

CoCADA permite editar código javascript en la web con capacidad de resaltado de sintaxis mediante el editor **Ace Editor**<sup>18</sup>.

Si la opción de “actualizar automáticamente” se encuentra habilitada, el código se ejecuta de forma automática. De lo contrario es necesario ejecutar el programa mediante las teclas shift + enter.

Un script OpenJSCAD debe tener al menos una función definida, la función *main()*, que retorna un objeto CSG o un vector (array) de dos o más

---

<sup>18</sup><https://ace.c9.io/>

objetos CSG. A continuación se analiza un programa que genera un objeto 3D mediante una operación booleana y transformaciones geométricas. En el código se puede apreciar el uso de una función ordinaria de javascript para calcular un radio, transformaciones sobre los cilindros (rotación), una operación booleana (diferencia) entre una esfera y los tres cilindros; y la definición de parámetros para el usuario. El resultado del script se puede ver en la figura 4.16.

```

1  function radiusFromDiameter (d) {// Función de javascript
2    return d / 2;
3  }
4
5  function rotcy (rot, r, h) {// Función para rotar los cilindros
6    return rotate(rot, cylinder({r: r, h: h, center: true}));
7  }
8
9  function esferaHuecos (params) {
10    var size = params.size;
11    var hole = params.hole;
12    var radius = radiusFromDiameter(hole);
13    var height = radiusFromDiameter(size * 2.5);
14    // Operación booleana: diferencia entre una esfera y 3 cilindros
15    return difference(
16      sphere({r: radiusFromDiameter(size)}),
17      rotcy([0, 0, 0], radius, height),
18      rotcy([90, 0, 0], radius, height),
19      rotcy([0, 90, 0], radius, height)
20    );
21  }
22
23  function main (params) {
24    return esferaHuecos(params);
25  }
26
27  function getParameterDefinitions () { // Parámetros
28    return [ { name: 'size', caption: 'Tamaño Esfera:', type: 'slider', initial: 30, min: 5, max: 100, step: 1 }, { name: 'hole', caption: 'Tamaño Huecos:', type: 'slider', initial: 10, min: 5, max: 50, step: 1 } ];
29
30
31
32 }
```

Todas las funcionalidades de OpenJSCAD y ejemplos de código se puede estudiar en la guía oficial<sup>19</sup>.

- - **Nueva Versión.** Genera una nueva versión del espacio de trabajo (Producto) y luego está disponible en el historial de versiones en el componente (D). El

---

<sup>19</sup><https://openjscad.org/dokuwiki/doku.php>

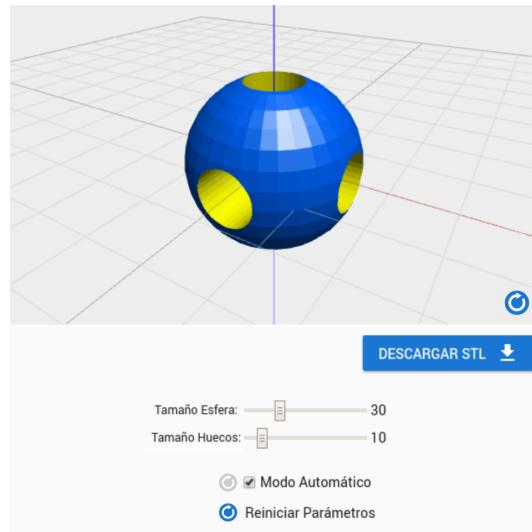


Figura 4.16: Modelo 3D resultante de la operación booleana (diferencia) entre una esfera y tres cilindros transformados. En la parte inferior se pueden apreciar los parámetros definidos en el código.

código versionado toma como base los parámetros modificados.

- **C. Componente Compartir modelo.**

Se sitúa en la parte superior derecha de la pantalla. Se comparte un enlace web a una pantalla con visor reducido con el modelo. El objetivo es poder mostrar un diseño a usuarios anónimos.

- **D. Componente de Comentarios e Historial de modelos.**

Se sitúa en la parte inferior derecha del layout. Utiliza un mecanismo de pestañas en inglés *tabs* de manera que la selección de una opción oculte la otra, las opciones son “Comentarios” e “Historial” .

- **Componente de comentarios.** Se accede a través de la primer pestaña o ícono en forma de comentario. Su objetivo es lograr la comunicación entre los usuarios en forma de conversación o *chat* (ver 4.17 (izquierda)). Existen dos maneras de utilizar imágenes en los comentarios:

**Como imagen adjunta.**

Presionando el ícono de la esquina inferior izquierda del componente (ver figura 4.17 (izquierda)) es posible explorar archivos del dispositivo y adjuntar una imagen. **Como anotaciones en el modelo.**

- **Componente de Historial.**

Se accede a través de la segunda pestaña o ícono en forma de reloj. Permite visualizar todas las versiones o iteraciones de un producto, en orden

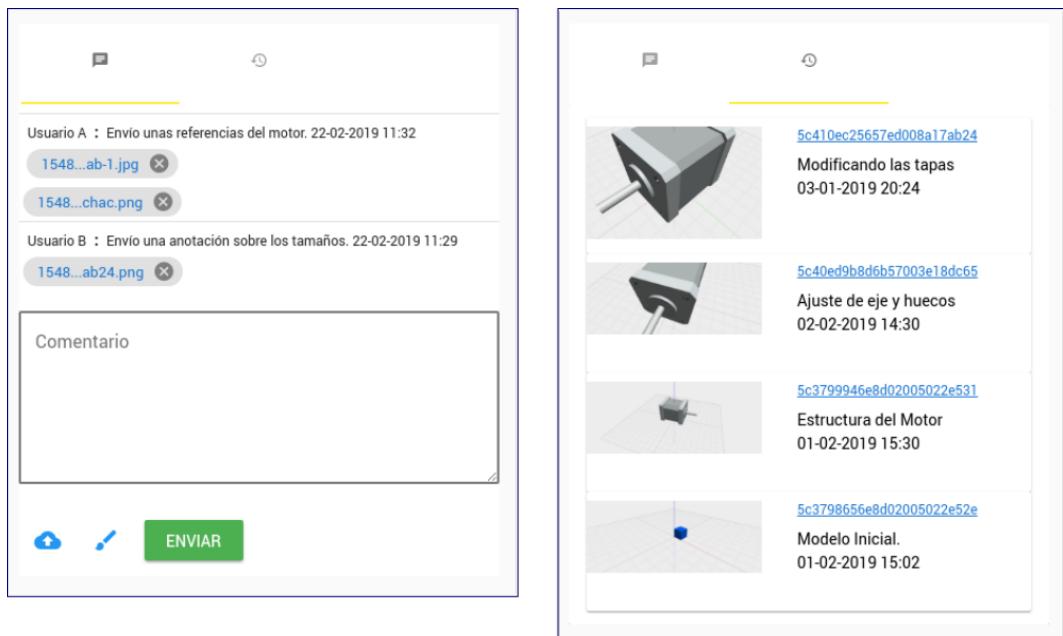


Figura 4.17: CoCADA. Pestaña de Comentarios: formulario para carga de comentarios en un producto (izquierda). Pestaña de Historial: Listado de productos o versiones (derecha).

cronológico. Una vez que se accede al producto o versión deseada, se puede modificar para generar otra versión. Esta funcionalidad es fundamental para soportar el diseño iterativo, explicado en la sección 3. Ver figura 4.17 (derecha).

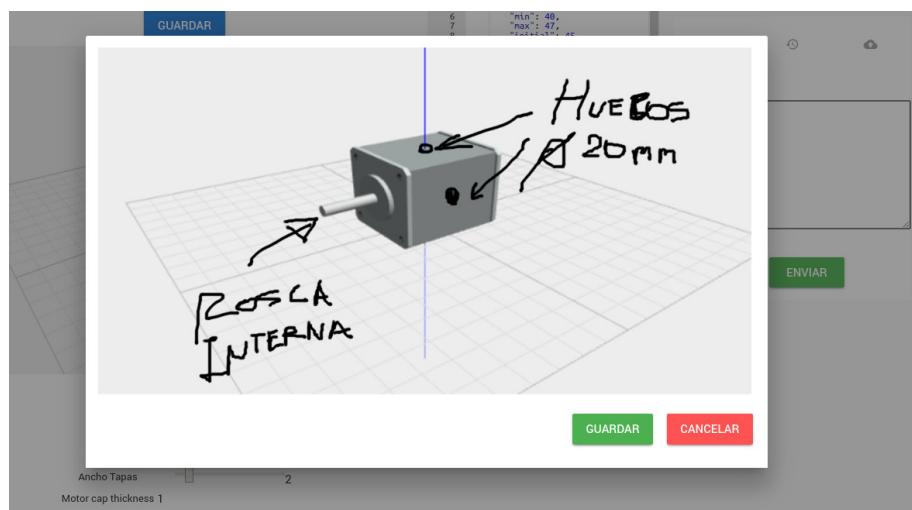


Figura 4.18: Herramienta para anotaciones sobre los modelos. Se utiliza la herramienta presionando el icono con forma de lápiz. Esto realiza una captura de pantalla del modelo y mediante una herramienta de dibujado permite hacer anotaciones sobre la imagen. Las anotaciones son muy útiles para comunicar de manera directa ideas sobre el modelo y lograr la colaboración entre participantes con diferentes competencias.

### 4.5.3. Interacción entre Usuarios, Front-End y Back-End

Mediante el gráfico resumido de la arquitectura (ver figura 4.19) se pueden analizar las interacciones de los usuarios y las áreas del sistema:

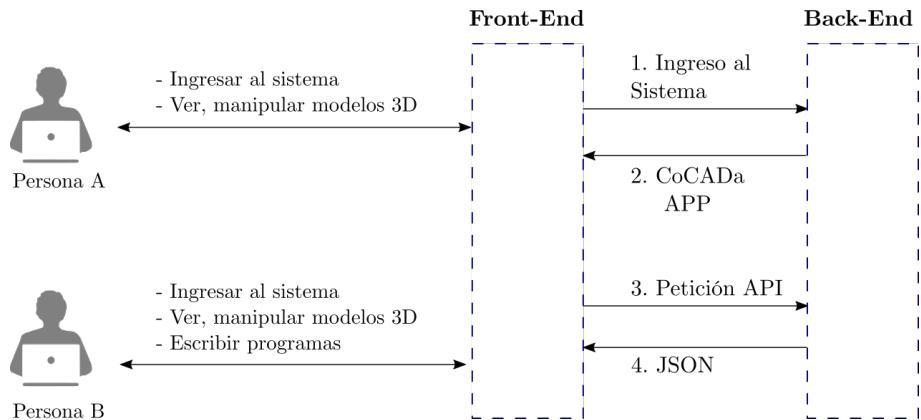


Figura 4.19: CoCADA interacción entre usuarios, Front-End y Back-End.

1. El usuario ingresa al sistema utilizando el navegador web (*Front-End*) mediante una URL, por ejemplo: <http://cocada:3000/proyecto/000001>.
2. Nuxt.js resuelve la petición en el servidor (*Back-End*) según la URL ingresada por el usuario, en caso de requerir recursos de la API se comunica con LoopBack. Finalmente, se genera la aplicación (CoCADA APP) con los datos solicitados y se envía al *Front-End*. A partir de este momento Nuxt.js no vuelve a intervenir.
3. El usuario a través de CoCADA APP realiza una acción, por ejemplo: *Login*, listar proyectos, ver un proyecto, generar una nueva versión, etc. El *Front-End* genera la petición API correspondiente mediante Vue.js.
4. LoopBack recibe la petición API, se encarga de resolver la acción y responder al *Front-End* mediante documentos en formato JSON. Por ejemplo: Durante el *Login*, se verifica el usuario y contraseña, como respuesta se obtienen los datos de aceptación o denegación. Si la petición es la “lista completa de los proyectos”, se retorna una colección de proyectos almacenados en la base de datos. Vue.js es el encargado de actualizar los cambios en la UI con los nuevos datos.

# Capítulo 5

## Conclusiones

Según lo investigado, la tendencia actual en el desarrollo de productos es la colaboración (co-diseño) entre personas con diferentes competencias y dispersas geográficamente. Por esto, surgió la necesidad de desarrollar una solución distribuida en la que los participantes puedan colaborar independientemente de su profesión.

Las herramientas CAD analizadas, en algunos casos no soportan la colaboración o son aplicaciones orientados a un perfil específico de profesionales (ingenieros, arquitectos, diseñadores industriales, etc).

El uso de *Lean UX* como metodología orientada a la experiencia de usuario (UX) permitió una comprensión de las necesidades reales. La utilización de la misma durante el desarrollo ha sido satisfactoria, debido a que las iteraciones o experimentos con prototipos de alta fidelidad (PMV) han permitido evaluar las soluciones de forma objetiva y obtener *feedback* de inmediato.

Los usuarios con experiencia en CAD (diseñadores) necesitan crear modelos 3D en un entorno que permita automatizar partes del diseño, de manera que se pueda registrar su evolución. En el capítulo 2 para tener una mejor comprensión de las herramientas necesarias para dicho entorno, se exploraron algunas aplicaciones de diseño paramétrico especificado en algoritmos, incluido OpenJSCAD. Además, deben poder comunicarse de forma eficiente con otros usuarios (no expertos).

Por otro lado, los usuarios no expertos necesitan de un marco de trabajo para expresar sus ideas y experimentar de forma intuitiva con modelos 3D. En este sentido, en el capítulo 3 se analizaron algunas aplicaciones de referencia que permiten explorar modelos 3D, realizar comentarios, anotaciones, exportar archivos, etc.

Para que CoCADA sea un sistema de CAD distribuido se han implementado características de **WPDM** (gestión de los datos del producto basados en la web), brindando acceso a usuarios dispersos geográficamente, independientemente de las

plataformas de software que utilicen. El registro de la evolución de los diseños se ha logrado mediante la incorporación de los conceptos **diseño paramétrico** (*scripting*) y **FBDE** (intercambio de datos basado en características). Se implementan en el WPDM como versiones del producto, también llamado “árbol de historias”. A su vez, en cada versión (marco de trabajo compartido) se permite la colaboración entre los usuarios (**co-diseño**) mediante las herramientas para manipular los diseños, comentar, adjuntar archivos y realizar anotaciones sobre los modelos.

El uso de herramientas **FLOSS** ha sido fundamental para adaptar el código fuente de otras aplicaciones a las necesidades del proyecto. Las tecnologías (Node.js, Nuxt.js, LoopBack, Vue.js, Vuetify) permiten emplear el mismo lenguaje de programación Javascript en toda la aplicación (*Backend* y *Frontend*), evitando la complejidad de mantener código de diferentes lenguajes.

Finalmente, la interfaz gráfica UI ha sido diseñada para una experiencia de usuario UX satisfactoria, facilitando así el auto-aprendizaje de los usuarios sin necesidad de un entrenamiento previo. En base a los experimentos, se llega a la conclusión que la aplicación puede ser utilizada en ambientes de gran complejidad técnica como el modelado mecánico y también para uso recreativo, por ejemplo en el diseño de piezas simples para impresión 3D.

Como trabajo futuro resulta interesante implementar comentarios y características propias de redes sociales como **@nombre** para hacer referencia al usuario destinatario del mensaje, notificaciones, mensajes de audio, etc.

Así también durante las pruebas, los usuarios sugirieron la posibilidad de visualizar los modelos considerando otros factores (texturas, iluminación, transparencias, etc). Sería posible extender los modelos sólidos a otras representaciones mediante librerías como Three.js<sup>1</sup> y así obtener otros resultados en la visualización de productos.

---

<sup>1</sup><https://threejs.org/>

# Bibliografía

*Introduction to Computer Graphics.* Addison-Wesley Professional, 1993. ISBN 978-0201609219. [15](#)

AA.VV. *Gaudí. Miscelania.* 2002. ISBN 84-9708-093-9. [11](#)

A. Alfaiate. A Browser-based Programming Environment for Generative Design. (May):83, 2017. [7](#), [13](#), [35](#)

L. A. Anthony Frausto-Robledo AIA. Vectorworks, inc. partners with modelo for aec presentation and collaboration plugin, 2016. URL <https://t.co/kn9SGu9CvB>. [Internet; descargado 28-noviembre-2018]. [62](#)

Autodesk. Acerca del dibujo paramétrico y las restricciones, 2017. URL <https://knowledge.autodesk.com/es/support/autocad/learn-explore/caas/CloudHelp/clouddhelp/2018/ESP/AutoCAD-Core/files/GUID-899E008D-B422-4DF2-AC8D-1A4F5701ED4E-htm.html>. [13](#), [14](#)

B. Berman. 3-d printing: The new industrial revolution. *Business Horizons*, 55 (2):155 – 162, 2012. ISSN 0007-6813. doi: <https://doi.org/10.1016/j.bushor.2011.11.003>. URL <http://www.sciencedirect.com/science/article/pii/S0007681311001790>. [6](#)

Bethany. How cad has evolved since 1982, 2017. URL <https://www.scan2cad.com/cad/cad-evolved-since-1982/>. [Internet; descargado 28-noviembre-2018]. [11](#)

T. R. Blender Foundation. Blender, 2018. URL <https://www.blender.org/>. [32](#)

G. Blokdyk. *Iterative Design: A Complete Guide.* CreateSpace Independent Publishing Platform, 2018. ISBN 9781717488121. URL <https://books.google.com.ar/books?id=2L9ptgEACAAJ>. [12](#)

- S. D. Bob McNeel. Grasshopper, 2018. URL <https://www.grasshopper3d.com/>.  
33
- H. Bohnacker, B. Gross, J. Laub, and C. Lazzeroni. *Generative Design: Visualize, Program, and Create with Processing*. Princeton Architectural Press, 2012. ISBN 9781616890773. URL <https://books.google.com.ar/books?id=tSS9uAAACAAJ>. 13
- D. Borodin. The lean startup by e. ries, summary review, 2018. URL <https://the.gt/lean-startup-e-ries-summary-review/>. [Internet; descargado 28-noviembre-2018]. 56
- T. Brown. Desing Thinking, 2009. ISSN 2385-7951. 54
- J. Burry and M. Burry. *The New Mathematics of Architecture*. Advances in geographic information science. Thames & Hudson, 2012. ISBN 9780500290255. URL <https://books.google.com.ar/books?id=9uJRpwAACAAJ>. 12, 14
- G. Celani. Teaching Cad Programming To Architecture Students. *Gestão & Tecnologia de Projetos*, 3(2):1–23, 2008. doi: 10.4237/gtp.v3i2.73. 15
- P.-Y. Chao and Y.-c. Wang. A data exchange framework for networked CAD/-CAM. *Computers in Industry*, 44(2):131–140, 2001. ISSN 01663615. doi: 10.1016/S0166-3615(00)00082-8. URL <http://linkinghub.elsevier.com/retrieve/pii/S0166361500000828>. 6
- D. Christev. Grasshopper - making a parametric bench, 2016. URL <https://www.youtube.com/watch?v=O3R1vXfIZF4>. [Internet; descargado 28-noviembre-2018].  
34
- C. Chronicles. Three Reasons to Start Designing Iteratively, 2009. 41
- G. Chryssolouris, D. Mavrikios, N. Papakostas, D. Mourtzis, G. Michalos, and K. Georgoulias. Digital manufacturing: History, perspectives, and outlook. *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, 223(5):451–462, 2009. ISSN 09544054. doi: 10.1243/09544054JEM1241. 6
- A. Cockburn. *Agile Software Development*. Agile software development series. Addison-Wesley, 2002. ISBN 9780201699692. URL <https://books.google.com.ar/books?id=JxYQ1Zb61zkC>. 42, 54

- P. Cromwell. *Polyhedra*. Cambridge University Press, 1999. ISBN 9780521664059.  
URL <https://books.google.com.ar/books?id=0Jowej1QWpoC>. 16
- D. Davis. A History of Parametric. *Http://Www.Danieldavis.Com/*, (2006):1–22, 2013. URL <http://www.danieldavis.com/a-history-of-parametric/>. 6, 10
- J. de Areba. *Metodología del análisis estructurado de sistemas*. Libros de Texto. Universidad Pontificia Comillas (Publicaciones), 2001. ISBN 9788484680437. URL <https://books.google.com.ar/books?id=PUqxsNVaQC8C>. 37
- Dimitrie. Speckle streams, 2017. URL <https://www.food4rhino.com/app/speckle-streams>. [Internet; descargado 28-noviembre-2018]. 61
- I. G. Dino. Creative design exploration by parametric generative systems in architecture. *Metu Journal of the Faculty of Architecture*, 29(1):207–224, 2012. ISSN 02585316. doi: 10.4305/METU.JFA.2012.1.12. 14
- S. Elfving. Managing Collaborative Product Development : A Model for Identifying Key Factors in Product Development Projects. nov 2007. 6, 38
- D. Flanagan. *JavaScript: la guía definitiva*. Anaya Multimedia/O Reilly. Anaya Multimedia, 2007. ISBN 9788441522022. URL <https://books.google.com.ar/books?id=mriFGgAACAAJ>. 34
- J. Foley, F. Van, A. Van Dam, S. Feiner, J. Hughes, J. Hughes, and E. ANGEL. *Computer Graphics: Principles and Practice*. Addison-Wesley systems programming series. Addison-Wesley, 1996. ISBN 9780201848403. URL <https://books.google.com.ar/books?id=-4ngT05gmAQC>. 29
- A. Gamble. *The Guide to Agile UX Design Sprints A Playbook for Product Teams*. 2016. 69
- GitHub. The state of Octoverse, 2018. URL <https://octoverse.github.com/>. 54
- J. Gothelf and J. Seiden. *Lean UX - Applying Lean Principles to Improve User Experience*. O'Reilly Media Inc., 2013. ISBN 9781449311650. doi: 10.1017/CBO9781107415324.004. URL <http://shop.oreilly.com/product/0636920021827.do><http://ebooks.cambridge.org/ref/id/CB09781107415324A009>. 8, 54, 55, 59, 60

- T. Grimm. *User's Guide to Rapid Prototyping*. Raymond F. Boyer Library Collection. Society of Manufacturing Engineers, 2004. ISBN 9780872636972. URL <https://books.google.com.ar/books?id=o2B70mABPNUC>. 47, 50
- R. Hartson and P. S. Pyla. *The UX Book: Process and guidelines for ensuring a quality user experience*. Elsevier, 2012. 7
- M. Huang, Y. Lin, and H. Xu. A framework for web-based product data management using j2ee. *The International Journal of Advanced Manufacturing Technology*, 24(11):847–852, Dec 2004. ISSN 1433-3015. doi: 10.1007/s00170-003-1697-8. URL <https://doi.org/10.1007/s00170-003-1697-8>. 49
- E. Huerta. La Co-Creación y el Diseño Colaborativo. page 3, 2013. URL [http://webprod.esdi.es/content/pdf/articuloweb\\_esdi-4\\_ehuerta180913.pdf](http://webprod.esdi.es/content/pdf/articuloweb_esdi-4_ehuerta180913.pdf). 38, 39
- G. Jeronimo. *Álgebra Lineal*. Departamento de Matemática, FCEyN, Universidad de Buenos Aires, 2008. ISBN ISSN 1851-1317. 16
- M. A. Kaled. Diseño Paramétrico - Aproximaciones al diseño generativo y su aplicación en el diseño industrial. 2016. URL [https://fido.palermo.edu/servicios\\_dyc/proyectograduacion/archivos/4161.pdf](https://fido.palermo.edu/servicios_dyc/proyectograduacion/archivos/4161.pdf). 12
- J. Kessenich, G. Sellers, and D. Shreiner. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 4.5 with SPIR-V*. OpenGL. Pearson Education, 2016. ISBN 9780134495538. URL <https://books.google.com.ar/books?id=vUK1DAAAQBAJ>. 28
- M. S. Kleinsmann. *Understanding Collaborative Design*, volume Revista-Es. 2006. ISBN 9090209743. URL <https://books.google.com.ar/books?id=zk5RXwAACAAJ>. 41
- P. Kukhnavets. Agile vs waterfall: Pros and cons, differences and similarities, 2016. URL <https://blog.ganttpro.com/en/waterfall-vs-agile-with-advantages-and-disadvantages/>. [Internet; descargado 28-noviembre-2018]. 43
- S. Kwon, B. Chul Kim, D. Mun, and S. Han. Graph-based simplification of feature-based three-dimensional computer-aided design models for preserving connectivity. *Journal of Computing and Information Science in Engineering*, 15(3):

- 031010–031010–14, Jun 2015. ISSN 1530-9827. doi: 10.1115/1.4030748. URL <http://dx.doi.org/10.1115/1.4030748>. 52
- C. Lardiés and Á. Cuello. *Criterios de diseño mecánico en tecnologías industriales*. Colección de textos docentes. Prensas de la Universidad de Zaragoza, 2012. ISBN 9788415538493. URL <https://books.google.com.ar/books?id=gpqs3b4Ex7oC>. 35
- B. Laurel and P. Lunenfeld. *Design Research: Methods and Perspectives*. Mediaworks Pamphlets Series. Tit Press, 2003. ISBN 9780262122634. URL <https://books.google.com.ar/books?id=xVeFdy44qMEC>. 40
- M. Layton. *Scrum For Dummies*. Wiley, 2015. ISBN 9781118905777. URL <https://books.google.com.ar/books?id=ap08BQAAQBAJ>. 43
- A. Learner. Mathematics illuminated, 2017. URL <https://www.learner.org/courses/mathilluminated/units/4/textbook/03.php>. [Internet; descargado 28-noviembre-2018]. 20
- M. L. Maher and J. S. Gero. Synchronous Support and Emergence in Collaborative CAAD. *Exchange Organizational Behavior Teaching Journal*, pages 455–470, 2006. 45, 46
- M. L. Maher and J. H. Rutherford. A model for synchronous collaborative design using CAD and database management. *Research in Engineering Design - Theory, Applications, and Concurrent Engineering*, 9(2):85–98, 1997. ISSN 09349839. doi: 10.1007/BF01596484. 44, 45
- M. Mantyla. *Introduction to Solid Modeling*. W. H. Freeman & Co., New York, NY, USA, 1988. ISBN 0-88175-108-1. 46
- A. Mardan. *Full Stack JavaScript: Learn Backbone.js, Node.js and MongoDB*. Apress, 2015. ISBN 9781484217511. URL <https://books.google.com.ar/books?id=-6xPCwAAQBAJ>. 73
- D. Marsh. *Applied Geometry for Computer Graphics and CAD*. Springer Undergraduate Mathematics Series. Springer London, 2005. ISBN 9781852338015. URL [https://books.google.com.ar/books?id=Q\\_YN2L39\\_yYC](https://books.google.com.ar/books?id=Q_YN2L39_yYC). 27, 28
- M. Masse. *REST API Design Rulebook*. O'Reilly and Associate Series. O'Reilly Media, 2011. ISBN 9781449310509. URL <https://books.google.com.ar/books?id=4lZcsRwXo6MC>. 73

- I. A. C. Matías. Capítulo 3 : Transformaciones Geométricas. pages 54–99, 2007. URL [http://catarina.udlap.mx/u\\_dl\\_a/tales/documentos/mcc/cruz\\_m\\_ia/capitulo3.pdf](http://catarina.udlap.mx/u_dl_a/tales/documentos/mcc/cruz_m_ia/capitulo3.pdf). 18
- Matthias. Blender boolean modifier for beginners, 2018. URL <https://www.blendernation.com/2018/01/23/blender-boolean-modifier-beginners/>. [Internet; descargado 28-noviembre-2018]. 33
- P. Mell, T. Grance, et al. The nist definition of cloud computing. 2011. 7
- L. Mezzalira. *Front-End Reactive Architectures: Explore the Future of the Front-End using Reactive JavaScript Frameworks and Libraries*. Apress, 2018. ISBN 9781484231791. URL <https://books.google.com.ar/books?id=1xBlswEACAAJ>. 78
- T. e. I. P. Ministerio de Ciencia. *Impresión 3D en Argentina: acciones, proyectos, actores*, volume 136. Godoy Cruz. Buenos Aires, 2015. ISBN 9789871632503. 6
- Modelo.io. Modelo.io, 2018. URL <https://modelo.io/>. 62
- J. Nieuwenhuijse, R. K. Müller, S. Baumann, Z. Dev, M. Moissette, E. Bespalov, and G. Hogdson. Openjscad.org, 2018. URL <https://openjscad.org/>. [Internet; descargado 28-noviembre-2018]. 35
- R. Nyman. The concepts of WebGL, 2013. URL <https://hacks.mozilla.org/2013/04/the-concepts-of-webgl/>. 7
- D. Olsen. *The Lean Product Playbook: How to Innovate with Minimum Viable Products and Rapid Customer Feedback*. Wiley, 2015. ISBN 9781118961025. URL <https://books.google.com.ar/books?id=j5rgCAAAQBAJ>. 56
- Onshape. Multiple section planes — what's new in onshape, 2016. URL <https://www.youtube.com/watch?v=jveEkh1NmzU>. [Internet; descargado 28-noviembre-2018]. 36
- J. Patton. Embrace Uncertainty, 2007. URL <http://www.jpattonassociates.com/brace-uncertainty/>. 42
- J. Pereiro. Diseño de productos en ISO 9001, 2005. URL [http://www.portalcalidad.com/articulos/52-diseno\\_{\\_}productos\\_{\\_}iso\\_{\\_}9001](http://www.portalcalidad.com/articulos/52-diseno_{_}productos_{_}iso_{_}9001). 39

- S. L. Pérez García. *Desarrollo de un entorno para el diseño colaborativo*. PhD thesis, 2014. URL <http://hdl.handle.net/10016/19672>. 6, 38
- R. Ramos. *Informática Gráfica*. España, 2011. 14, 15, 16, 17, 19, 21, 22, 23, 24, 25, 26, 27, 31, 48
- S. Randjelovic and S. Zivanovic. Cad-cam data transfer as a part of product life cycle. 5:87–96, 01 2007. 49
- L. Richardson and S. Ruby. *RESTful Web Services*. O'Reilly Media, 2008. ISBN 9780596554606. URL <https://books.google.com.ar/books?id=XUaErakHsoAC>. 49
- E. Ries and J. Julián. *El método Lean Startup: Cómo crear empresas de éxito utilizando la innovación continua*. Biblioteca empresarial Deusto. Deusto, 2012. ISBN 9788423409495. URL [https://books.google.com.ar/books?id=v3\\_C4yd-wR4C](https://books.google.com.ar/books?id=v3_C4yd-wR4C). 55
- L. F. Ruiz. Las nuevas herramientas para el desarrollo de proyectos intensivos en ingeniería con la colaboracion de varias empresas. *ROBOTIKER, Corporación Tecnológica TECNALIA*, NA(NA):4, 5, 2009. 37, 47
- M. Russo. *Polygonal Modeling: Basic and Advanced Techniques*. Wordware Game and Graphics Library. Jones & Bartlett Learning, 2010. ISBN 9780763798123. URL [https://books.google.com.ar/books?id=kFF-5\\_H072oC](https://books.google.com.ar/books?id=kFF-5_H072oC). 28
- E. B.-N. Sanders and P. J. Stappers. Co-creation and the new landscapes of design. *CoDesign*, 4(1):5–18, 2008. doi: 10.1080/15710880701875068. URL <https://doi.org/10.1080/15710880701875068>. 40
- L. Santaló. *Geometría proyectiva*. Manuales de Eudeba: Matemática. Editorial Universitaria de Buenos Aires, 1966. URL <https://books.google.com.ar/books?id=A33QAAAAMAAJ>. 17
- K. Schwaber. Scrum development process. In *Proceedings of the 10th Annual ACM Conference on Object Oriented Programming Systems, Languages, and Applications (OOPSLA*, pages 117–134, 1995. 42
- T. Sharon and B. Gadbaw. *Validating Product Ideas: Through Lean User Research*. Rosenfeld Media, 2016. ISBN 9781933820293. URL <https://books.google.com.ar/books?id=034kjwEACAAJ>. 57

- A. Silberschatz, H. Korth, S. Sudarshan, and F. Pérez. *Fundamentos de bases de datos*. McGraw-Hill, 2006. ISBN 9788448146443. URL <https://books.google.com.ar/books?id=ntEOPwAACAAJ>. 44
- S. Spitz and A. Rappoport. Integrated feature-based and geometric cad data exchange. In *Proceedings of the Ninth ACM Symposium on Solid Modeling and Applications*, SM '04, pages 183–190, Aire-la-Ville, Switzerland, Switzerland, 2004. Eurographics Association. ISBN 3-905673-55-X. URL <http://dl.acm.org/citation.cfm?id=1217875.1217904>. 50
- R. Stallman. FLOSS and FOSS. URL <https://www.gnu.org/philosophy/floss-and-foss.html>. 9
- R. Stallman, D. Paratcha, and L. Lessig. *Software libre para una sociedad libre*. Mapas (Madrid). Traficantes de Sueños, 2007. ISBN 9788493355517. URL <https://books.google.com.ar/books?id=ww9XAgAACAAJ>. 9
- I. E. Sutherland. Sketchpad: A man-machine graphical communication system. In *Proceedings of the May 21-23, 1963, Spring Joint Computer Conference*, AFIPS '63 (Spring), pages 329–346, New York, NY, USA, 1963. ACM. doi: 10.1145/1461551.1461591. URL <http://doi.acm.org/10.1145/1461551.1461591>. 11
- G. Tassey, S. B. Brunnermeier, and S. A. Martin. Interoperability cost analysis of the US automotive supply chain. *Research Triangle Institute Final Report. RTI Project*, 700703, 1999. 50
- K. S. Tek-Jin Nam. Collaborative 3D Workspace and Interaction Techniques for Synchronous Distributed Product Design Reviews, 2009. URL <http://www.ijdesign.org/ojs/index.php/IJDesign/article/view/387>. 44
- The Bartlett. Speckle, 2015. URL <https://speckle.works>. 61
- J. C. Torres. 4<sup>a</sup> Curso Ingeniería Informática. (December), 2014. 18, 19, 26
- M. Ungerer and K. Buchanan. Usage Guide for the STEP PDM Schema Release 4.3. *PDM Implementor Forum, Darmstadt*, (January), 2002. URL [https://www.cax-if.org/documents/pdmug\\_release4\\_3.pdf](https://www.cax-if.org/documents/pdmug_release4_3.pdf). 48
- D. Villamarin. Estado del Arte, Herramientas y Aplicaciones para Transformaciones geométricas 3D. 2015. 16

- M. Waerner. 3D Graphics Technologies for Web Applications: An Evaluation from the Perspective of a Real World Application. 2012. 7
- B. Wassermann, T. Bog, S. Kollmannsberger, and E. Rank. A design-through-analysis approach using the finite cell method. *Proceedings of the VII European Congress on Computational Methods in Applied Sciences and Engineering (ECCOMAS Congress 2016)*, 2016. doi: 10.7712/100016.1984.8920. URL <http://dx.doi.org/10.7712/100016.1984.8920>. 30
- P. R. Wilson. STEP and EXPRESS, 1998. URL <http://deslab.mit.edu/DesignLab/dicpm/step.html>. 47, 49
- Q. Zhou. PyMesh. Mesh Boolean, 2018. URL [https://pymesh.readthedocs.io/en/latest/mesh{\\_}boolean.html](https://pymesh.readthedocs.io/en/latest/mesh{_}boolean.html). 25