



**UNIVERSIDAD TÉCNICA ESTATAL DE QUEVEDO**  
**FACULTAD CIENCIAS DE LA INGENIERÍA**  
**CARRERA DE INGENIERÍA EN SISTEMAS**

Proyecto de Investigación previo a  
la obtención del título de Ingeniero  
en Sistemas.

**Título del Proyecto de Investigación:**

**"LIBRERÍA JAVASCRIPT PARA DETECTAR ERRORES EN LA ESCRITURA DE  
LOS CASOS DE USO ESCRITOS EN UN LENGUAJE DE SÍMBOLOS"**

**Autor:**

Dúval Ricardo Carvajal Suárez

**Director del Proyecto de Investigación:**

Ing. Gleiston Ciceron Guerrero Ulloa Msc.

**QUEVEDO - LOS RIOS - ECUADOR**

**2022**





## **DECLARACIÓN DE AUDITORIA Y CESIÓN DE DERECHOS**

Yo, **Dúval Ricardo Carvajal Suárez** declaro que la investigación aquí descrita es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

La Universidad Técnica Estatal de Quevedo, puede hacer uso de los derechos correspondientes a este documento, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento por la normativa institucional vigente.

---

**Dúval Ricardo Carvajal Suárez**

**070638894-9**



## **CERTIFICACIÓN DE CULMINACIÓN DEL PROYECTO DE INVESTIGACIÓN**

El suscrito, Ing. Gleiston Ciceron Guerrero Ulloa Msc. docente de la Universidad Técnica Estatal de Quevedo, certifica que el estudiante Dúval Ricardo Carvajal Suárez, realizó el Proyecto de Investigación de grado titulado “LIBRERÍA JAVASCRIPT QUE PERMITA INTERPRETAR LA ESCRITURA DE LOS CASOS DE USO EXTENDIDOS”, previo a la obtención del título de Ingeniero en Sistemas, bajo mi dirección, habiendo cumplido con las disposiciones reglamentarias establecidas para el efecto.

---

**Ing. Gleiston Ciceron Guerrero Ulloa Msc.**  
**DIRECTOR DEL PROYECTO DE INVESTIGACIÓN**



## **CERTIFICADO DEL REPORTE DE LA HERRAMIENTA DE PREVENCIÓN DE COINCIDENCIA Y/O PLAGIO ACADÉMICO**

El suscrito Ing. Gleiston Ciceron Guerrero Ulloa Msc. certifica que:

El proyecto de investigación titulado "'LIBRERÍA JAVASCRIPT QUE PERMITA INTERPRETAR LA ESCRITURA DE LOS CASOS DE USO EXTENDIDOS", ha sido analizado mediante la herramienta de URKUND y presentó resultados satisfactorios.

---

**Ing. Gleiston Ciceron Guerrero Ulloa Msc.**  
**DIRECTOR DEL PROYECTO DE INVESTIGACIÓN**



## **CERTIFICACIÓN DE APROBACIÓN DEL PROYECTO DE INVESTIGACIÓN**

**Título:**

**"LIBRERÍA JAVASCRIPT PARA INTERPRETAR LA ESCRITURA DE LOS CASOS DE  
USO EXTENDIDOS USANDO UN LENGUAJE DE SÍMBOLOS"**

Presentado al Consejo Directivo como requisito previo a la obtención del título de Ingeniero en Sistemas.

Aprobado por:

## **AGRADECIMIENTO**

Agradezco en especial a Jehova Dios por a verme dado las fuerzas e inteligencia necesario y a toda mi familia en general por a ver sido el pilar fundamental para lograr un objetivo mas en mi vida como persona.

**Duval Ricardo Carvajal Suárez**

## **DEDICATORIA**

El presente trabajo se lo dedico a todas las personas, familiares y amigos que confiaron en mi y mi esfuerzo para lograr ejercer la carrera de Ingeniería en Sistemas.

**Duval Ricardo Carvajal Suárez**



## RESUMEN Y PALABRAS CLAVES

El modelamiento de software sirve para crear una idea abstracta sobre todo lo que se quiere lograr con un sistema informático. Existen lenguajes de modelado que sirven para realizar esta tarea, pero para la mayor parte de desarrolladores esta fase en el desarrollo de software es muy tediosa por la documentación que se debe realizar antes de generar el código fuente. Uno de los diagramas mas utilizados para representar totalmente un software es el denominado diagrama de clases, el cual permite detallar todos los objetos que serán necesarios para que el producto final cumpla con las expectativas del cliente. Para tomar los requisitos del cliente existe otro tipo de diagrama que es muy útil para realizar esta tarea que son los diagramas de casos de uso, el cliente puede mencionar todo lo que el software realice, ya sea de forma verbal o escrita. El desarrollador deberá prestar atención a todo lo que el cliente necesita para su sistema, los casos de uso son los mas convenientes para redactar las necesidades o requisitos que el cliente ha mencionado. A partir de los requisitos mencionados por el usuario final que utilizará el sistema se empieza a reconocer todos los objetos que se deben generar para el software. Por eso el objetivo de este trabajo es crear una librería javascript que permita interpretar las descripciones de los casos de uso escritas con un lenguaje de símbolos que usa la herramienta TDDT4IoTS para detallar datos técnicos sobre los objetos del software, y generar la estructura de un diagrama de clases en formato JSON y XML para que luego pueda ser manipulado mediante otras librerías web que permitan dibujar otros tipos de diagramas, pero con la estructura que genera esta librería pueda conseguir el diagrama de clases de una forma mas rápida y correcta.

**Palabras claves:** generación de diagramas, detector de errores, modelado uml, desarrollo ágil, diagrama de clases

## ABSTRACT AND KEYWORDS

Software modeling is used to create an abstract idea of what you want to achieve with a computer system. There are modeling languages that are used to perform this task, but for most developers this phase in software development is very tedious because of the documentation that must be done before generating the source code. One of the most used diagrams to fully represent a software is the so-called class diagram, which allows to detail all the objects that will be necessary for the final product to meet the customer's expectations. To take the customer's requirements there is another type of diagram that is very useful to perform this task, which are the use case diagrams, the customer can mention everything that the software performs, either verbally or in writing. The developer should pay attention to everything that the customer needs for his system, the use cases are the most convenient to write the needs or requirements that the customer has mentioned. From the requirements mentioned by the end user who will use the system, we start to recognize all the objects that must be generated for the software. That is why the objective of this work is to create a javascript library that allows to interpret the descriptions of the use cases written with a symbol language that uses the TDDT4IoTS tool to detail technical data about the objects of the software, and generate the structure of a class diagram in JSON and XML format so that I can then be manipulated by other web libraries that allow to draw other types of diagrams, but with the structure generated by this library can get the class diagram in a faster and more correct way.

**Keywords:** diagram generation, error detector, uml modeling, agile development, class diagram, uml modeling

# Tabla de contenido

<b>Tabla de contenido</b>	<b>x</b>
<b>Lista de tablas</b>	<b>xiii</b>
<b>Lista de figuras</b>	<b>xiv</b>
<b>I CONTEXTUALIZACIÓN DE LA INVESTIGACIÓN</b>	<b>4</b>
1.1 Problema de investigación. . . . .	5
1.1.1 Planteamiento del problema. . . . .	5
1.1.2 Formulación del problema. . . . .	6
1.1.3 Sistematización del problema. . . . .	6
1.2 Objetivos. . . . .	6
1.2.1 Objetivo General. . . . .	6
1.2.2 Objetivos Específicos. . . . .	7
1.3 Justificación. . . . .	7
<b>II FUNDAMENTACIÓN TEÓRICA DE LA INVESTIGACIÓN</b>	<b>9</b>
2.1 Marco conceptual . . . . .	10
2.1.1 Lenguajes de marcado . . . . .	10
2.1.2 Compiladores . . . . .	11
2.1.3 Modelamiento de software . . . . .	11
2.1.4 Metamodelado . . . . .	12
2.1.5 Desarrollo ágil de software . . . . .	14
2.2 Marco referencial . . . . .	16

III	MÉTODOLÓGÍA DE LA INVESTIGACIÓN	18
3.1	Localización . . . . .	19
3.2	Tipo de investigación . . . . .	19
3.2.1	Investigación aplicada . . . . .	19
3.3	Método de investigación . . . . .	20
3.3.1	Método inductivo . . . . .	20
3.3.2	Método deductivo . . . . .	20
3.3.3	Método analítico . . . . .	20
3.4	Fuentes de recopilación de información . . . . .	20
3.4.1	Fuentes primarias . . . . .	20
3.4.2	Fuentes secundarias . . . . .	21
3.5	Diseño de la investigación . . . . .	21
3.5.1	Metodología de desarrollo aplicada al proyecto . . . . .	21
3.5.2	Modelo de prototipado . . . . .	27
3.6	Instrumentos de investigación . . . . .	28
3.7	Recursos humanos y materiales . . . . .	28
3.7.1	Talento humano . . . . .	28
3.7.2	Equipo de trabajo . . . . .	28
3.7.3	Recursos de software . . . . .	28
3.7.4	Recursos de hardware . . . . .	29
IV	RESULTADOS Y DISCUSIÓN	30
4.1	Resultados de la metodología . . . . .	31
4.1.1	Análisis de requisitos y obtención de pruebas . . . . .	31
4.2	Modelamiento . . . . .	43
4.3	Generación de código . . . . .	49
4.4	Ejecución de pruebas . . . . .	53
4.5	Evaluación con sistemas de información . . . . .	58
V	CONCLUSIONES Y RECOMENDACIONES	59
5.1	Conclusiones . . . . .	60
5.2	Recomendaciones . . . . .	61

VI	BIBLIOGRAFÍA	62
VII	ANEXOS	68

# Lista de tablas

3.1	Los 12 principios ágiles y su análisis en la metodología. . . . .	22
3.2	Recursos de software utilizados en el proyecto . . . . .	29
3.3	Recursos de hardware utilizados en el proyecto . . . . .	29
4.4	Código ASCII para cada simbolo del lenguaje. . . . .	38
4.5	Código ASCII para cada simbolo del lenguaje que especifican las relaciones.	39
4.6	Descripción del caso de uso para Tutor registration. . . . .	40
4.7	Descripción del caso de uso para Tutor registration. . . . .	41
4.8	Descripción del caso de uso para ingresar la descripción. . . . .	44
4.9	Descripción del caso de uso para interpretar la descripción. . . . .	45
4.10	Descripción del caso de uso para validar símbolos. . . . .	46
4.11	Descripción del caso de uso para identificar errores. . . . .	47
4.12	Descripción del caso de uso para generar estructura json y xml. . . . .	48
4.13	Descripción del caso de uso para iniciar sesión. . . . .	58

# Lista de figuras

2.1	Estructura json. . . . .	10
2.2	Estructura xml. . . . .	11
2.3	La estructura y los con-tentos de un FRP asociado a un ATP. . . . .	13
2.4	El metamodelo SoPaMM. . . . .	15
3.5	Localización de la UTEQ campus "La María". . . . .	19
3.6	Fases de la metodología . . . . .	23
4.7	Símbolos que utiliza la herramienta TDDT4IoTS . . . . .	31
4.8	Ejemplo de como se debe utilizar el símbolo respectivo para crear una clase.	32
4.9	Ejemplo de como se debe utilizar el símbolo respectivo generar los parámetros de un método declarado con el lenguaje de símbolos. . . . .	32
4.10	Diagrama de casos de uso para las pruebas. . . . .	39
4.11	Diagrama de clases de prueba . . . . .	41
4.12	Diagrama de casos de uso armadillo. . . . .	43
4.13	Diagrama de flujo para detectar errores en las descripciones de los casos de uso utilizando un el lenguaje de símbolos. . . . .	49
4.14	Diagrama de clases generado por la aplicación web de demostración usando la librería de armadillo.js . . . . .	57
6.15	Interfaz principal de aplicación demostrativa de la librería. . . . .	69
6.16	Interfaz donde se ingresaran las descripciones de los casos de uso a interpretar.	69
6.17	Descripción del caso de uso interpretada con su respectiva retroalimentación.	70
6.18	Diagrama de clases generado por la librería usando una librería externa denominada jsUML2. . . . .	70
6.19	Estructura JSON generada por la librería. . . . .	71
6.20	XML generado por la librería. . . . .	71
6.21	Error detectado en la descripción del caso de uso . . . . .	72

# Introducción

Los sistemas informáticos se están volviendo cada vez más indispensables en todos los aspectos de la vida cotidiana. Considerando que la tecnología cambia aceleradamente, y que los usuarios requieren soluciones rápidas, son las principales razones por las que la Ingeniería de Software busca disminuir el tiempo en implementar los sistemas requeridos [1]. Para la construcción de una aplicación informática que cumpla con los requisitos del cliente y que sea eficiente se necesitan aplicar varios aspectos técnicos [2], que la Ingeniería de Software detalla. Por ejemplo: estilo arquitectural, tecnologías a utilizar, infraestructura, escalabilidad, entre otros [3].

El campo de la Ingeniería de Software es análogo al campo de la construcción de obras civiles. Los documentos de diseño de un Software guardan semejanza con los documentos de diseño de un edificio (por ejemplo). Para la construcción de un edificio se deben analizar las principales características físicas del terreno sobre el cual se va a construir, mientras que, para la construcción del software se debe analizar la/s plataformas sobre las cuales se ejecutará el software. En el caso de un edificio, se deben detallar aspectos de la construcción ambientes y sus dimensiones precisas, columnas, puertas, ventanas, etc., así mismo se necesita documentar con precisión los módulos y las funcionalidades y alcance de cada uno de ellos, los elementos que conformarán el producto de software a desarrollar, además del comportamiento que deben tener todos los artefactos de un software [4]. Lo que se puede mencionar como diferencias que estos elementos de software pueden ser contruidos para ser reutilizados en otros sistemas [5].

*Unified Modeling Language* (UML) con su traducción al español Lenguaje Unificado de Modelado, Uno de los lenguajes más conocidos por los desarrolladores de software para la representación de los requisitos o necesidades del cliente/usuario final (en adelante solamente usuario) son los diagramas UML [6]. Sin embargo, estos diagramas no son fáciles de comprender para todos los usuarios, ya que, los elementos que intervienen y que darán solución al problema planteado son representados mediante símbolos y algo de texto. UML se utiliza específicamente en la industria del software para especificar, visualizar, construir y documentar los artefactos de un sistema de software [7]. UML se encuentra definido oficialmente por el *Object Management Group* (OMG) con su traducción al español Grupo de Administración de Objetos [8]. Algunos investigadores afirman que, a partir de modelos



dinámicos (diagramas de casos de uso) y estáticos (diagramas de clases) se pueden generar otros modelos UML de manera más eficiente [9], y así obtener software de calidad.

Los estándares ISO/IEC 9126-1, especifican la calidad del software tanto interna, externa y en uso del producto. De ahí la importancia de considerar seguir estándares que permitan asegurar un lenguaje común para el entendimiento de todos los involucrados en el proyecto de software [10].

Los diagramas de casos de uso son una representación del sistema mediante los usuarios (actores) y sus requisitos (casos de uso). Son una buena herramienta para representar los requisitos iniciales del sistema, y pueden ser fácilmente entendidos por todos los involucrados [11]. Sin embargo en un diagrama de caso de uso no se especifican los detalles sino que son los requisitos a *grano grueso* del sistema. La información de cada caso de uso se detalla mediante condiciones previas (precondiciones), poscondiciones y secuencia de eventos normales. Además, incluye la secuencia alternativa de eventos en caso de excepciones o condiciones específicas y las postcondiciones que se deben tomar en cuenta al momento de estar utilizando el software [12]. Estas especificaciones de los casos de uso se denominan los requisitos a *grano fino*, y se plasman en documentos denominados *casos de uso extendidos*. Estos requisitos de grano fino pueden ser comprimidos en los diagramas de clases [13].

Los diagramas de clases en UML representan la estructura estática de los objetos y sus posibles conexiones dentro del software. Se lo utiliza para ilustrar el punto de vista estático, exponiendo un conjunto de clases, interfaces y relaciones [14]. Se lo desarrolla durante la fase de elaboración y se perfecciona posteriormente en la fase de construcción [8], representando un modelo del dominio del sistema. Además, es uno de los diagramas más útiles en UML, ya que trazan claramente la estructura de un sistema concreto al modelar sus clases con sus atributos y operaciones, y las relaciones entre clases [14].

Los diagramas de clases son utilizados por muchas herramientas (Rational Rose<sup>1</sup>, MagicDraw<sup>2</sup>, ArgoUML<sup>3</sup>, por mencionar algunas) para la generación del código de software a partir de los diagramas de clases, sin embargo ninguna considera la documentación de los casos de uso extendidos, por lo tanto, tampoco utilizan las especificaciones a grano fino para obtener diagramas UML que ayudan a generar el código como lo es el diagrama de clases.

---

<sup>1</sup><https://www.ibm.com/docs/es/rsas/7.5.0?topic=migration-rational-rose-model>

<sup>2</sup><https://www.magicdraw.com/>

<sup>3</sup><https://argouml-tigris-org.github.io/>

Existe una herramienta TDDT4IoTS [15] que está en fase de prueba. En esta herramienta es posible que el analista escriba los requisitos detallados de los casos de uso, y, mediante el uso de un lenguaje de símbolos (SymLen) marcar las palabras que representan nombres de clases, atributos, métodos, interfaces y otros elementos de los diagramas de clases. A su vez, presenta los casos de uso extendidos en lenguaje natural exactamente como el usuario expresó los requisitos detallados que el sistema debe cumplir. De estas especificaciones de grano fino es posible generar los diagramas de clases, y el código de software correspondiente. Al ser una herramienta en su fase inicial, el interprete implementado en TDDT4IoTS no presenta una retroalimentación eficiente, simplemente presenta los artefactos resultado, sin especificar los posibles errores en el uso del lenguaje. Además el actual intérprete no se puede utilizar fácilmente en otra herramienta.

En el presente trabajo se propone una librería escrita en JavaScript denominada *Armadillo* (Armadillo.js). Armadillo permite interpretar las descripciones de los casos de uso extendidos escritas en SymLen. Armadillo le permite al analista conocer los posibles errores en el uso de SymLen, con el objetivo de obtener de forma automática un diagrama de clases pertinente a la información proporcionada por el usuario. Esto permitirá potenciar el uso de la herramienta TDDT4IoTS. Además, Armadillo le permitirá eliminar todos los inconvenientes del intérprete que actualmente utiliza TDDT4IoTS que han sido detectados.

El resto de este documento está organizado por capítulos. En el capítulo I se contextualiza la investigación, donde se especifica la problemática a resolver, los objetivos que se lograron y la hipótesis que se demostró, además de una argumentación que justifica que se haya realizado esta investigación. En el capítulo II se presenta la fundamentación teórica, en la que se presentan los principales trabajos relacionados con el trabajo propuesto en este documento, conceptos y definiciones importantes, y las limitaciones, herramientas y tecnologías se expresan en el marco contextual de la investigación.

**CAPÍTULO I**  
**CONTEXTUALIZACIÓN DE LA INVESTIGACIÓN**

## **1.1. Problema de investigación.**

### **1.1.1. Planteamiento del problema.**

Con el aumento en la complejidad de los productos de software los desarrolladores de sistemas informáticos han encontrado la manera de mejorar el desarrollo de software, mediante el uso del modelado UML [9]. El modelado de software permite a los desarrolladores comprender todo el diseño de software, obteniendo como resultado una visión general del sistema y una herramienta de comunicación con otros desarrolladores [16].

Para empezar con el modelado de un software se necesitan los requisitos planteados por el cliente. Un estudio reveló que el 95% de los documentos de requisitos de un sistema estaban redactados en algún tipo de lenguaje natural [9]. Todos los requisitos planteados son plasmados en casos de uso, siendo la herramienta de modelado más habitualmente utilizada para representar las especificaciones del software [17]. Estos casos de uso permiten analizar más a fondo de forma privada las necesidades del cliente. Sin embargo, al realizar esto suelen surgir confusiones entre el cliente y el desarrollador, debido a que los casos de uso podrían ser modificados, detallando características técnicas del software que el cliente no logrará comprender.

El desacuerdo que existe entre el cliente y el desarrollador podría llevar a resultados poco favorables para ambos. Las descripciones de los casos de uso modificados por el desarrollador podrán ser útiles para construir uno de los diagramas más populares del modelado UML como lo es el diagrama de clases [13]. Pero, para el cliente los casos de uso no tendrán sentido lógico respecto a las condiciones previas dictadas por él.

TDDT4IoTS es una herramienta que usa un lenguaje de símbolos para escribir las descripciones de los casos de uso, permitiendo detallar datos técnicos sobre las clases, interfaces, métodos, etc. que formaran parte del sistema informático a desarrollar. Esta herramienta no cuenta con algún mecanismo que permita notificar si se está utilizando de manera correcta los símbolos respectivos. Además, solo permite generar una estructura JSON del diagrama de clases personalizada, sin tomar en cuenta la posibilidad de generar una estructura que pueda ser comprendida por otras herramientas que realizan el modelado con UML.

Sería importante proveer a los desarrolladores de software o usuarios que utilicen la herramienta TDDT4IoTS, una tecnología que permita detectar los errores cometidos al momento de usar

el lenguaje de símbolos para crear los casos de uso extendidos. Además, sería interesante generar una estructura del diagrama de clases pertinente a los casos de uso y pueda ser utilizado por otras herramientas de modelado UML.

#### **Diagnóstico.**

...

#### **Pronóstico.**

...

### **1.1.2. Formulación del problema.**

¿Es posible detectar los errores de escritura en los casos de uso extendidos, escritos en el lenguaje de símbolos utilizado en la herramienta TDDT4IoTS para generar un diagrama de clases?

### **1.1.3. Sistematización del problema.**

1. ¿Qué estructura de dato permitirá a otras librerías que generar diagramas modificar el diagrama de clases generado por la librería propuesta?
2. ¿Se puede notificar al desarrollador sobre la incorrecta escritura de los casos de uso redactados con el lenguaje de símbolos?
3. ¿Cómo determinar el nivel de efectividad del producto de este proyecto de investigación respecto a la generación del diagrama de clases?

## **1.2. Objetivos.**

La problematización de este proyecto ha llevado a plantearse los siguiente objetivos.

### **1.2.1. Objetivo General.**

Desarrollar una librería JavaScript que interprete los casos de uso escritos con el lenguaje de símbolos usado en la herramienta TDDT4IoTS, detectando los errores de escritura para generar un diagrama de clases.

### **1.2.2. Objetivos Específicos.**

1. Generar un estructura en formato JSON y XML para almacenar el diagrama de clases, y pueda ser visualizado mediante otras librerías que generen diagramas.
2. Diseñar e implementar una manera de retroalimentar a los usuarios de TDDT4IoTS sobre la incorrecta escritura de los casos de uso redactados con el lenguaje de símbolos.
3. Evaluar la librería JavaScript propuesta, con la elaboración de diagramas de clases a partir de casos de uso extendidos correspondientes a requisitos de sistemas de información.

### **1.3. Justificación.**

Los requisitos planteados por el cliente no siempre estarán claros desde el principio [12]. Al momento de comenzar la fase de desarrollo de un sistema informático, pueden surgir problemas que deberán ser resueltos de diferentes formas a las que fueron planteadas al inicio. Existen herramientas que permiten crear los diagramas de casos de uso transmitiendo de forma gráfica los requisitos que se deben ejecutar, pero no detallan toda la información requerida por un solo caso de uso [13].

Toda la información de un caso de uso detalla las condiciones previas, precondiciones y secuencia de eventos que deberá seguir un sistema informático [12]. Es decir, toda esa información permitirá generar otros tipos de diagramas usando el modelado UML. Sin embargo, las herramientas existentes hasta el momento permiten generar varios tipos de diagramas UML, pero es necesario para poder crearlos o modificarlos, construirlos por separado y de manera manual.

Una solución para mejorar el rendimiento de los desarrolladores es utilizar un lenguaje para la escritura de los casos de uso que permita escribir toda la información necesaria de un caso de uso y al mismo tiempo permita detallar información técnica del sistema informático. Se podrá obtener todo lo necesario para crear un diagrama de casos de uso y generar de forma automática la estructura del diagrama de clases pertinente a los datos técnicos detallados con la ayuda del lenguaje de símbolos usando por la herramienta TDDT4IoTS.

Para los usuarios de la herramienta TDDT4IoTS será muy beneficioso contar con una librería que le permita interpretar para detectar si la escritura de la información es correcta. Además,

el producto final de este trabajo permitirá modificar los casos de uso y al mismo tiempo poder hacerlo con los datos técnicos referentes al diagrama de clases, tratando de reducir el tiempo en el refinamiento de modelos del software. Finalmente se puede mencionar que no se ha encontrado aplicaciones o algún tipo de software que permita escribir los casos de uso, haciendo posible la generación de otros tipos de diagramas.

**CAPÍTULO II**  
**FUNDAMENTACIÓN TEÓRICA DE LA INVESTIGACIÓN**



## FUNDAMENTACIÓN TEÓRICA

En este capítulo, se expondrán los trabajos relacionados con el presentado en este documento que se han identificado en la literatura, y que demuestran que el trabajo propuesto en este documento, aporta algo. Además se contextualiza el trabajo y define los términos novedosos y de poco dominio para los investigadores y para la comunidad.

### 2.1. Marco conceptual

En esta sección, se detalla los modelos teóricos, conceptos, argumentos o definiciones que se han desarrollado o investigado en relación con el tema en particular.

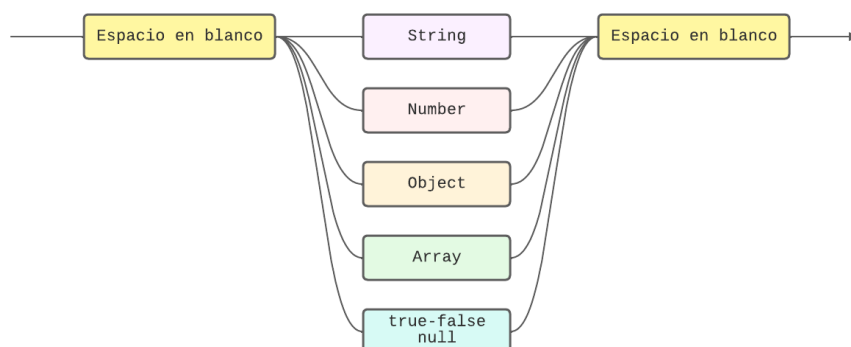
#### 2.1.1. Lenguajes de marcado

A continuación se describen una serie de formatos de texto utilizados para el intercambio de datos entre varias aplicaciones.

##### 2.1.1.1. JSON

Javascript Object Notation (JSON) es un formato ligero de intercambio de datos. Consisten en asociación de nombres y valores. A pesar de ser independiente del lenguaje de programación, es admitido en una gran cantidad de lenguajes de programación. Se basa en un subconjunto del Estándar de lenguaje de programación JavaScript [29].

Figura 2.1  
*Estructura json.*

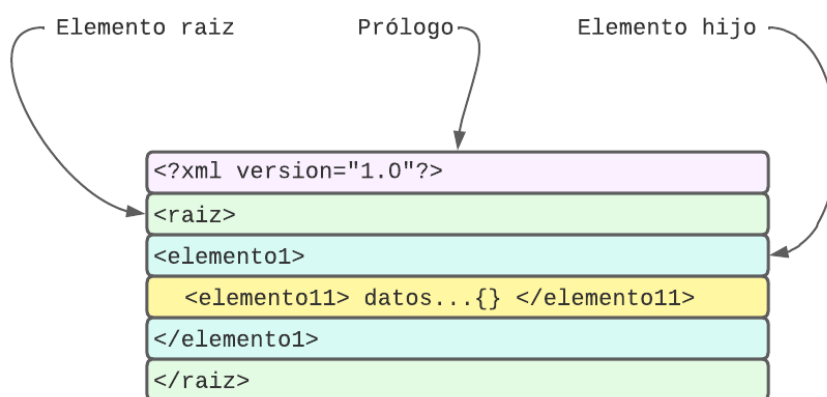


**FUENTE: PROPIA**  
**ELABORADO: DÚVAL CARVAJAL SUÁREZ**

### 2.1.1.2. XML

Es un lenguaje de marcado similar a HTML. Significa Extensible Markup Language y pertenece a la especificación W3C como lenguaje de marcado de propósito general. Esto significa que, a diferencia de otros lenguajes de marcado, XML no está predefinido, por lo que debe definir su propio marcado. El objetivo principal del lenguaje es compartir datos entre diferentes sistemas, como Internet [30].

Figura 2.2  
*Estructura xml.*



**FUENTE: PROPIA**  
**ELABORADO: DÚVAL CARVAJAL SUÁREZ**

### 2.1.2. Compiladores

Están diseñados para traducir un fragmento de código escrito en un lenguaje de programación a lenguaje de máquina que es, el que puede entender la computadora. El compilador analiza el código fuente en busca de errores antes de la traducción. Si se detecta un error, el compilador notificar al autor del código para que pueda arreglarlo.. Además, los compiladores modernos o entornos de desarrollo (IDE), puede sugerir soluciones para algunos tipos de errores usando métodos de corrección de errores [31].

### 2.1.3. Modelamiento de software

Representan una serie de requisitos basados en la construcción de elementos visuales para definir estructuras y comportamientos que tendrá el software. UML (Lenguaje Unificado de

Modelado) a través del mecanismo de perfilado, se han basado históricamente en notaciones gráficas. UML mediante el mecanismo de perfiles, maximiza la comprensión humana y facilita la comunicación entre las partes interesadas como son el cliente y desarrollador [32].

También existen lenguajes de modelado personalizados para distintas áreas, como por ejemplo en [33] proponen un lenguaje de modelado conceptual multinivel al denominan ML2 (Lenguaje de Modelado Multinivel). El lenguaje está orientado al modelado conceptual multinivel (de dominio) y pretende cubrir un amplio conjunto de dominios multiniveles. En el diseño de ML2 sigue un enfoque basado en principios, definiendo su sintaxis abstracta para reflejar una teoría formal para el modelado multinivel que se fue desarrollado previamente.

#### 2.1.4. Metamodelado

Existen varias formas de realizar el metamodelado de un software. En [24] mencionan el *Software Pattern MetaModel* (SoPaMM) o en su traducción al español Metamodelo del patrón de software. El objetivo principal de este proceso es la especificación de patrones de requisitos funcionales (FRP) vinculados a patrones de pruebas de aceptación (ATP). En la figura 2.3 se observa los componentes más importantes de un FRP relacionado con un ATP.

Basada en metodología ágil BDD, FRP es una estructura inspirada en la descripción de las historias de usuario. A continuación se detalla la estructura de un FRP [24]:

- **As:** Describir la parte que se beneficia de la característica.
- **I\_can:** La característica en sí.
- **So\_that:** El valor agregado de la característica.
- **Given:** Describe, en una o más cláusulas, el contexto inicial del escenario.
- **When:** Describe los eventos que desencadenan un escenario.
- **Then:** Describe, en una o más cláusulas, los resultados esperados del escenario.

Como se observa en la figura 2.4 FRP\_User\_Creation describe una parte de la función para crear usuarios. Se puede visualizar que el usuario administrador es el actor beneficiado de esta función para que se logre registrar un nuevo usuario al sistema. El intento de crear un usuario conlleva a dos posibles escenarios: uno exitoso y otro no exitoso, pero ambos

Figura 2.3

*La estructura y los contenidos de un FRP asociado a un ATP.*

<b><u>FUNCTIONAL REQUIREMENT PATTERN</u></b>
<b>Name:</b> FRP_User_Creation
<b>Problem:</b> The user must be registered for the system
<b>Forces:</b> Ensuring that the user is successfully created.
<b>FEATURE USER CREATION</b>
<b>As:</b> an administrator
<b>I_can:</b> create a new user
<b>So_that:</b> he/she is registered for the system
<b>SCENARIO SUCCESSFUL USER CREATION</b>
<b>Given:</b> I am trying to create a new user
<b>When:</b> I enter <ITRN>, <name>, <gender>, <date of birth>, <father's name>, <mother's name> and <role>
<b>Then:</b> the system should register a new user with these data
<b>EXAMPLE</b>
735.101.320-92   Carlos Chagas   Male   01.01.2000   Jose Chagas   Carla Chagas   Doctor
342.052.020-40   Jose Mouro Brasil   Male   01.01.2000   Jose Brasil   Josefa Brasil   Ophthalmologist Doctor
<b>SCENARIO NOT SUCCESSFUL USER CREATION - INVALID ITRN</b>
<b>Given:</b> I am trying to create a new user
<b>When:</b> I enter an invalid <ITRN>
<b>Then:</b> The system should display the <message> error message
<b>EXAMPLE</b>
735.101.320-00   "This is an invalid individual taxpayer registration number!"
342.052.020-00   "This is an invalid individual taxpayer registration number!"
<b><u>ACCEPTANCE TEST PATTERN</u></b>
<b>Name:</b> ATP_User_Creation
<b>Problem:</b> The user creation feature must be tested against its scenarios and respective example data.
<b>Forces:</b> Ensuring that all user creation scenarios are successfully tested.
<b>Test Case SUCCESSFUL USER CREATION</b>
InputData ← <b>Example of Scenario SUCCESSFUL USER CREATION</b>
OutputData ← <b>Example of Scenario SUCCESSFUL USER CREATION</b>
<b>Test Case NOT SUCCESSFUL USER CREATION - INVALID ITRN</b>
InputData ← <b>Example of Scenario NOT SUCCESSFUL USER CREATION - INVALID ITRN</b>
OutputData ← <b>Example of Scenario NOT SUCCESSFUL USER CREATION - INVALID ITRN</b>

**FUENTE:** [24]

**ELABORADO:** DÚVAL CARVAJAL SUÁREZ

vinculados a una misma precondition. Sin embargo, dependiendo de la validez de los usuarios la ejecución puede ser distinta. Del mismo modo por cada escenario se representara un resultado diferente, es decir se visualizaran mensajes diferentes al momento de registrar un usuario o mensajes de error [24].

Finalmente el ejemplo mostrado permite definir y vincular varios datos cada escenario. En cada escenario se consta con dos instancias de datos, de manera que uno de los escenarios registrado un nuevo usuario con éxito, mientras que el otro no lo hace debido a que los datos de entrada no son validos, como es el numero de identificación del usuario. Esta representación define el enfoque de patrones de requisitos funcionales basado en la ejecución de los posibles escenarios. En la figura 2.4 se observa el metamodelo de SoPaMM [24].

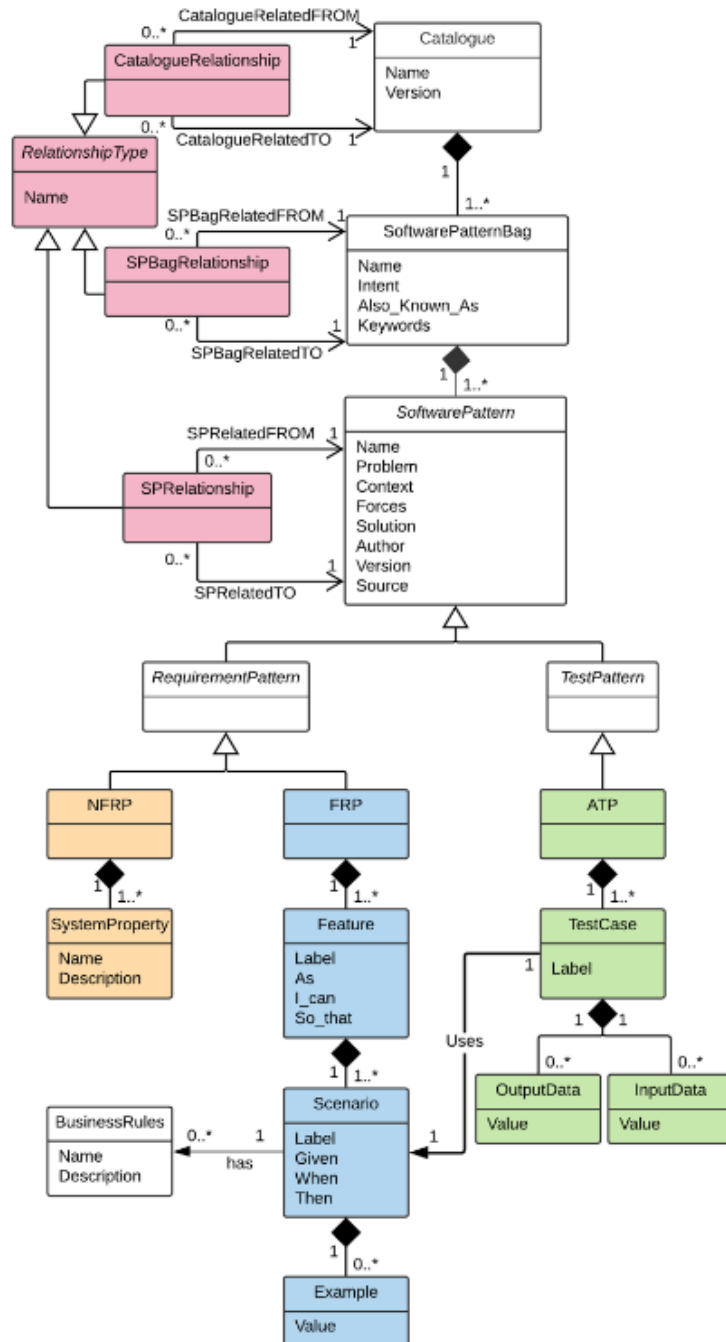
### 2.1.5. Desarrollo ágil de software

El desarrollo ágil de software hace referencia a los principios del manifiesto ágil que son aplicadas por varias metodologías de desarrollo. Las metodologías de desarrollo ágil comparten varias características como: la entrega frecuente de software de trabajo (desarrollo iterativo), la velocidad constante y la comunicación abierta [28]. En el desarrollo ágil los métodos que se emplean pueden ser de bajo costo, es decir que cualquier modificación que se realice en alguna etapa de desarrollo el cliente podrá observar los resultados deseados referente al costo que ha pagado [34].

Ademas de ser una buena alternativa para mejorar el desarrollo de software, el desarrollo ágil es muy productivo por que brecha de comunicación entre los clientes y los desarrolladores de software. Conforme avance el tiempo, difícilmente pueden surgir motivos para generar retrasos. Lo beneficioso de aplicar este proceso es que los cambios pueden ser acoplados continuamente de acuerdo a las necesidades del cliente. A continuación se mencionaran varias metodologías ágiles que se utilizan en el desarrollo de software [34]:

- **La programación extrema (XP)** : Esta es una metodología centrada específicamente en obtener un software de calidad. Para cumplir con ese objetivo, se debe considerar en primer lugar una buena comunicación verbal entre todos los integrantes del proyecto. Ademas no existen reglas que obliguen a llevar una documentación rigurosa, con el fin de aumentar la eficiencia de desarrollo del software.

Figura 2.4  
El metamodelo SoPaMM.



FUENTE: [24]  
ELABORADO: DÚVAL CARVAJAL SUÁREZ

- **Dynamic Systems Development Method (DSDM):** En DSDM o en su traducción al español Sistemas dinámicos Método de desarrollo la documentación es totalmente necesaria, debe ser magra y oportuna describiendo paso a paso todo el proceso que se realizó. Esto significa que el modelo y el prototipo deberán representar una visión

general sobre lo que va hacer el sistema. Además se deberán documentar los diagramas de modelado, estructuras físicas de datos y decisiones de diseño.

- **Scrum:** Se centra netamente en la versatilidad rentabilidad y adaptabilidad. Le permite al desarrollador tomar la decisión para elegir la técnica de ejecución del modulo que este encargado. La administración de Scrum permite reconocer las carencias u obstáculos que se presentaran a lo largo del proyecto. Los equipos de Scrum tiene la libertad de elegir y reconocer los enfoques para realizar su trabajo, es decir que cada integrante del grupo cuanta con las habilidades necesarias para cumplir su trabajo sin necesidad de recurrir a pedir ayuda fuera del grupo.

## 2.2. Marco referencial

El diagrama de clases sin duda es el artefacto más importante para modelar un sistema y el punto de partida para otros diagramas [18]. La problemática de la obtención de diagramas de clases a partir de los casos de uso detallados no ha sido tratada a profundidad. Por lo tanto los investigadores se encuentran en un campo fértil de investigación.

Las soluciones que se han propuesto y que se han recuperado en este trabajo, utilizan el procesamiento del lenguaje natural (NLP por sus siglas en inglés *Natural Language Process*). Entre las soluciones encontradas podemos mencionar el trabajo de Chen y Zeng [19], en el cual presentan una aproximación sobre la obtención del diagrama de casos de uso y el diagrama de clases UML a partir de los requisitos del producto expresados en lenguaje natural. Sin embargo, en este trabajo intentan analizar el texto para determinar el conjunto de palabras significativas que pueden representar los elementos de los diagramas, por ejemplo, para una clase un sustantivo, para un método un verbo nominal para los métodos, la conexión entre dos objetos (clases) expresarían una relación. El trabajo de Chen & Zeng [19] tiene buenos resultados con pocos y bien establecidos requisitos de software, que pueden animar a los investigadores a seguir mejorando esa herramienta, como por ejemplo, identificar los diferentes tipos de relaciones entre clases, y su representación en el diagrama UML.

El trabajo presentado por Dawood Omer & Eltyeb [20] es otra de las soluciones propuestas. En [20] proponen un modelo de razonamiento que basado en casos puede facilitar el proceso de generación de diagramas de clases UML a partir de requisitos textuales. Para ello utilizan técnicas de minería de texto. Por lo tanto el trabajo es bastante abrumador: primero se deben

tener una base de casos para entrenar el modelo, luego se debe entrenar el modelo. Además, aunque las diferencias sean muy pequeñas, los resultados (diagrama de clases UML) puede variar.

En la misma línea de la aplicación de procesamiento del lenguaje natural para la obtención del diagrama de clases a partir de la descripción textual de los requisitos, podemos citar el trabajo de Abdelkareem M. Alashqar [21], en el que, para lograr este objetivo proponen un algoritmo y una herramienta. El algoritmo consiste básicamente en la separación de la oración en cada palabra que la conforman para luego aplicar un análisis morfológico a cada palabra, según el tipo de palabra (sustantivo, verbos en diferentes voces o tiempos, etc.) determinar los diferentes elementos de los diagramas. En este caso, la herramienta que implementan para aplicar el algoritmo muestra las oraciones que han sido mal escritas. Alshgar en su trabajo [21] especifica que, para el algoritmo funcione correctamente, los requisitos textuales deben estar escritos con ciertas restricciones, como por ejemplo: cada oración debe estar escrita en una línea (separadas con un nueva línea), y que cada oración represente una acción a realizar con el software, entre otras restricciones.

De los trabajos que se revisado hay que recalcar que [21] y [22] se preocupan por el análisis de oraciones (acciones) pasivas negativas. Sin embargo todos los trabajos que utilizan NLP están limitados al idioma en el que están escritos los requisitos, y al uso correcto de la gramática y todos consideran restricciones en la escritura. Por lo que se presenta a Armadillo como una librería que ayuda que SymLen sea un lenguaje a utilizar para la escritura de los casos de uso detallados sin importar el idioma, y sin ninguna restricción. Con Armadillo la eficiencia de los diagramas de clases y la generación de código de software que satisface los requisitos del usuario, depende estrictamente del uso correcto de SymLen, y/o de la corrección de los posibles errores de su uso que Armadillo muestre al analista.



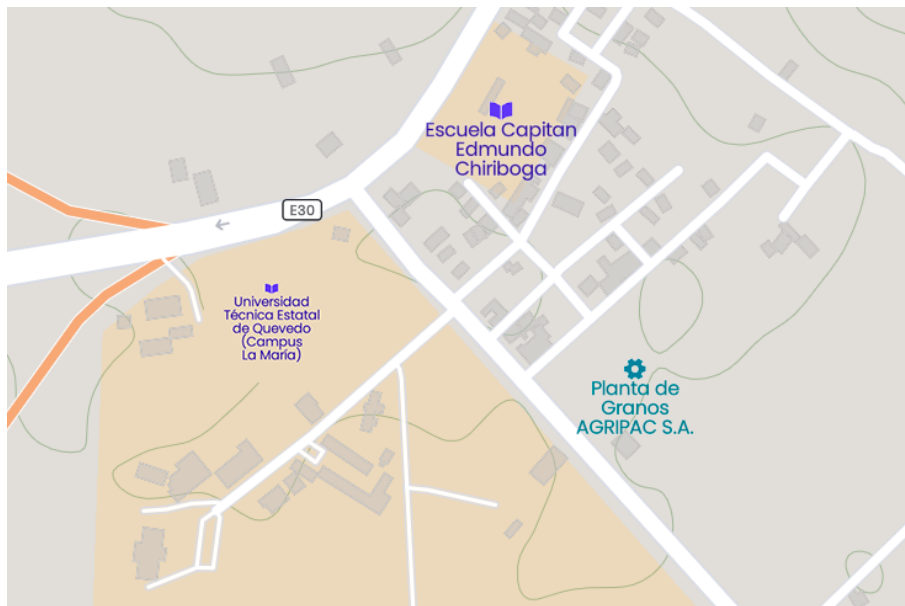
**CAPÍTULO III**  
**MÉTODOLOGÍA DE LA INVESTIGACIÓN**

### 3.1. Localización

El proyecto se realizó en la Universidad Técnica Estatal de Quevedo (UTEQ), campus "La María", ubicada en el Cantón Quevedo de la Provincia Los Ríos, en Ecuador. Previo a la obtención del título de Ingeniero en Sistemas. La ejecución del proyecto duró 4 meses, desde el mes de junio de 2022 hasta el mes de septiembre del 2022.

Figura 3.5

*Localización de la UTEQ campus "La María".*



**ELABORADO: DÚVAL CARVAJAL SUÁREZ**

### 3.2. Tipo de investigación

#### 3.2.1. Investigación aplicada

La ejecución de este proyecto implicó realizar una investigación de tipo aplicada. Se analizarán conceptos como lo que son los compiladores, lenguajes de modelado UML y detección de errores. El trabajo presentado fue un proyecto de desarrollo, es decir que se aplicaron los conceptos investigados al producto final para que cumpla con todo lo establecido en apartado inicial de este trabajo.

### **3.3. Método de investigación**

#### **3.3.1. Método inductivo**

Con la ayuda del método inductivo se pudo conocer que para los desarrolladores de software es necesario contar con algún tipo de conocimiento sobre lenguajes de modelado, llevar a un orden de trabajo mediante una metodología de desarrollo ágil que permita satisfacer las necesidades y requisitos que fueron planteados por el cliente.

#### **3.3.2. Método deductivo**

El método deductivo permitió emplear de la problemática del proyecto presentado, el cual se evidencio la manera de cómo obtener un diagrama de clases de forma automática mediante las descripciones de los casos de uso. Además de potenciar el uso de la herramienta TDDT4IoTS. El objetivo de esta librería es disminuir el tiempo que le toma a un desarrollador empezar con el modelado un sistema informático empezando por la generación del diagrama de clases.

#### **3.3.3. Método analítico**

Se empleo el método analítico para extraer los símbolos que serán analizados por la librería, con el objetivo de conocer que símbolos serán los que se lograrán identificar en las descripciones de los casos de uso. En este caso se debe conocer exactamente los datos de entrada para ser procesados y generar lo requerido por el proyecto presentado.

### **3.4. Fuentes de recopilación de información**

#### **3.4.1. Fuentes primarias**

Entre las fuentes primarias que se recurrió para el desarrollo de la librería JavaScript que permitirá detectar los errores de escritura de los casos de uso escritos en un lenguaje de símbolos , la información y la metodología de desarrollo se acoplo a los conceptos del manifiesto ágil, generando una metodología adapta a las necesidades del proyecto tomando en cuenta los 12 principios del desarrollo de software ágil.

### **3.4.2. Fuentes secundarias**

Como fuente secundaria para el soporte del desarrollo de la librería JavaScript se puede mencionar que toda la información fue recopilada mediante el uso de Internet, artículos científicos, libros y folletos.

## **3.5. Diseño de la investigación**

Para el desarrollo del presente proyecto se tomarán en cuenta la ejecución de varias etapas, aplicando el Modelo de Prototipado para desarrollar un prototipo de la aplicación web que utilice el producto final de esta investigación. A continuación, se describe el enfoque metodológico correspondiente a cada una de las fases.

### **3.5.1. Metodología de desarrollo aplicada al proyecto**

Para ejecutar el proyecto presentado, se analizó el Manifiesto por el Desarrollo Ágil de Software. La metodología aplicada a este proyecto se basó los Principios del Manifiesto Ágil. Con el objetivo de cumplir las directrices de una metodología ágil. En la tabla 3.1 se redactan los 12 principios del manifiesto ágil y análisis aplicado a la metodología [23].

#### **3.5.1.1. Descripción general de la metodología**

Esta metodología consta de 5 fases. Aparentemente agrupan fases que estén relacionadas al desarrollo de un sistema, desde el análisis de requisitos hasta su debido mantenimiento. Esta última es muy olvidada por los investigadores, y es de mucho interés en la vida de desarrollo. En la figura 3.6 se pueden observar las fases que deben ser implementadas para utilizar la metodología redactada.

Las principales razones del uso de una metodología ágil es el uso de un ciclo de desarrollo iterativo e incremental para la ejecución de este proyecto son:

- Entregas frecuentes y continuas de forma que puede disponer de una funcionalidad básica en un tiempo mínimo y a partir de ahí un incremento y mejora continua del sistema.

Tabla 3.1

*Los 12 principios ágiles y su análisis en la metodología.*

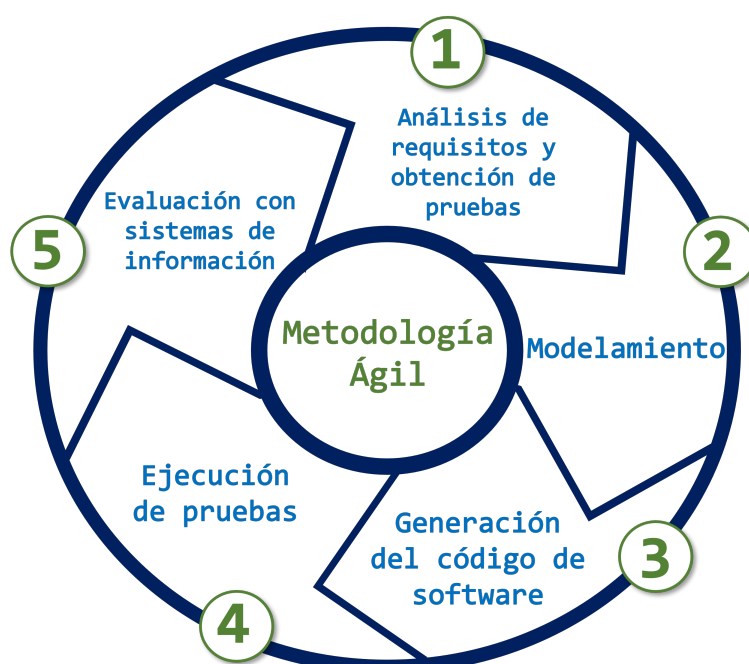
<b>Principios del manifiesto ágil</b>	<b>Análisis aplicado a la metodología</b>
1. <i>Nuestra mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor</i>	Para la entrega continua de cambios sobre el desarrollo del guion, se describieron varias fases de desarrollo.
2. <i>Aceptamos que los requisitos cambien, incluso en etapas tardías del desarrollo. Los procesos Ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.</i>	A lo largo de la ejecución de la metodología se podrá realizar modificaciones en los requisitos que se plantearon, con el objetivo de mejorar las funcionalidades del software.
3. <i>Entregamos software funcional frecuentemente, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible.</i>	Se organiza una fase en la que se genera todo el código fuente del sistema informático con comprobaciones constantes.
4. <i>Los responsables de negocio y los desarrolladores trabajamos juntos de forma cotidiana durante todo el proyecto.</i>	Durante la ejecución del proyecto se mantiene constante contacto con el cliente
5. <i>Los proyectos se desarrollan en torno a individuos motivados. Hay que darles el entorno y el apoyo que necesitan, y confiarles la ejecución del trabajo.</i>	Para brindar el entorno de trabajo adecuado a los desarrolladores, la metodología se la puede llevar a cada de forma remota, es decir en la comodidad de su hogar.
6. <i>El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara.</i>	Se realizaron reuniones virtuales y en ciertas ocasiones reuniones presenciales con el cliente final.
7. <i>El software funcionando es la medida principal de progreso.</i>	Se realiza el desarrollo completo de cada módulo que fue designado.
8. <i>Los procesos Ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios debemos ser capaces de mantener un ritmo constante de forma indefinida.</i>	Dentro de la metodología se especifican varias fases en las cuales todos los integrante pueden opinar o tratar con mejoras mediante ideas claras y precisas.
9. <i>La atención continua a la excelencia técnica y al buen diseño mejora la Agilidad.</i>	En la fase de desarrollo se debe realizar la codificación de manera forma, con código Totalmente documentado y organizado.

- Previsible inestabilidad de requisitos. Es posible que el sistema incorpora más funcionalidades de las inicialmente identificadas
- Es posible que durante la ejecución del proyecto se altere el orden en el que se desean

Principios del manifiesto ágil	Análisis aplicado a la metodología
10. <i>La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial.</i>	Al momento de entrar en la fase final de desarrollo se realiza análisis sobre la funcionalidad intuitiva del software.
11. <i>Las mejores arquitecturas, requisitos y diseños emergen de equipos auto-organizados.</i>	Todos los integrantes del proyecto estarán designados por varios roles que permitirán llevar todo de forma ordenada.
12. <i>A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia.</i>	Cuando se culmina la fase de desarrollo se realiza una revisión general sobre todo el funcionamiento del software mediante pruebas de software.

ELABORADO: DÚVAL CARVAJAL SUÁREZ

Figura 3.6  
Fases de la metodología



ELABORADO: DÚVAL CARVAJAL SUÁREZ

recibir los módulos.

Para mejorar la ejecución de la metodología se organizaron diferentes tipos de roles que deben ser tomados por las personas que desarrollen del proyecto. Cada rol implementado juega un papel muy importante dentro de la metodología, debido a que deben cumplir tareas de suma importancia para controlar todas las fases que se deben seguir y cumplir los objetivos del trabajo a desarrollar.

**Roles** El equipo de esta metodología está conformado por 2 roles, director de proyecto (DP) y el Desarrollador de Sistemas (DS). Todos los miembros de un equipo de desarrollo tienen diferentes roles en la gestión y supervisión de los proyectos. Todos los roles son necesarios para que el proceso funcione eficientemente.

- **Director de proyecto (DP):** Se encarga de administrar el proceso del proyecto, su planificación, coordinación con el analista y realizar un seguimiento e informes del progreso del proyecto, en términos de calidad, costo y plazos de entrega. El DP es la interfaz principal entre el propietario del producto y el analista de desarrollo de software.
- **Desarrollador de Sistemas (DS):** La persona encargada de este rol debe contar con conocimientos de desarrollo de software. El desarrollador deberá estar totalmente familiarizado con el objetivo principal del trabajo. Debido a que será el encargado de analizar todos los requisitos que se necesiten para cumplir con el funcionamiento óptimo del sistema, también se encargará de analizar, diseñar y codificar el producto final del proyecto.
  - Comprometerse al inicio de cada módulo y desarrollar todas las funcionalidades en el tiempo determinado.
  - Son responsables de entregar un producto a cada término de un módulo.
  - Definir el desarrollo del sistema.

**Fases** En esta sección se describen cada una de las fases que se mencionaron en el apartado anterior, también se mencionarán breves conceptos relacionados a cada fase y como pueden mitigar los problemas que se pueden presentar a lo largo de la ejecución de la metodología redactada.

1. **Análisis de requisitos y obtención de pruebas:** Para la recopilación de los requisitos del software es necesario comprenderlos en profundidad para asegurar su correcto funcionamiento cuando este desarrollado. Si al momento de recopilar la información, no lleva todo el tiempo suficiente para ser analizado correctamente podría afectar todo el proyecto en general [24].

Para que los requisitos obtenidos sean realizados de manera correcta se pueden priorizar. Existen técnicas actuales para realizar la priorización de requisitos como es el Proceso de Jerarquía Analítica (AHP) ya que produce resultados muy precisos. A continuación se especifican tres tipos de técnicas de priorización [24]:

- *Escala nominal:* El personal de trabajo que recopilo los requisitos, deberán asignar cada requisito a un grupo de prioridad, transformando a todos los requisitos de ese grupo prioritarios. La asignación numérica, categoriza los requisitos en grupos. Cada grupo cuenta con un identificador único que les permite ser identificados dentro de los demás grupos.
- *Escala ordinal:* Esta técnica produce una lista de requisitos y cada uno cuenta con una prioridad en específica. Existe una técnica similar a la de escala nominal que se basa en crear varios grupos de prioridad. Pero solo son 3 tipos de grupos: alto, medio y bajo, los desarrolladores priorizan y clasifican los requisitos dentro del mismo grupo en otro subgrupo; ese bucle se repite hasta que cada grupo tenga sólo uno.
- *Escala de relación:* Son similares a la escala ordinal, además muestran importancia relativa entre todos los requisitos, es decir que dan valores de prioridad a los requisitos. En estas técnicas, los desarrolladores conocen hasta qué punto cada requisito es más importante que el resto. La votación acumulativa (CV) es una técnica proporcional que depende de la votación de los desarrolladores; cada desarrollador tiene 100 puntos y se distribuyen entre los requisitos en función de su prioridad

La obtención de pruebas se basa en la recolección de los posibles escenarios que puedan ocurrir al momento consumir el producto final del proyecto. Además, se necesita conocer el resultado que se deberá obtener por los parámetros ingresados por cada prueba.

2. **Modelamiento:** El modelado de software básicamente son las abstracciones que describen la arquitectura de un sistema informático. Para llevar a cabo la ejecución de esta fase es necesario tener conocimientos sólidos sobre el desarrollo de software, debido a que los modelos pueden tener diferentes niveles de abstracción y detalles [25].



En [25] se propone un metamodelo que permite representar al software a base de patrones de diseño. Es decir, se pueden crear diferentes tipos de diagramas personalizados especificando el comportamiento que tendrá el sistema informático cuando este en ejecución. Además se pueden vincular los patrones encontrados con pruebas de aceptación con los usuarios que usaran el producto final del proyecto presentado.

3. **Generación de código:** En la fase de generación de código se relaciona con obtener el producto o software de manera real. En esta etapa los desarrolladores deberán cumplir con algunos puntos claves en el desarrollo de software que son: personalizar la interfaz de usuario, implementar la lógica comercial, integrar servicios de terceros si es necesario o resolver problemas de desarrollo muy complejos que requieran un análisis más profundo para llegar a una solución óptima [26].

Se recomienda a los encargados de ejecutar esta etapa tener conocimientos de programación. En [26] mencionan que si la curva de aprendizaje de los desarrolladores es baja, no necesitan obtener conocimientos al momento de generar el código del software. Esto permitirá suponer que el nivel de conocimiento puede impactar en el proceso de aprendizaje y adopción de la tecnología.

Además para suprimir un poco los grandes desafíos a los que se enfrentan los desarrolladores en esta fase, se recomienda el uso de tecnologías con una documentación detallada y cuenten con recursos de aprendizaje. Otra solución potente para ayudar a los desarrolladores es basarse en un sistema similar al que se pretende desarrollar usando el conocimiento de aplicaciones previamente desarrolladas [26].

4. **Ejecución de pruebas:** La siguiente fase está relacionada con la primera. Como se podrá observar en los títulos de estas dos fases, se realiza la obtención y ejecución de pruebas sobre el software que se obtendrá. Existen varios problemas que surgen al momento de poner en producción un software desarrollado. En [27] muestran varios estudios que indican que algunos tipos de fallos son intrínsecamente difíciles si no imposibles de detectar en entornos de desarrollo.

Utilizando las pruebas de campo se puede realizar la manipulación del sistema teniendo a la mano los resultados esperados por los datos que son ingresados como entrada sobre el software. Es decir, en esta fase se debe utilizar los datos que fueron recopilados en la primera fase y ser ingresados en el sistema que ya se encuentra totalmente

desarrollado. Con el fin de obtener resultados similares o precisamente los mismos a los resultados obtenidos inicialmente [27].

5. **Evaluación con sistemas de información:** La fase final de la metodología redactada permitirá usar sistemas de información ya desarrollados que permitan utilizar el producto final del proyecto ejecutado. Se recomienda utilizar sistemas de información con su documentación totalmente detallada, de esa manera se ahorra tiempo en analizar por completo los trabajos para ingresarlos al sistema informático que se desarrolló con esta metodología [28].

### 3.5.2. Modelo de prototipado

El Modelo de Prototipado se aplica cuando la información detallada relacionada a requerimientos de entrada y salida del sistema no está disponible. En este modelo se asume que tal vez no todos los requerimientos son conocidos en el inicio del desarrollo del sistema. Se usa generalmente cuando un sistema no existe, o en caso de un largo y complejo sistema, cuando no hay procesos manuales para determinar los requerimientos. Los pasos que se ejecutan en el modelo de prototipado son:

1. **Obtención y análisis de requisitos:** es el punto de partida del modelo. El usuario es entrevistado para conocer los requisitos del sistema.
2. **Diseño rápido:** teniendo claro todos los requisitos, se procede a crear un diseño rápido y preliminar incluyendo solo los aspectos más importantes
3. **Construir el prototipo:** se trabaja con la información tomada por el diseño rápido y crear el prototipo de la aplicación.
4. **Evaluación de usuarios:** el sistema es presentado a varios usuarios para evaluar y verificar sus puntos fuertes y débiles Se reciben comentarios y sugerencias que serán analizadas por los desarrolladores.
5. **Reajuste del prototipo:** el prototipo actual debe reajustarse según los nuevos requerimientos, es decir que, se debe crear un nuevo prototipo con la información adicional proporcionada por los usuarios evaluados. Este nuevo prototipo será reevaluado justo como el anterior. Este proceso se repite hasta que se cumplan todos los requerimientos especificados

por el usuario. Cuando el usuario esté satisfecho con el resultado, se desarrollará un sistema final basado en el prototipo final.

6. **Implementación y mantenimiento:** Una vez que se tenga listo el sistema, ya estará listo para ser desplegado a producción. El sistema se somete a un mantenimiento de rutina para minimizar el tiempo de inactividad y evitar fallas a gran escala.

### **3.6. Instrumentos de investigación**

### **3.7. Recursos humanos y materiales**

En la siguiente sección se describirán todos los recursos que fueron necesarios para la ejecución del proyecto presentado. Se mencionarán al personal de trabajo que se necesitó y todos los materiales que fueron utilizados.

#### **3.7.1. Talento humano**

Organización y planeación sobre el personal que se encargara de llevar a cabo el proceso de desarrollo del proyecto. Se encarga de coordinar todas las actividades que serán necesarias ser que el proyecto sea culminado con éxito. Además, será el área encarga de definir los tiempos en que se deberá trabajar en todo momento y también designar los tiempos donde se puede descansar para no frutar al personal de trabajo.

#### **3.7.2. Equipo de trabajo**

El equipo de trabajo está conformado por una sola persona. Esta persona será la encargada de llevar a cabo la documentación de todo su trabajo realizado, a más de la recopilación de todos los requisitos funcionales y no funcionales para la librería JavaScript. El integrante del equipo es:

- Dúval Ricardo Carvajal Suárez

#### **3.7.3. Recursos de software**

A continuación se detallaran los recursos de software que fueron necesarios para ejecutar el desarrollo del proyecto presentado:

Tabla 3.2  
*Recursos de software utilizados en el proyecto*

Software	Descripción	Vigencia
Lucidchart	Diseñar algunos gráficos.	Libre (Gratuito)
WebStorm	Editor de código para desarrollar la demo de la librería.	Licencia corporativa (Universitaria)
TexStudio	Herramienta para realizar la documentación del proyecto	Libre (Gratuito)
Teams	Herramienta para realizar videoconferencias con el tutor de proyecto	Libre (Gratuito)
Sistema operativo	Windows 11	Libre (Gratuito)
Programas de oficina	Microsoft Excel, Microsoft Word, Microsoft Power Point	Licencia corporativa (Universitaria)

**FUENTE: PROPIA**  
**ELABORADO: DÚVAL CARVAJAL SUÁREZ**

### 3.7.4. Recursos de hardware

Los recursos que se observan a continuación son tan necesarios para llevar a cabo la ejecución del proyecto. Como el trabajo se realizó remotamente los recursos físicos corren por cuenta del personal de trabajo.

Tabla 3.3  
*Recursos de hardware utilizados en el proyecto*

Cantidad	Equipo	Adquisición	Costo total
1	Laptop.	\$ 1.200,00	\$ 1.200,00
1	Dispositivo de almacenamiento (Disco duro extraíble).	\$ 500,00	\$ 500,00

**FUENTE: PROPIA**  
**ELABORADO: DÚVAL CARVAJAL SUÁREZ**

# **CAPÍTULO IV**

## **RESULTADOS Y DISCUSIÓN**

# 4.1. Resultados de la metodología

Los resultados que se obtuvieron al aplicar la metodología basada en el manifiesto del desarrollo de software ágil se redactan a continuación:

## 4.1.1. Análisis de requisitos y obtención de pruebas

Para llevar a cabo el análisis de los requisitos sobre la librería JavaScript que se desarrolló se tomaron en cuenta varios favores observando las carencias que existen al momento de realizar el modelamiento de cualquier software. Mediante el docente encargado de impartir las clases sobre cómo utilizar los lenguajes de modelado como lo es UML se dio cuenta que podría existir una forma en generar uno de los diagramas más importantes que es el diagrama de clases, pero a partir de las descripciones que se generan en los casos de uso del sistema.

Existe una herramienta denominada TDDT4IoTS que permite realizar el modelamiento de varios tipos de software. La herramienta cuenta con un lenguaje de símbolos que permiten escribir dentro de las descripciones de casos de uso datos técnicos sobre los objetos que intervendrán en el sistema informático que llevara a cabo la ejecución de todos los escenarios que se están describiendo. A petición del cliente recomendó que visitemos el sitio web de aplicaciones.uteq.edu.ec/tddt4iots para visualizar los símbolos que usa la herramienta (ver figura 4.7).

Figura 4.7  
*Símbolos que utiliza la herramienta TDDT4IoTS*

Shared Projects

USER MANAGEMENT


Application users

UML

Interpret

IIOT SYSTEMS

IIoT Components



To compile and burn the code to your arduino board click on the button that says download to get the respective application.

Download


News Highlights - Latest projects




Creation date: 2022-09-22T02:07:43.505 -  
Update date:  
**CRIPTO INFO**  
Intento diagramas proyecto, anthony pachay




Creation date: 2022-09-21T08:32:02.319 -  
Update date:  
**Smart classroom**  
proyecto



Creation date: 2022-09-20T18:32:34.587 -  
Update date:  
**Smart Plant 2.0**  
Desarrollo de una maceta inteligente para hogar



Creation date: 2022-09-19T18:08:25.787 -  
Update date:  
**ejemplo**  
guia



Creation date: 2022-09-18T28:45:49.23 -  
Update date:  
**IdeaAir2.0**  
Idea Air

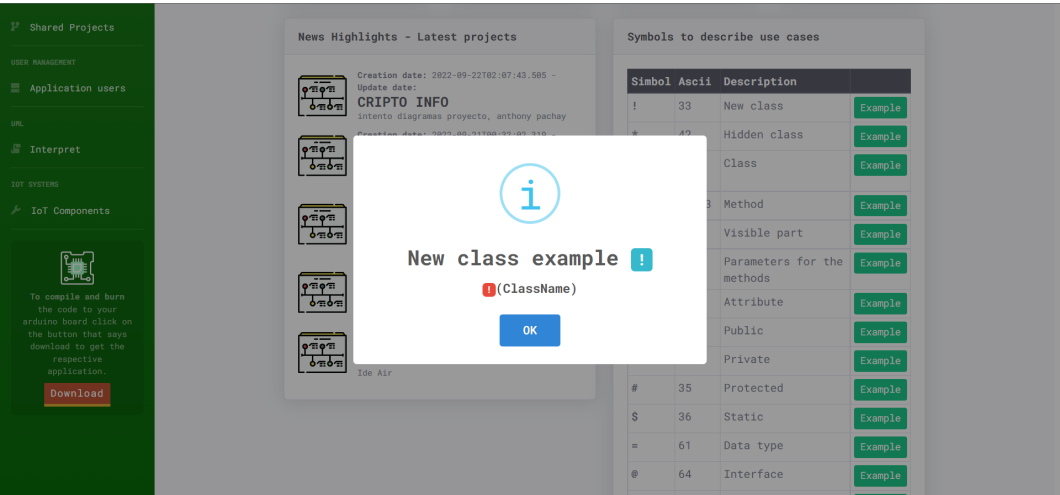
Symbols to describe use cases

Simbol	Ascii	Description	
!	33	New class	Example
*	42	Hidden class	Example
()	40 - 41	Class	Example
[]	91-93	Method	Example
/	47	Visible part	Example
{.}	46	Parameters for the methods	Example
&	38	Attribute	Example
+	43	Public	Example
-	45	Private	Example
#	35	Protected	Example
\$	36	Static	Example
=	61	Data type	Example
@	64	Interface	Example

FUENTE: PROPIA  
ELABORADO: DÚVAL CARVAJAL SUÁREZ

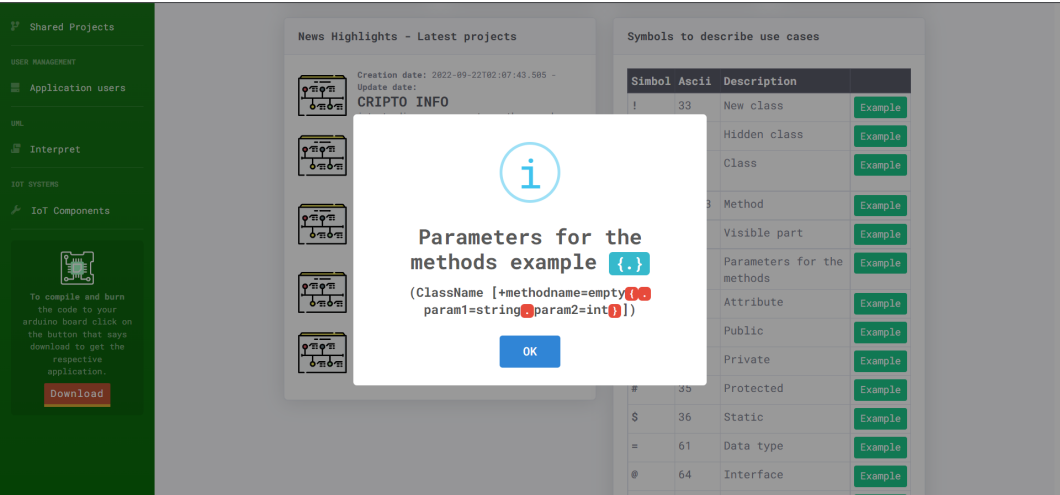
En la lista de símbolos que se visualizar en la figura anterior se encuentra un botón que dice "Example". Si presionamos en ese botón se pudo observar un ejemplo detallado sobre como se deben utilizar los símbolos para tratar de especificar algún objeto del diagrama de clases que se generar mediante la descripción del caso de uso pertinente (ver figuras 4.8, 4.9).

Figura 4.8  
*Ejemplo de como se debe utilizar el símbolo respectivo para crear una clase.*



**FUENTE: PROPIA**  
**ELABORADO: DÚVAL CARVAJAL SUÁREZ**

Figura 4.9  
*Ejemplo de como se debe utilizar el símbolo respectivo generar los parámetros de un método declarado con el lenguaje de símbolos.*



**FUENTE: PROPIA**  
**ELABORADO: DÚVAL CARVAJAL SUÁREZ**

Luego de tener una idea general sobre los símbolos que utiliza la herramienta se especificaron

detalladamente los significados de cada símbolo y como puede ser utilizado en las descripciones de los casos de uso.

- **Asterisco \***: Este símbolo sirve para ocultar cualquier carácter que se encuentre redactado después el. Esto permitirá al momento de interpretar la descripción del caso de uso ingresada eliminar los caracteres mezclados con los símbolos y solo dejar visibles los caracteres normales formando un texto natural que pueda ser entendido por cualquier persona regular. **Ejemplo:**

`* texto de prueba`

- **Paréntesis de apertura y cierre ()** : Este símbolo sirve para especificar uno de los componentes más importantes del diagrama de clases, dentro de los paréntesis de apertura y cierre se deberá especificar el nombre de la clase u objeto que se pretende generar de forma automática. Cabe mencionar que, si se escribe el nombre de clase separada por espacios, la librería deberá omitir estos espacios y auto completarlos con la segunda letra después del espacio con mayúscula. **Ejemplo:**

`(Nombre de la clase)`

- **Corchete de apertura y cierre []**: Este símbolo podrá ser utilizado dentro de los símbolos para crear las clases, permitirá definir Los métodos o funciones que pertenecerá a la clase respectiva. **Ejemplo:**

`(Nombre de la clase [+nombre del metodo=empty])`

- **Slash /**: Este símbolo es bastante interesante, con el se debe visualizar cualquier caracteres que se encuentre encerrado de un slash de apertura y otro slash de cierre. Este símbolo se lo deberá usar cuando todos los caracteres se encuentren después del símbolo del asterisco, la idea es permitir que se visualicen caracteres específicos dentro de los datos técnicos para generar el diagrama de clases, sin afectar el uso de los demás símbolos. **Ejemplo:**

`(Nombre de la clase &+/nombre/=string)`



- **Llaves de apertura y llaves de cierre con punto {.}:** Con el símbolo de las llaves se podrá definir los parámetros de entrada para los métodos o funciones de las clases. Cada parámetro estará separado por un punto, además se deberá utilizar el símbolo del igual (=) para especificar su tipo de dato. Cada recalcar que todos los atributos también deberán estar especificados dentro de las llaves de apertura y cierre. **Ejemplo:**

```
(Nombre de la clase [+nombre del metodo=empty
{.param1=string.param2=string}])
```

- **Ampersand &:** Este símbolo debe permitir declarar los atributos de la clase que fue creada con el símbolo anterior. Existen otros símbolos que interviene en la creación de otros componentes del diagrama de clases, más adelante se detallaran para que sirven. **Ejemplo:**

```
(Nombre de la clase &+atributo uno=string &+atributo
dos=string)
```

- **Visibilidad +:** El símbolo de suma permite especificar que la visibilidad del atributo, método o función será de manera pública. **Ejemplo:**

En este ejemplo se visualiza la forma en cómo utilizar el símbolo de suma en atributos de clase.

```
(Nombre de la clase &+atributo uno=string &+atributo
dos=string)
```

En este ejemplo se visualiza la forma en cómo utilizar el símbolo de suma en métodos o funciones de clase.

```
(Nombre de la clase [+nombre del metodo=empty
{.param1=string.param2=string}])
```

- **Visibilidad -:** El símbolo de resta permite especificar que la visibilidad del atributo, método o función será de manera privada. **Ejemplo:**

En este ejemplo se visualiza la forma en cómo utilizar el símbolo de suma en atributos de clase.

```
(Nombre de la clase &-atributo uno=string &-atributo  
dos=string)
```

En este ejemplo se visualiza la forma en cómo utilizar el símbolo de suma en métodos o funciones de clase.

```
(Nombre de la clase [-nombre del metodo=empty  
{.param1=string.param2=string})
```

- **Visibilidad #:** El símbolo de almohadilla permite especificar que la visibilidad del atributo, método o función será de manera protegida. **Ejemplo:**

En este ejemplo se visualiza la forma en cómo utilizar el símbolo de suma en atributos de clase.

```
(Nombre de la clase &#atributo uno=string &#atributo  
dos=string)
```

En este ejemplo se visualiza la forma en cómo utilizar el símbolo de suma en métodos o funciones de clase.

```
(Nombre de la clase [#nombre del metodo=empty  
{.param1=string.param2=string})
```

- **Visibilidad \$:** El símbolo de dólar permite especificar que la visibilidad del atributo, método o función será de manera estática. **Ejemplo:**

En este ejemplo se visualiza la forma en cómo utilizar el símbolo de suma en atributos de clase.

```
(Nombre de la clase &$atributo uno=string &$atributo  
dos=string)
```

En este ejemplo se visualiza la forma en cómo utilizar el símbolo de suma en métodos o funciones de clase.

```
(Nombre de la clase [$nombre del metodo=empty
{.param1=string.param2=string}])
```

- **Igual =:** El símbolo de igual debe permitir asignar el tipo de dato a los atributos, métodos o funciones declaradas dentro de la clase. En el caso de los métodos de tipo **void** se deberá utilizar la palabra *empty* indicando que es un método que retornara ningún valor. **Ejemplo:**

```
{.param1=string.param2=string}
```

- **Arroba @:** El símbolo de arroba se lo utiliza dentro de los paréntesis que permiten generar una clase del diagrama. Pero se recuerda que en el diagrama de clases también se pueden crear interfaces. El objetivo de este simbolo es utilizarlo dentro de los paréntesis pero el interprete identifica que sera una interfaz la que se deberá crear. **Ejemplo:**

```
(@nombre interfaz)
```

- **Signo de interrogación apertura y cierre ¿?:** El simbolo de interrogación se lo utiliza para generar otro tipo de objeto principal del diagrama de clases. El objetivo principal de este simbolo es generar los enum. **Ejemplo:**

```
¿enumTest?
```

- **Cierra comillas bajas »:** El simbolo de cierre comillas bajas se lo utiliza para agregar atributos a un objeto principal del diagrama de clases. El objetivo principal de este simbolo es atributos dentro de los objetos enum. **Ejemplo:**

```
¿enumTest »atributoUno »atributoDos?
```

- **Porcentaje %:** El simbolo de porcentaje sirve para agregar nuevos constructores a las clases que ya fueron definidas, ademas para especificar sus parámetros se utiliza los mismos símbolos para agregara atributos a la clase, con la diferencia que los parámetros van separados por la coma. **Ejemplo:**

```
* (class %.class=Class, .param=string%)
```

Dentro del diagrama de clases existen diferentes tipos de relaciones con las que se pueden relacionar las clases que se encuentran definidas. Para implementar las relaciones también se utilizan una combinación de símbolos que permitan detectarlas y la librería se encarga de generarlas. A continuación se especificaran los símbolos que se utilizan para los tipos de relaciones en el diagrama de clases:

- **Simbolo de admiración de apertura ;!:** Para indicar que se va realizar una relación entre 2 objetos del diagrama de clases, el texto debe estar entre este simbolo tanto en su apertura y cierre.
- **Corchete de apertura y cierre []:** Dentro del simbolo de admiración se podrá indicar un texto como leyenda sobre la linea de la relación, esto indicara alguna palabra clave sobre la relación entre los objetos.
- **Cardinalidad 1 o n:** Sobre la linea de la relación se puede indicar una cardinalidad de uno a muchos. El numero 1 indicara que la cardinalidad es de 1 y si usa la letra n indicara que la cardinalidad des de muchos. Se los debe utilizar en la parte exterior de los corchetes.
- **Agregación > >:** Para indicar que el tipo de relación entre dos objetos del diagrama de clases se utiliza los símbolos de > >. Se deben colocar en la parte exterior donde se el texto de cardinalidad. **Ejemplo:**

```
*|claseUno 1>[texto de cardinalidad]>n claseDos!
```

- **Dependencia < <:** Para indicar que el tipo de relación entre dos objetos del diagrama de clases se utiliza los símbolos de < <. Se deben colocar en la parte exterior donde se el texto de cardinalidad. **Ejemplo:**

```
*|claseUno 1<[texto de cardinalidad]<n claseDos!
```

- **Generalización < >:** Para indicar que el tipo de relación entre dos objetos del diagrama de clases se utiliza los símbolos de < >. Se deben colocar en la parte exterior donde se el texto de cardinalidad. **Ejemplo:**

```
*;claseUno 1<[texto de cardinalidad]>n claseDos!
```

- **Asociación > <:** Para indicar que el tipo de relación entre dos objetos del diagrama de clases se utiliza los símbolos de > <. Se deben colocar en la parte exterior donde se el texto de cardinalidad. **Ejemplo:**

```
*;claseUno 1>[texto de cardinalidad]<n claseDos!
```

En la tabla se describe el código ASCII para escribir los símbolos mediante el teclado de una computadora. Se pretendió que cada simbolo tenga una combinación ASCII para que el analista pueda escribir cada simbolo con el teclado y no obtenerlo de alguna forma que sea complicada.

Tabla 4.4  
*Código ASCII para cada simbolo del lenguaje.*

Simbolo	ASCII	Descripción
*	33	Ocultar texto
( )	40 - 41	Crear clase
[ ]	91 - 93	Crear método
/	47	Parte visible
{.}	123 - 125	Parámetro de los métodos
&	38	Atributo de clase
+	43	Publico
-	45	Privado
#	35	Protegido
\$	36	Estático
=	61	Tipo de dato
@	64	Interfaz
¿?	168 - 63	Enum
%	37	Constructores

**ELABORADO: DÚVAL CARVAJAL SUÁREZ**

Para definir los diferentes tipos de relaciones que se usaran en el diagrama de clases. En la tabla se visualiza un grupo de símbolos que servirán para crear las relaciones de tipo: agregación, dependencia, herencia o generalización y asociación.

Tabla 4.5

*Código ASCII para cada símbolo del lenguaje que especifican las relaciones.*

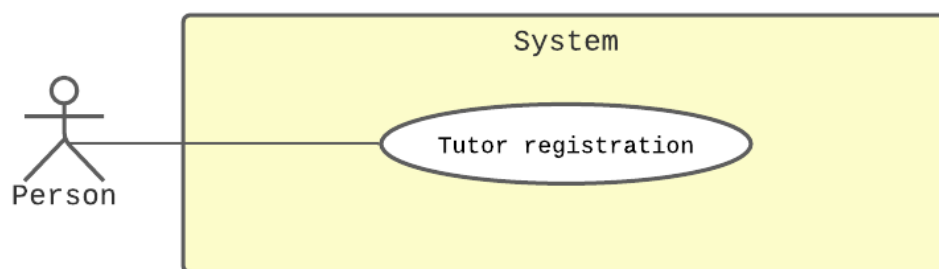
<b>Símbolo</b>	<b>ASCII</b>	<b>Descripción</b>
!;	33 - 173	Crear relación
[ ]	91 - 93	Especificar leyenda en la relación
1[ ]1	–	Cardinalidad de uno a uno
1[ ]n	–	Cardinalidad de uno a muchos
n[ ]1	–	Cardinalidad de muchos a uno
>>	62 - 62	Agregación
<<	60 - 60	Dependencia
<>	60 - 62	Generalización
><	62 - 60	Asociación

**ELABORADO: DÚVAL CARVAJAL SUÁREZ**

Luego de haber recopilado todos los símbolos que la librería va a interpretar, el docente proporciona 1 caso de uso. Además proporciona el gráfico del diagrama de clases que se escribió en los casos de uso. En la figura 4.10 se observa el caso de uso de prueba para saber qué objetos debe generar la librería.

Figura 4.10

*Diagrama de casos de uso para las pruebas.*



**ELABORADO: DÚVAL CARVAJAL SUÁREZ**

Para comprender totalmente las acciones que se deben ejecutar en el caso de uso que se usará como prueba, está escrito en 2 distintas formas. En la tabla 4.6 se observa la descripción del caso de uso en lenguaje natural especificando todas las acciones con el objetivo que cualquier persona regular logre entender. En la tabla 4.7 se observa la misma descripción del caso de uso pero utilizando los símbolos.

Tabla 4.6

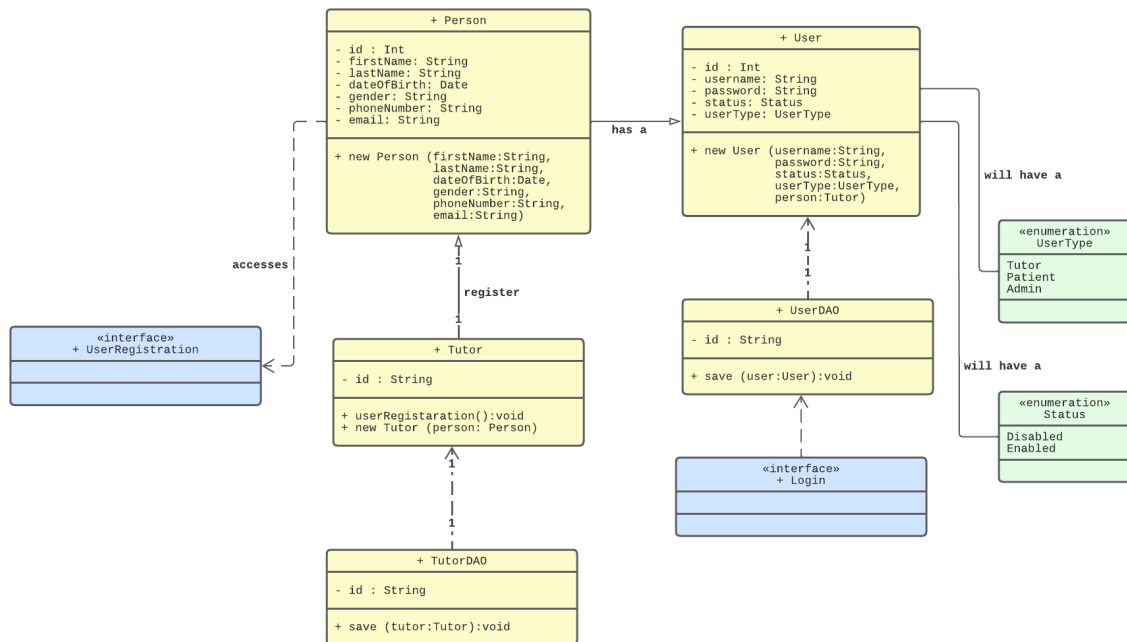
*Descripción del caso de uso para Tutor registration.*

<b>Use case:</b>	Tutor registration
<b>Actors:</b>	Person
<b>Postcondition:</b>	User registered in the database.
<b>Precondition:</b>	The user exists in the database.
<b>Purpose:</b>	Ingresar a la interfaz principal de la aplicación
<b>Abstract:</b>	Permite identificar las credenciales del usuario que intenta ingresar al sistema, de esa forma verificar el tipo de usuario que ingresa.
<b>Type:</b>	Primary
<b>Normal flow</b>	
1. This use case starts when a person wants to register as a tutor user in the system.	
2. The person accesses then User Registration interface.	
	3. Shows you the fields to fill in the interface user registration : first name last name date of birth gender phone number email username password. It should be taken into account that the registration of a Person has a User. There are states that can be Disabled o Enabled that each User will have a Status . There are also several types of users, which are Tutor or Patient or Admin and every User will have a UserType. In addition, the system assigns an id when storing it.
4. The person enters all the data required by the view, and clicks submit.	
	5. Create a person object.
	6. Create a tutor object.
	7. Create an user object.
	8. Saves the tutor data in the database.
	9. Saves the user data in the database.
	10. This use case ends when the system displays the login Interface.

**ELABORADO: DÚVAL CARVAJAL SUÁREZ**

Para la elaboración de la descripción del caso de uso con los símbolos proporcionaron información de un diagrama de clases que ya se encontraba realizado. El objetivo de este caso de uso era escribir todas las acciones con información técnica que permita crear el diagrama de clases lo mas parecido posible al original. En la figura 4.11 se observa el diagrama de clases creado de forma manual.

Figura 4.11  
Diagrama de clases de prueba



ELABORADO: DÚVAL CARVAJAL SUÁREZ

Tabla 4.7  
Descripción del caso de uso para Tutor registration.

<b>Use case:</b>	Tutor registration
<b>Actors:</b>	Person
<b>Postcondition:</b>	User registered in the database.
<b>Precondition:</b>	The user exists in the database.
<b>Purpose:</b>	Ingresar a la interfaz principal de la aplicación
<b>Abstract:</b>	Permite identificar las credenciales del usuario que intenta ingresar al sistema, de esa forma verificar el tipo de usuario que ingresa.
<b>Type:</b>	Primary
<b>Normal flow</b>	
1. This use case starts when a person *(person &-id=int) wants to register as a tutor *(tutor &-id=int [+userRegistration=Tutor]) user in the system. *;tutor 1<[register]>n Person!	
2. The person accesses then User Registration *(@User Registration) interface. *;person 1<[accesses]<1 User Registration!	



	<p>3. Shows you the fields to fill in the interface user registration <code>*(@User registration):</code> <code>*(Person &amp;-/first name/=String, &amp;-/last name/=String, &amp;-/date of birth/=Date, &amp;-/gender/=String, &amp;-/phone number/=String, &amp;-/email/=String) *(User &amp;-id=Int, &amp;-/username/=String, &amp;-/password/=String &amp;-status=Status &amp;-user type=UserType).</code> It should be taken into account that the registration of a <code>*!/Person/ u&lt;[/has a/]&gt;u /User/;</code>. There are states that can be <code>*;Status »/Disabled/ /o/ »/Enabled/?</code> that each <code>*;/User/ u&gt;[/will have a/]&lt;u /Status/!</code>. There are also several types of users, which are <code>*;UserType »/Tutor/ /or/ »/Patient/ /or/ »/Admin/?</code> and every <code>*;/User/ u&gt;[/will have a/]&lt;u /UserType/!</code>. In addition, the system assigns an id when storing it.</p>
4. The person enters all the data required by the view, and clicks submit.	
	<p>5. Create a <code>*/(person/ %.first name=String, .last name=String, .date of birth=Date, .gender=String, .phone number=String, .email=String%)</code> object.</p>
	<p>6. Create a <code>*/(tutor/r %.person=Person%)</code> object</p>
	<p>7. Create an <code>*/(user/ %.username=String, .password=String, .status=Status, .user type=UserType, .person=Tutor%)</code> object</p>
	<p>8. <code>*(TutorDAO &amp;id=Int [/Save/=Tutor.tutor=Tutor])s</code> the tutor data in the database. <code>*;TutorDAO&lt;[]&lt;tutor!.</code></p>
	<p>9. <code>*(UserDAO &amp;-id=Int [/Save/=User.user=User])s</code> the user data in the database. <code>*;UserDAO &lt;[]&lt;User!.</code></p>
	<p>10. This use case ends when the system displays the <code>*(@Login)</code> login Interface. <code>*;Login u&lt;[]&lt;u UserDAO!</code></p>

**ELABORADO: DÚVAL CARVAJAL SUÁREZ**

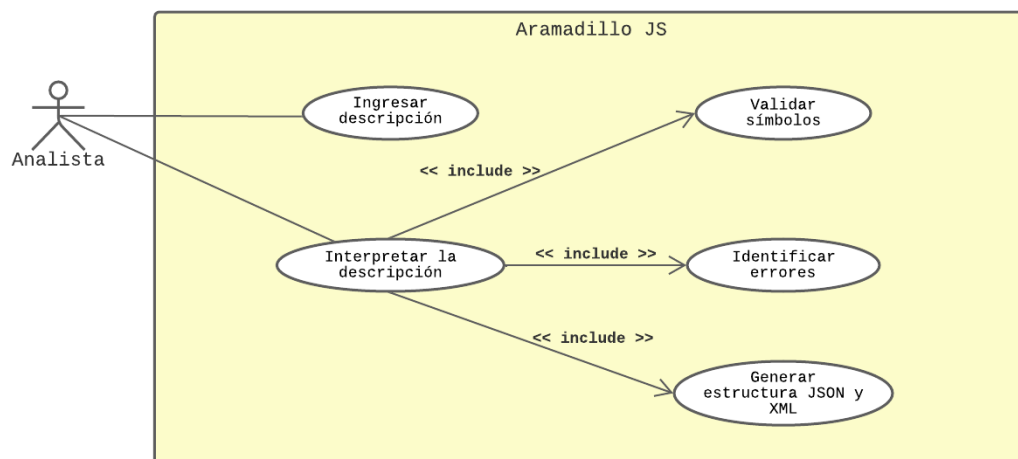
## 4.2. Modelamiento

Se utilizó el lenguaje de programación JavaScript con la intención de generar un paquete totalmente exportable a otros proyectos que requieran utilizar la librería para generar sus propios diagramas con otras librerías de dibujo. Para empezar con el diseño o modelamiento de la librería se especificó el proceso normal que deberá seguir la al momento de recibir como datos de entrada las descripciones de los casos de uso. En la figura 4.12 se observa el proceso que se llevara a cabo de forma general la ejecución de la librería.

En la figura 4.12 se pudo observar el diagrama de casos de uso que explica las acciones que el analista podrá realizar al momento de implementar la librería, o podrá utilizar la aplicación de demostración

Figura 4.12

*Diagrama de casos de uso armadillo.*



**ELABORADO: DÚVAL CARVAJAL SUÁREZ**

A continuación, se detallarán las descripciones de los casos de uso que se observan en la imagen anterior. Se especificarán todos los pasos que la librería debe recibir para responder de forma correcta. Se recuerda que la librería no cuenta con acceso a datos o algún tipo de información en la nube.

En la tabla 4.8 se detallan los pasos que se siguen al momento de ingresar las descripciones de los casos de uso que se pretenden interpretar.

Tabla 4.8

*Descripción del caso de uso para ingresar la descripción.*

<b>Caso de uso:</b>	Ingresar descripción
<b>Actores:</b>	Analista
<b>Precondición:</b>	Tener instanciada la librería en su proyecto web, o debe utilizar la aplicación de demostración.
<b>Postcondición:</b>	Descripción ingresada en la librería.
<b>Propósito:</b>	Ingresar un párrafo de la descripción del caso de uso.
<b>Resumen:</b>	Permite ingresar un párrafo de toda la descripción del caso de uso utilizando el lenguaje de símbolos.
<b>Tipo:</b>	Primario
<b>Flujo normal</b>	
1. Este caso de uso inicia el analista pretende ingresar la descripción del caso de uso.	
2. Redacta la descripción del caso uso utilizando el lenguaje de símbolos.	
	3. Muestra la caja de texto donde se debe colocar la descripción del caso de uso con el lenguaje de símbolos.
4. Este caso de uso finaliza cuando el analista ingresa la descripción del caso de uso.	
<b>Flujos alternos</b>	

**ELABORADO: DÚVAL CARVAJAL SUÁREZ**

Tabla 4.9

*Descripción del caso de uso para interpretar la descripción.*

<b>Caso de uso:</b>	Interpretar descripción
<b>Actores:</b>	Analista
<b>Precondición:</b>	Tener ingresada la descripción a interpretar redactada con el lenguaje de símbolos.
<b>Postcondición:</b>	Obtener la descripción redactada en lenguaje natural.
<b>Propósito:</b>	Obtener la descripción del caso de uso e interpretar los símbolos del lenguaje.
<b>Resumen:</b>	Permite identificar cada simbolo del lenguaje para identificar los objetos que se deben crear en el diagrama de clases.
<b>Tipo:</b>	Primario
<b>Flujo normal</b>	
<b>Acción del actor</b>	<b>Respuesta del sistema</b>
1. Este caso de uso inicia cuando el analista pretende interpretar la descripción del caso de uso.	
2. El analista da la orden de interpretar la descripción con el lenguaje de símbolos.	
	3. Muestra dentro de un apartado denominado consola, todos los procesos que realizaron al momento de interpretar la descripción. Además de mostrar rápidamente el texto redactado en lenguaje normal.
4. El analista se dirige a la pestaña de "Class diagram".	
	5. Muestra el grafico del diagrama de clases generado por la librería. Se dibujan los objetos que pudieron ser identificados por los simbolos usados.
6. El analista se dirige a la pestaña de "JSON".	
	7. Muestra la estructura JSON generada.
8. El analista se dirige a la pestaña de "XML".	
	9. Muestra la estructura XML generada.
10. Este caso de uso finaliza cuando el analista descarga las estructuras JSON o XML para posteriores proyectos.	
<b>Flujos alternos</b>	

**ELABORADO: DÚVAL CARVAJAL SUÁREZ**

Tabla 4.10  
*Descripción del caso de uso para validar símbolos.*

<b>Caso de uso:</b>	Validar símbolos
<b>Actores:</b>	Analista
<b>Precondición:</b>	La descripción del caso de uso debe contar con los símbolos del lenguaje.
<b>Postcondición:</b>	Todos los símbolos serán omitidos en el texto que sea interpretado.
<b>Propósito:</b>	Identificar los objetos para generar el diagrama de clases
<b>Resumen:</b>	Permite identificar los símbolos que son usados para redactar los casos de uso. Además de verificar si se los está usando de forma correcta.
<b>Tipo:</b>	Secundario
<b>Flujo normal</b>	
	1. Este caso de uso inicia cuando la descripción del caso fue ingresada con los símbolos del lenguaje.
	2. Se identifican los tipos de símbolos que tiene el lenguaje, si son símbolos con apertura y cierre o son individuales.
	3. Se verifica que los símbolos están usándose de forma correcta sin combinarlos con otros símbolos desconocidos.
	4. Este caso de uso finaliza cuando se identificaron todos los símbolos de la descripción y oculta los símbolos para obtener un texto natural.
<b>Flujos alternos</b>	

**ELABORADO: DÚVAL CARVAJAL SUÁREZ**

Tabla 4.11

*Descripción del caso de uso para identificar errores.*

<b>Caso de uso:</b>	Identificar errores
<b>Actores:</b>	Analista
<b>Precondición:</b>	La descripción del caso de uso debe contar con los símbolos del lenguaje.
<b>Postcondición:</b>	Todos los símbolos serán omitidos en el texto que sea interpretado.
<b>Propósito:</b>	Permite identificar la mala escritura de las descripciones de los casos de uso.
<b>Resumen:</b>	Identificar los posibles errores que se pueden cometer al momento de usar los símbolos en las descripciones de los casos de uso.
<b>Tipo:</b>	Secundario
<b>Flujo normal</b>	
	1. Este caso de uso inicia cuando la descripción del caso fue ingresada con los símbolos del lenguaje.
	2. Se identifican los tipos de símbolos que tiene el lenguaje, si son símbolos con apertura y cierre o son individuales.
	3. Si existen inconsistencias en la redacción de la descripción del caso de uso, se empieza acumular mensajes de alerta.
	4. Este caso de uso finaliza mostrando los mensajes en un apartado y clasificando los mensajes acumulados mediante un color diferentes: Azul para información, amarillo para advertencia, verde para exitoso y rojo para notificar errores graves.
<b>Flujos alternos</b>	

**ELABORADO: DÚVAL CARVAJAL SUÁREZ**

Tabla 4.12

*Descripción del caso de uso para generar estructura json y xml.*

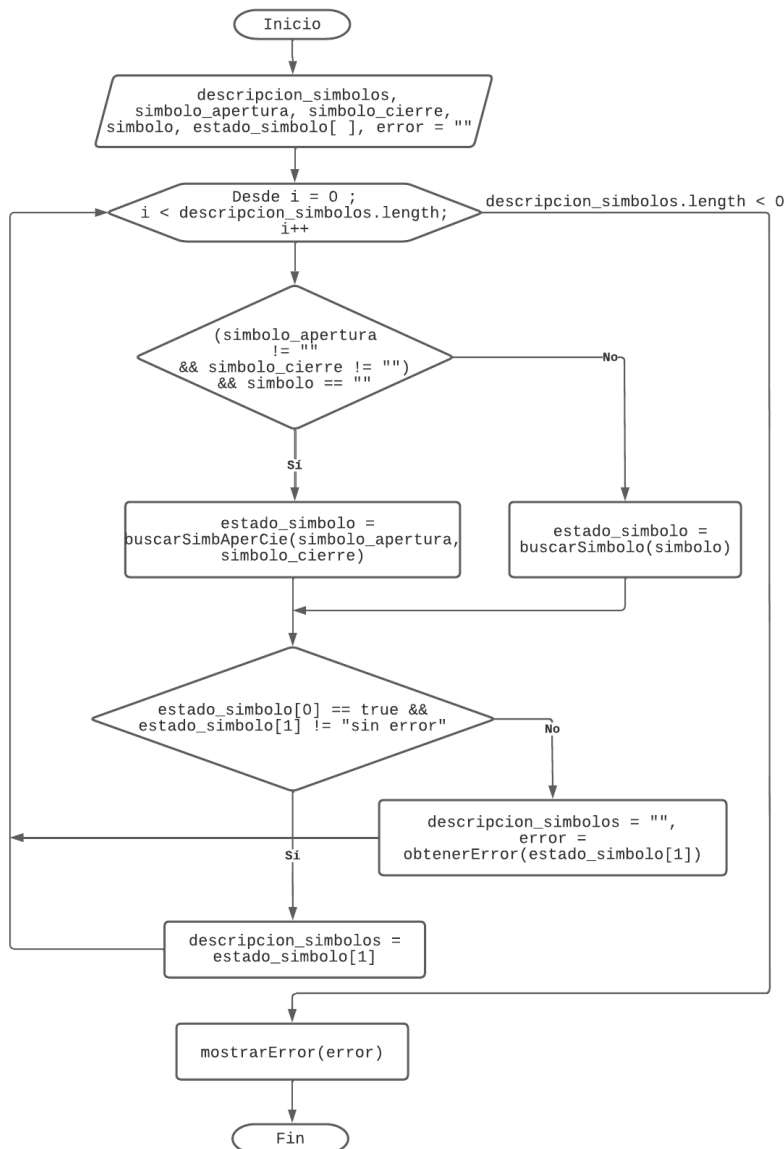
<b>Caso de uso:</b>	Generar estructura JSON y XML
<b>Actores:</b>	Analista
<b>Precondición:</b>	La descripción del caso de uso debe contar con los símbolos del lenguaje.
<b>Postcondición:</b>	Todos los símbolos serán omitidos en el texto que sea interpretado.
<b>Propósito:</b>	Generar 2 estructuras con la información necesaria del diagrama de clases.
<b>Resumen:</b>	Permite obtener toda la estructura del diagrama de clases generado en formato JSON Y XML
<b>Tipo:</b>	Secundario
<b>Flujo normal</b>	
	1. Este caso de uso inicia cuando la descripción del caso fue ingresada con los símbolos del lenguaje.
	2. Buscar los objetos que fueron identificados por los símbolos e ir armando el json y xml.
	3. Este caso de uso finaliza cuando se muestra por completo el json y xml uniendo todos los objetos del diagrama de clases
<b>Flujos alternos</b>	

**ELABORADO: DÚVAL CARVAJAL SUÁREZ**

Para cumplir con los requisitos recopilados en la fase anterior se realizaron diagramas de flujo para realizar el análisis pertinente a lo que la librería debe cumplir. Esto permitirá identificar las posibles fallas que se mostrarían al momento de estar utilizando la librería. En la figura 4.13 se observa el algoritmo que permitió identificar los errores en la escritura de las descripciones de los casos de uso utilizando el lenguaje de símbolos.

Figura 4.13

Diagrama de flujo para detectar errores en las descripciones de los casos de uso utilizando un el lenguaje de símbolos.



ELABORADO: DÚVAL CARVAJAL SUÁREZ

### 4.3. Generación de código

En la siguiente fase se desarrollaron las funciones y todos los métodos que fueron necesarios para que la librería funcione correctamente. A continuación se observan todas las variables que fueron necesarias para que todo funcione de manera correcta.



```

1 // Global variables
2 var DataPrimitive_Armadillo = ["String", "string", "byte", "short",
3 "int", "Int", "long", "Long", "float", "Float", "double", "Double",
4 "boolean", "Boolean", "date", "Date", "list", "List", "array",
5 "Array", "arrayList", "ArrayList"];
6 var DataTypeVisibility = [
7   {symbol: "+", text: "public"},
8   {symbol: "-", text: "private"},
9   {symbol: "#", text: "protected"}
10 ];
11 var DataTypeModifiers = [
12   {symbol: "$", text: "static"},
13   {symbol: "?", text: "abstract"},
14   {symbol: "_", text: "final"},
15   {symbol: "@", text: "interface"}
16 ];
17 var DataTypeClass = [
18   {symbol: "@", text: "interface"}
19   , {symbol: "?", text: "abstract"}
20   , {symbol: "_", text: "final"}
21 ];
22 var indexDataType_Armadillo = 0;
23 var DataType_Armadillo = [];
24 var notifications = [];
25 DataType_Armadillo[indexDataType_Armadillo] =
26 DataPrimitive_Armadillo.slice();

```

En el siguiente bloque de código se observa la función que permite acumular los mensajes de error que pueden ocurrir al momento de interpretar las descripciones de los casos de uso.

```

1 /**
2  * @function Function for adding notification messages
3  * @param {number} status
4  * @param {string} information
5  * @param {array} data
6  * */
7 function setNotifications(status, information, data) {
8   let notification =
9   {status: 0, information: "Thank you for using armadillo.js",
10   type: "", data: []};

```

```

11     notification.status = status;
12     notification.information = information;
13     notification.data = data;
14     notification.type =
15     status === 0 ? "information" : status === 1 ? "success"
16         : status === 2 ? "warning" : "error";
17     console.log(notification);
18     notifications.push(notification);
19 }

```

En el siguiente bloque de código se observa una de las funciones principales para detectar los símbolos de apertura y cierre, con el objetivo de utilizar una sola función que permita identificar los símbolos que necesariamente deben ser con apertura y cierre.

```

1  /**
2   * @function Function to validate the start and end character of each use
3   *   case
4   * to be interpreted.
5   * @param {character} beging
6   * @param {character} end
7   * @param {string} characters
8   * */
9  function manualPartition(beging, end, characters) {
10     var subs = {x1: 0, x2: 0};
11     var elements = [];
12     var open = true;
13     let countBegin = count(characters, beging);
14     let countEnd = count(characters, end);
15     if (countBegin === countEnd) {
16         for (var rec = 0; rec < characters.length; rec++) {
17             if (characters[rec] === beging && open === true) {
18                 subs.x1 = rec;
19                 open = false;
20                 if (characters[rec - 1] === "!"
21                     && characters[rec] === "(") {
22                     subs.x1 = rec - 1;
23                 }
24                 if (characters[rec - 1] === "*"
25                     && characters[rec] === "(") {

```

```

25         subs.x1 = rec - 1;
26     }
27     if (characters[rec] === "*"
28         && characters[rec] === "(") {
29         subs.x1 = rec - 2;
30     }
31     if (characters[rec] === "*"
32         && characters[rec] === "¿") {
33         subs.x1 = rec - 1;
34     }
35     if (characters[rec] === "*"
36         && characters[rec] === "¡") { // ¿?
37         subs.x1 = rec - 1;
38     }
39 }
40 if (characters[rec] === end
41     && open === false && subs.x1 !== rec) {
42     open = true;
43     subs.x2 = rec;
44     var resultSelect =
45     characters.toString()
46     .substr(subs.x1, (subs.x2 - (subs.x1 - 1)));
47     elements.push(resultSelect.toString());
48 }
49 }
50 } else {
51     setNotifications(
52         3,
53         "An opening but not a closing symbol was detected."+
54         "> begin: " +begin+", end: "+end+". Near: "
55         +characters+"", []
56     );
57 }
58 return elements;
59 }

```

Para mejorar la presentación de la librería se desarrollo una aplicación web sencilla que permita utilizar de forma grafica las funciones de la misma. La aplicación web tendrá como propósito proporcionar un lugar en la web de donde descargar el script para que pueda ser

usado por la comunidad de desarrolladores. Además de proporcionar documentación de como instalarla en otros proyectos y como utilizarla.

## 4.4. Ejecución de pruebas

Para la ejecución de las pruebas, se analizo el caso de uso que fue proporcionado por el docente. Cada párrafo estaba redactado con el lenguaje de símbolos, por lo que se ingresaron las descripciones en la aplicación de demostración y los resultados fueron los siguientes:

- Descripción #1:

```
1  This use case starts when a person *(person &-id=int) wants to
   register as a tutor *(tutor &-id=int [+userRegistration=Tutor])
   user in the system. *;tutor 1<[register]>n Person!
```

Interpretando el párrafo, se creo una clase denominada Tutor con sus respectivos atributos que son: **id** privado y de tipo String, los métodos que se agregaron son: **userRegistration()** publico. Además se realizo una relación de tipo generalización. A continuación se observa la estructura json y xml generada por la librería a partir del párrafo ingresado.

```
1  {
2    "diagram": [
3      {
4        "packName": "Default",
5        "class": [
6          {
7            "action": "update",
8            "derivative": [],
9            "className": "Person",
10           "visibility": "public",
11           "modifiers": "",
12           "attributes": [
13             {
14               "visibility": "private",
15               "name": "id",
16               "type": "Int"
17             }
18           ]
19         }
20       ]
21     }
22   ]
23 }
```

```

18         ],
19         "methods": [],
20         "constructors": []
21     },
22     {
23         "action": "update",
24         "derivative": [],
25         "className": "Tutor",
26         "visibility": "public",
27         "modifiers": "",
28         "attributes": [
29             {
30                 "visibility": "private",
31                 "name": "id",
32                 "type": "String"
33             }
34         ],
35         "methods": [
36             {
37                 "visibility": "public",
38                 "name": "userRegistration",
39                 "type": "void",
40                 "parameters": []
41             }
42         ],
43         "constructors": []
44     }
45 ],
46 "enums": []
47 }
48 ],
49 "relationships": [
50     {
51         "from": "Tutor",
52         "to": "Person",
53         "typeRelationship": "generalization",
54         "value": "register",
55         "cardinalidate": "1..*",
56         "from_fk": " Person:Person[]",

```

```

57     "to_fk": "tutor :Tutor",
58     "simbol": "1<>n"
59 }
60 ]
61 }

```

```

1  <package>
2    <diagram>
3      <packName>Default</packName>
4      <class>
5        <action>update</action>
6        <className>Person</className>
7        <visibility>public</visibility>
8        <modifiers></modifiers>
9        <attributes>
10         <visibility>private</visibility>
11         <name>id</name>
12         <type>Int</type>
13       </attributes>
14     </class>
15     <class>
16       <action>update</action>
17       <className>Tutor</className>
18       <visibility>public</visibility>
19       <modifiers></modifiers>
20       <attributes>
21         <visibility>private</visibility>
22         <name>id</name>
23         <type>String</type>
24       </attributes>
25       <methods>
26         <visibility>public</visibility>
27         <name>userRegistration</name>
28         <type>void</type>
29       </methods>
30     </class>
31   </diagram>
32   <relations>
33     <from>Tutor</from>

```

```

34         <to>Person</to>
35         <typeRelationship>generalization</typeRelationship>
36         <value>register</value>
37         <cardinalidate>1..*</cardinalidate>
38         <from_fk> Person:Person[]</from_fk>
39         <to_fk>tutor :Tutor</to_fk>
40         <simbol>1<>n
41         </simbol>
42     </relations>
43 </package>

```

Para que la descripción del caso de uso pueda ser entendida por el cliente del sistema que se este desarrollando se obtiene como resultado el párrafo redactado en lenguaje natural omitiendo los símbolos y pueda ser comprendido con mayor facilidad.

```

1      This use case starts when a person wants to register as a tutor
      user in the system.

```

Como el texto esta redactado de forma correcta la retroalimentación generada muestra mensajes exitosos:

**information:** Description entered: This use case starts when a person \*(person &-id=int) wants to register as a tutor \*(tutor &-id=int [+userRegistration=Tutor]) user in the system. \*;tutor 1<[register]>n Person!

**success:** Successfully added attributes: id

**success:** The class Person was generated successfully

**success:** Successfully added attributes: id

**success:** The class Tutor was generated successfully

**success:** The generalization relationship between objects from: Tutor 1<>n to: Person was successfully generated.

Para verificar el funcionamiento de la retroalimentación identificando la mala escritura de las descripciones del caso de uso usando el lenguaje de símbolos, se redacto a propósito el mismo párrafo pero usando de forma incorrecta los símbolos.

```

1      This use case starts when a person *(person &-id=int) wants to
      register as a tutor *(tutor &-id=int [+userRegistration=Tutor])
      user in the system. *;tutor 1<[register]>n Person

```

Observando los mensajes de retroalimentación ya se identificaron que errores se están cometiendo en el uso de algunos símbolos y especificando en que lugar del texto se encuentra ese error.

**information:** Description entered: This use case starts when a person \*(person &-id=int) wants to register as a tutor \*(tutor &-id=int [+userRegistration=Tutor]) user in the system. \*;tutor 1<[register]>n Person!

**success:** Successfully added attributes: id

**success:** The class Person was generated successfully

**success:** Successfully added attributes: id

**error:** An opening but not a closing symbol was detected. => begin: [, end: ]. Near: \*(tutor &-id=int [+userRegistration=Tutor)

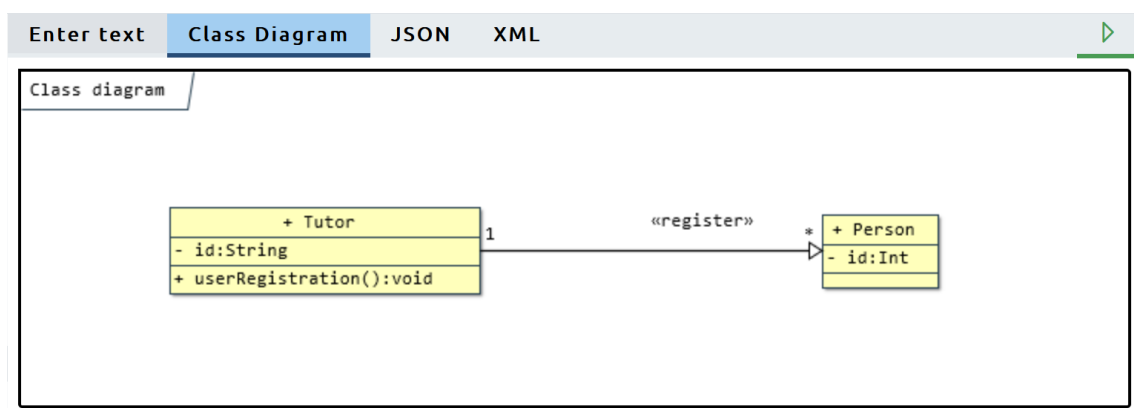
**success:** The class Tutor was generated successfully

**error:** An opening but not a closing symbol was detected. => begin: ;, end: !. Near: This use case starts when a person \*(person &-id=int) wants to register as a tutor \*(tutor &-id=int [+userRegistration=Tutor) user in the system. \*;tutor 1<[register]>n Person

Utilizando la aplicación web de demostración, se logra observar en la figura 4.14 el diagrama de clases creado a partir de la estructura json o xml que la librería proporciona de forma automática.

Figura 4.14

*Diagrama de clases generado por la aplicación web de demostración usando la librería de armadillo.js*



ELABORADO: DÚVAL CARVAJAL SUÁREZ



## 4.5. Evaluación con sistemas de información

Para la ejecución de la fase final de la metodología se revisaron varios proyectos de titulación realizados por estudiantes de la Universidad Técnica Estatal de Quevedo. Se analizaron los sistemas desarrollados reescribiendo las descripciones de los casos de uso utilizando el lenguaje de símbolos.

Tabla 4.13

*Descripción del caso de uso para iniciar sesión.*

<b>Caso de uso:</b>	Iniciar sesión
<b>Actores:</b>	Cuidador
<b>Propósito:</b>	Ingresar a la interfaz principal de la aplicación
<b>Resumen:</b>	Permite identificar las credenciales del usuario que intenta ingresar al sistema, de esa forma verificar el tipo de usuario que ingresa.
<b>Tipo:</b>	Primario
<b>Flujo normal</b>	
1. Este caso de uso inicia el actor cuidador pretende ingresar al sistema.	
2. Ingresa al sistema.	
	3. Muestra la interfaz donde se necesitan ingresar las credenciales de acceso que son un usuario y contraseña.
4. Ingresa sus credenciales de acceso que son el usuario y contraseña.	
	5. Verifica que el usuario y contraseña se encuentren registrados en la aplicación.
	6. El sistema da acceso a la interfaz de inicio de la aplicación.
<b>Flujos alternos</b>	
	5.1 Si el usuario y contraseña no existen en el sistema, se mostrara un mensaje de dialogo notificando lo sucedido. Retornar al paso 4.

**ELABORADO: DÚVAL CARVAJAL SUÁREZ**

**CAPÍTULO V**  
**CONCLUSIONES Y RECOMENDACIONES**

## 5.1. Conclusiones

- Con la utilización de un lenguaje de símbolos que permita estar combinado con las descripciones de los casos de uso escritos en texto natural, además que permita detallar datos técnicos que intervendrán en la generación del sistema informático, logra disminuir un poco el tiempo en la creación de este diagrama de clases. Previamente el desarrollador podrá utilizar la librería en su propio proyecto o utilizar la aplicación web de demostración.
- Para los usuarios que utilizan la herramienta TDDT4IoTS al momento de interpretar las descripciones de los casos de uso con la librería desarrollada, les permite identificar de forma más rápida donde están cometiendo errores al momento de utilizar los símbolos respectivos.
- La aplicación de demostración sobre como funciona la librería servirá para los desarrolladores que necesiten generar un diagrama de clases de una forma rápida dependiendo del sistema informático al que se estén enfrentando. La librería puede ser descargada para que cada desarrolladores personalice la forma en como y con que genere el diagrama de clases, obteniendo una aplicativo que funcione con todos los proyectos que desarrolle a futuro.
- Se logró obtener un solo archivo de tipo .js que puede ser exportado a cualquier tipo de proyecto web. Esto beneficia a muchas aplicaciones que necesiten de la interpretación de los casos de uso que usen los símbolos de una forma rápida y sin tener que instalar programas externos para su correcto funcionamiento.

## 5.2. Recomendaciones

- La librería desarrollada en el lenguaje javascript puede ser fácilmente escalable. Se puede desarrollar un proyecto npm y subirlo al repositorio oficial para que pueda ser utilizado mediante proyectos que utilicen como servidor de aplicaciones Node.js.
- Durante la elaboración de la retroalimentación sobre como utilizar los símbolos para escribir las descripciones de los casos de uso, se pueden realizar trabajos posteriores sobre como identificar la posición exacta del error que se esta cometiendo. Se pueden utilizar una serie de colores que permita identificar los caracteres donde se encuentren las anomalías.
- Para lograr que la librería pueda ser mejorada, se puede pensar en publicar el código fuente en una plataforma web como lo es GitHub y permitir que otros desarrolladores mejoren la librería para llegar a un mayor alcance en la comunidad de programadores a nivel mundial.
- Para lograr un mayor alcance sobre las tecnologías de desarrollo que existen actualmente, y con las que en un futuro serán desarrolladas. Se podrá crear un proyecto que este desplegado en un servidor que este en la nube para crear un servicio web que realice el mismo proceso de la librería, con la ventaja que podra ser utilizado en casi todos los tipos de proyectos que existen. Se podrá obtener los resultados del proyecto de forma clara y remota.

**CAPÍTULO VI**  
**BIBLIOGRAFÍA**

# Referencias

- [1] V. Panthi, A. Tripathi **and** D. P. Mohapatra, “Software validation based on prioritization using concurrent activity diagram,” *International Journal of Systems Assurance Engineering and Management*, **august** 2022, ISSN: 09764348. DOI: 10.1007/S13198-021-01551-8.
- [2] F. Chen, L. Zhang, X. Lian **and** N. Niu, “Automatically recognizing the semantic elements from uml class diagram images,” *Journal of Systems and Software*, **jourvol** 193, **page** 111431, **november** 2022, ISSN: 01641212. DOI: 10.1016/J.JSS.2022.111431. **url:** <https://linkinghub.elsevier.com/retrieve/pii/S0164121222001340>.
- [3] R. Kulesza, M. F. D. Sousa, M. L. M. D. Araújo, C. P. D. Araújo **and** A. M. Filho, “Evolution of web systems architectures: A roadmap,” *Special Topics in Multimedia, IoT and Web Technologies*, **pages** 3–21, **january** 2020. DOI: 10.1007/978-3-030-35102-1\_1/FIGURES/6. **url:** [https://link.springer.com/chapter/10.1007/978-3-030-35102-1\\_1](https://link.springer.com/chapter/10.1007/978-3-030-35102-1_1).
- [4] M. S. Hamdi, A. Ghannem **and** M. Kessentini, “Requirements traceability recovery for the purpose of software reuse: An interactive genetic algorithm approach,” *Innovations in Systems and Software Engineering*, **jourvol** 18, **pages** 193–213, **1 march** 2022, ISSN: 16145054. DOI: 10.1007/S11334-021-00418-2.
- [5] E. Guerra, A. D. O. Dias, L. G. D. Veras, A. Aguiar, J. Choma **and** T. S. D. Silva, “A model to enable the reuse of metadata-based frameworks in adaptive object model architectures,” *IEEE Access*, **jourvol** 9, **pages** 85124–85143, 2021, ISSN: 21693536. DOI: 10.1109/ACCESS.2021.3087795.
- [6] OMG, *About the unified modeling language specification version 2.4*. **url:** <https://www.omg.org/spec/UML/2.4/>.

- [7] G. Bergström, F. Hujainah, T. Ho-Quang **and others**, “Evaluating the layout quality of uml class diagrams using machine learning,” *Journal of Systems and Software*, **page** 111413, **october** 2022, ISSN: 01641212. DOI: 10.1016/J.JSS.2022.111413.
- [8] Omg, “An omg® unified modeling language® publication omg® unified modeling language® (omg uml®) omg document number: Date,” 2009. **url**: <https://www.omg.org/spec/UML/20161101/PrimitiveTypes.xml>.
- [9] M. Jahan, Z. S. H. Abad **and** B. Far, “Generating sequence diagram from natural language requirements,” *Proceedings of the IEEE International Conference on Requirements Engineering*, **jourvol** 2021-September, **pages** 39–48, **september** 2021, ISSN: 23326441. DOI: 10.1109/REW53955.2021.00012.
- [10] F. Losavio, A. Matteo **and** I. Pacilli, “Goal-oriented process for domain analysis using quality standards,” *Enlace*, **jourvol** 6, **pages** 11–28, 3 2009, ISSN: 1690-7515. **url**: [http://ve.scielo.org/scielo.php?script=sci\\_arttext&pid=S1690-75152009000300002&lng=es&nrm=iso&tlng=es](http://ve.scielo.org/scielo.php?script=sci_arttext&pid=S1690-75152009000300002&lng=es&nrm=iso&tlng=es)  
[http://ve.scielo.org/scielo.php?script=sci\\_abstract&pid=S1690-75152009000300002&lng=es&nrm=iso&tlng=es](http://ve.scielo.org/scielo.php?script=sci_abstract&pid=S1690-75152009000300002&lng=es&nrm=iso&tlng=es).
- [11] C. M. Zapata **and** G. González, “Formal ocl specification of consistency rules between the uml class and the use case models and the interfaces model,” *Revista Ingenierías Universidad de Medellín*, **jourvol** 7, 12 **june** 2008, ISSN: 2248-4094. **url**: [http://www.scielo.org.co/scielo.php?script=sci\\_arttext&pid=S1692-33242008000100010](http://www.scielo.org.co/scielo.php?script=sci_arttext&pid=S1692-33242008000100010).
- [12] S. Iqbal, I. Al-Azzoni, G. Allen **and** H. U. Khan, “Extending uml use case diagrams to represent non-interactive functional requirements,” *E-Informatica Software Engineering Journal*, **jourvol** 14, **pages** 97–115, 1 2020, ISSN: 20844840. DOI: 10.37190/E-INF200104.
- [13] E. A. Abdelnabi, A. M. Maatuk **and** M. Hagal, “Generating uml class diagram from natural language requirements: A survey of approaches and techniques,” *2021 IEEE 1st International Maghreb Meeting of the Conference on Sciences and Techniques of Automatic Control and Computer Engineering, MI-STA 2021 - Proceedings*, **pages** 288–293, **may** 2021. DOI: 10.1109/MI-STA52233.2021.9464433.

- [14] H. M. Abu-Dalbouh **and** S. A. Alateyah, “An extension to uml for the modeling of web based bus reservation system,” *Journal of Computer Science*, **jourvol** 16, **pages** 825–837, 7 2020, ISSN: 15526607. DOI: 10 . 3844 / JCSSP . 2020 . 825 . 837.
- [15] G. Guerrero-Ulloa, D. Carvajal-Suárez, G. Brito Casanova, A. Pachay Espinoza, M. J. Hornos **and** C. Rodríguez-Domínguez, *Test-driven development tool for iot-based system*. **url**: <https://aplicaciones.uteq.edu.ec/tddt4iots/>.
- [16] C. G. Moyano, L. Pufahl, I. Weber **and** J. Mendling, “Uses of business process modeling in agile software development projects,” *Information and Software Technology*, **jourvol** 152, **page** 107028, **december** 2022, ISSN: 0950-5849. DOI: 10 . 1016 / J . INF SOF . 2022 . 107028. **url**: <https://linkinghub.elsevier.com/retrieve/pii/S0950584922001483>.
- [17] Z. A. Hamza **and** M. Hammad, “Analyzing uml use cases to generate test sequences,” *International Journal of Computing and Digital Systems*, **jourvol** 10, **pages** 125–134, 1 2021, ISSN: 2210142X. DOI: 10 . 12785 / IJCDS / 100112.
- [18] L. Tan, Z. Yang **and** J. Xie, “Ocl constraints automatic generation for uml class diagram,” *Proceedings 2010 IEEE International Conference on Software Engineering and Service Sciences, ICSESS 2010*, **pages** 392–395, 2010. DOI: 10 . 1109 / ICSESS . 2010 . 5552361.
- [19] L. Chen **and** Y. Zeng, “Automatic generation of uml diagrams from product requirements described by natural language,” *Proceedings of the ASME Design Engineering Technical Conference*, **jourvol** 2, **pages** 779–786, PARTS A AND B **july** 2010. DOI: 10 . 1115 / DETC2009–86514.
- [20] O. S. D. Omer **and** S. Eltyeb, “Towards an automatic generation of uml class diagrams from textual requirements using case-based reasoning approach,” **pages** 1–5, **june** 2022. DOI: 10 . 1109 / ICAID51067 . 2022 . 9799502.
- [21] A. M. Alashqar, “Automatic generation of uml diagrams from scenario-based user requirements,” *Jordanian Journal of Computers and Information Technology (JJCIT)*, **jourvol** 07, 02 **june** 2021.



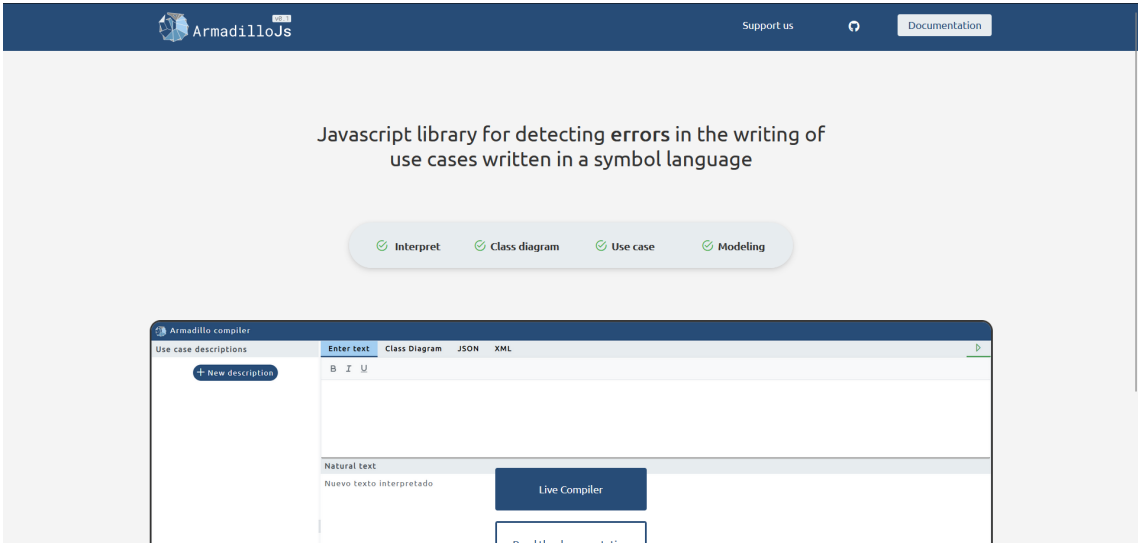
- [22] Shweta **and** R. Sanyal, “Impact of passive and negative sentences in automatic generation of static uml diagram using nlp,” *Journal of Intelligent Fuzzy Systems*, **jourvol** 39, **pages** 2047–2059, 2 **january** 2020, ISSN: 1064-1246. DOI: 10 . 3233 / JIFS – 179871.
- [23] *Manifiesto por el desarrollo ágil de software*. **url:** <https://agilemanifesto.org/iso/es/manifesto.html>.
- [24] N. Mohamed, S. Mazen **and** W. Helmy, “E-ahp: An enhanced analytical hierarchy process algorithm for priotrizing large software requirements numbers,” **page** 2022. **url:** [www.ijacsa.thesai.org](http://www.ijacsa.thesai.org).
- [25] T. N. Kudo, R. D. F. Bulcão-Neto, V. Vicente, G. Neto, Auri **and** M. R. Vincenzi, “Aligning requirements and testing through metamodeling and patterns: Design and evaluation,” **jourvol** 1, **page** 3, DOI: 10 . 1007 / s00766 – 022 – 00377 – 5. **url:** <https://doi.org/10.1007/s00766-022-00377-5>.
- [26] K. Rokis **and** M. Kirikova, “Challenges of low-code/no-code software development: A literature review,” **pages** 3–17, 2022. DOI: 10 . 1007 / 978 – 3 – 031 – 16947 – 2\_1. **url:** [https://link.springer.com/chapter/10.1007/978-3-031-16947-2\\_1](https://link.springer.com/chapter/10.1007/978-3-031-16947-2_1).
- [27] L. Gazzola, L. Mariani, M. Orru, M. Pezze **and** M. Tappler, “Testing software in production environments with data from the field,” *Proceedings - 2022 IEEE 15th International Conference on Software Testing, Verification and Validation, ICST 2022*, **pages** 58–69, 2022. DOI: 10 . 1109 / ICST53961 . 2022 . 00017.
- [28] E Whiting **and** S Datta, “Design and development of a technology-agnostic nfr testing framework introducing the framework and discussing the future of load testing in agile software development,” DOI: 10 . 1145 / 3520084 . 3520092. **url:** <https://doi.org/10.1145/3520084.3520092>.
- [29] P. Bourhis, J. L. Reutter **and** D. Vrgoč, “Json: Data model and query languages,” *Information Systems*, **jourvol** 89, **march** 2020, ISSN: 03064379. DOI: 10 . 1016 / J . IS . 2019 . 101478.
- [30] A. Khalili, “An xml-based approach for geo-semantic data exchange from bim to vr applications,” *Automation in Construction*, **jourvol** 121, **january** 2021, ISSN: 09265805. DOI: 10 . 1016 / J . AUTCON . 2020 . 103425.

- [31] A. Alwabel, “Coedit: A novel error correction mechanism in compilers using spelling correction algorithms,” *Journal of King Saud University - Computer and Information Sciences*, 2021, ISSN: 22131248. DOI: 10.1016/J.JKSUCI.2021.02.010.
- [32] L. Addazi **and** F. Ciccozzi, “Blended graphical and textual modelling for uml profiles: A proof-of-concept implementation and experiment,” *Journal of Systems and Software*, **jourvol** 175, **may** 2021, ISSN: 01641212. DOI: 10.1016/J.JSS.2021.110912.
- [33] C. M. Fonseca, J. P. A. Almeida, G. Guizzardi **and** V. A. Carvalho, “Multi-level conceptual modeling: Theory, language and application,” *Data and Knowledge Engineering*, **jourvol** 134, **july** 2021, ISSN: 0169023X. DOI: 10.1016/J.DATAK.2021.101894.
- [34] M. Shafiq **and** U. Waheed, “Documentation in agile development: A comparative analysis,” *Proceedings of the 21st International Multi Topic Conference, INMIC 2018*, **december** 2018. DOI: 10.1109/INMIC.2018.8595625.

## **CAPÍTULO VII**

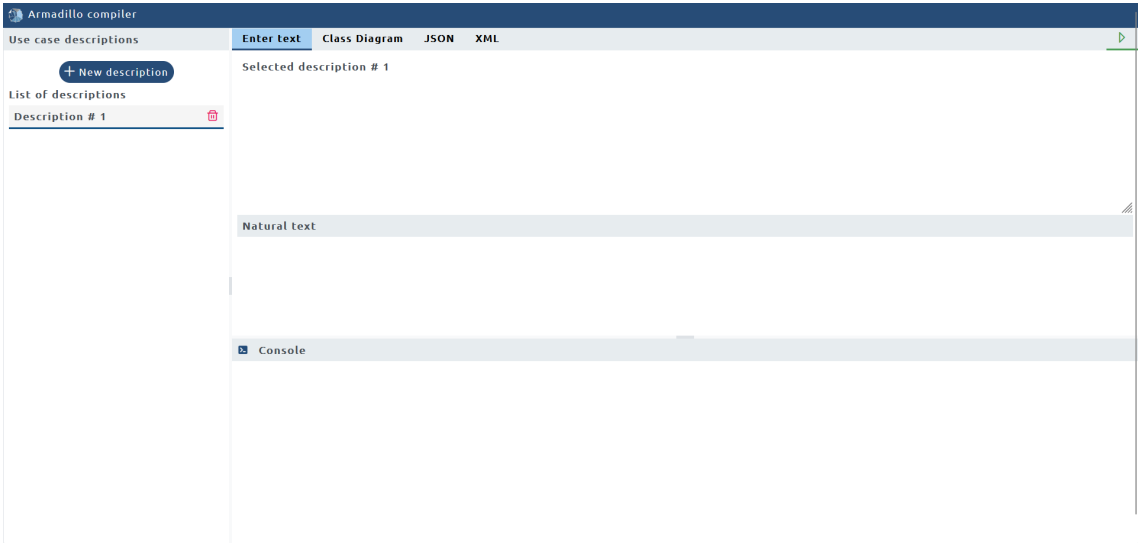
### **ANEXOS**

Figura 6.15  
*Interfaz principal de aplicación demostrativa de la librería.*



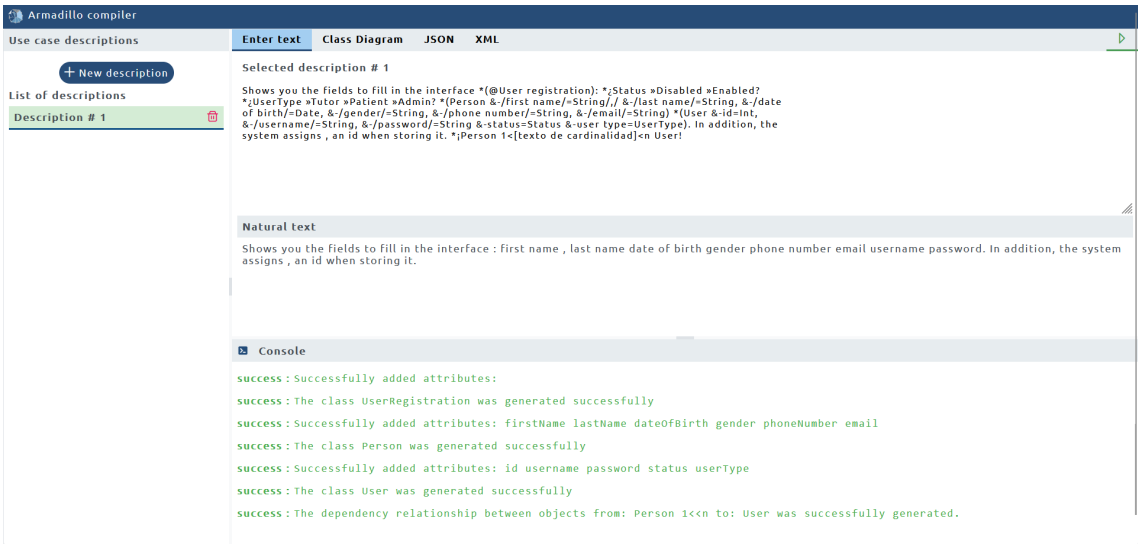
**FUENTE: PROPIA**  
**ELABORADO: DÚVAL CARVAJAL SUÁREZ**

Figura 6.16  
*Interfaz donde se ingresaran las descripciones de los casos de uso a interpretar.*



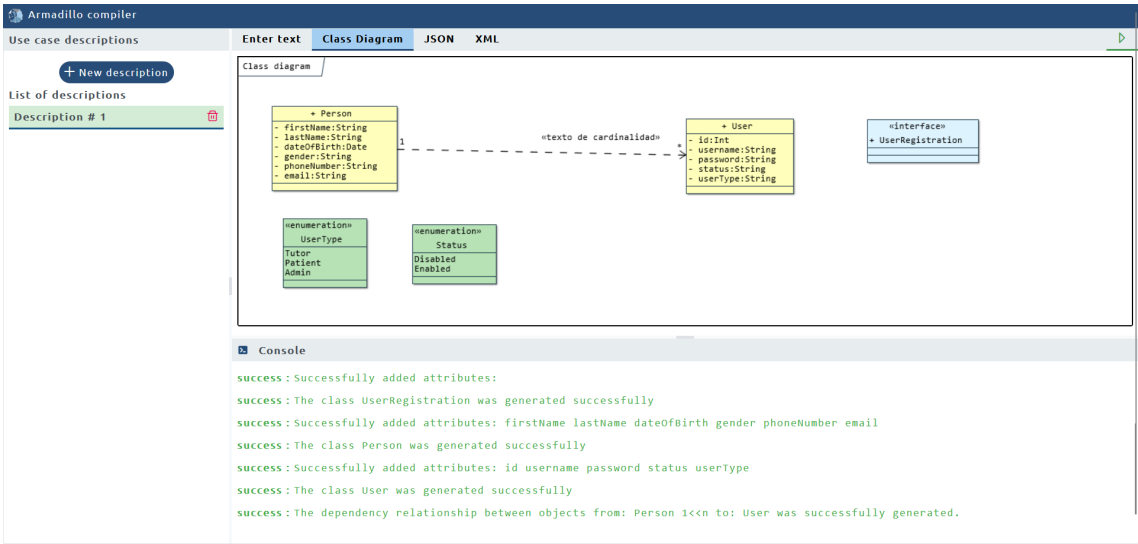
**FUENTE: PROPIA**  
**ELABORADO: DÚVAL CARVAJAL SUÁREZ**

Figura 6.17  
*Descripción del caso de uso interpretada con su respectiva retroalimentación.*



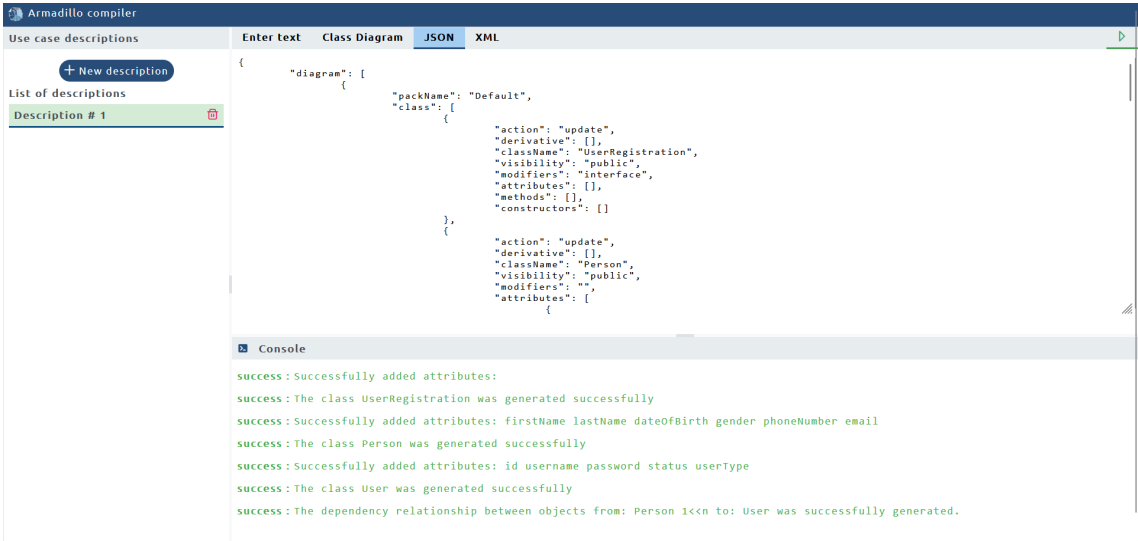
**FUENTE: PROPIA**  
**ELABORADO: DÚVAL CARVAJAL SUÁREZ**

Figura 6.18  
*Diagrama de clases generado por la librería usando una librería externa denominada jsUML2.*



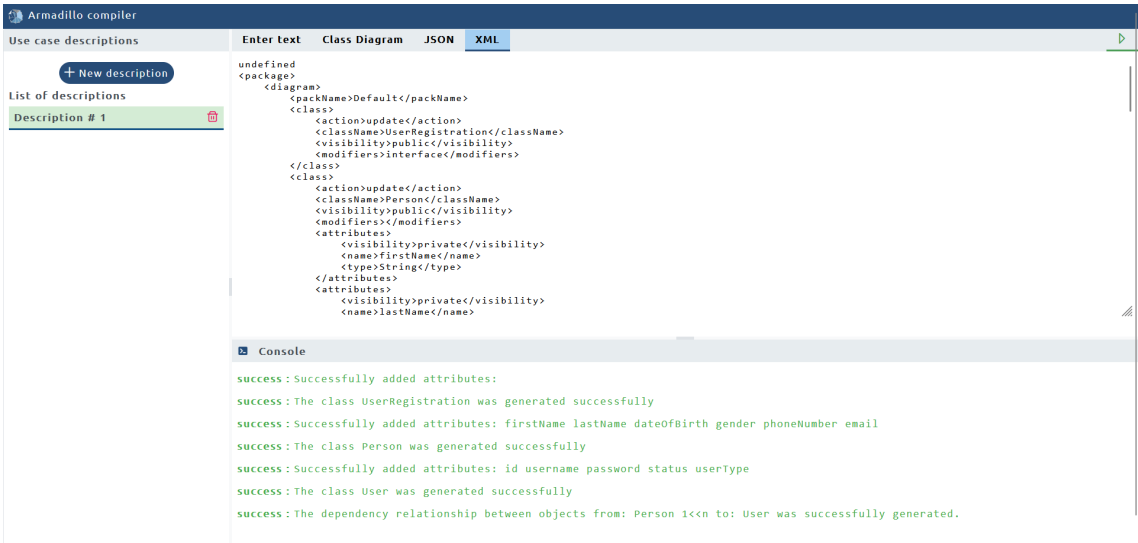
**FUENTE: PROPIA**  
**ELABORADO: DÚVAL CARVAJAL SUÁREZ**

Figura 6.19  
*Estructura JSON generada por la librería.*



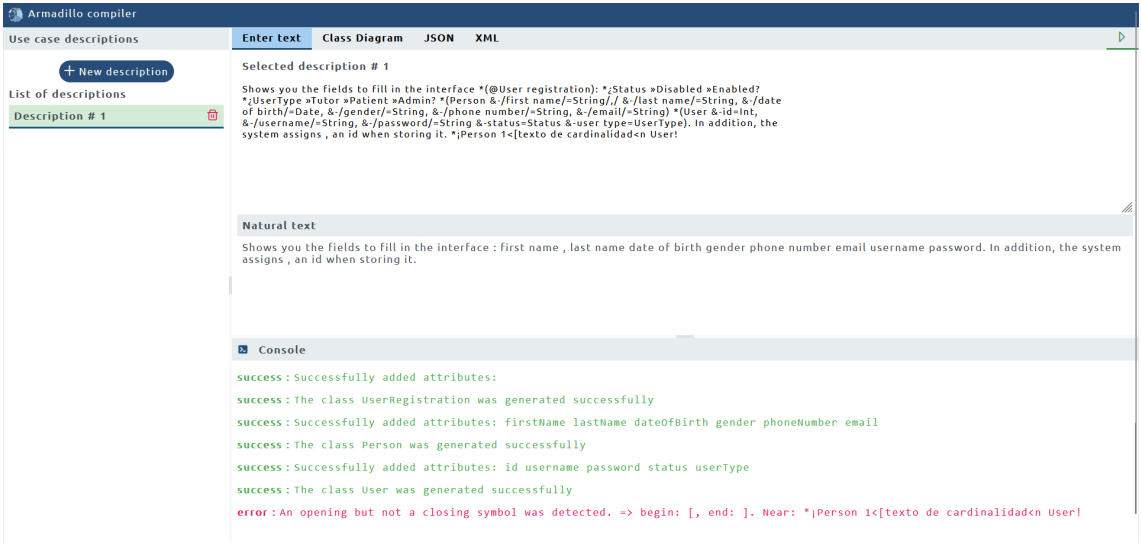
**FUENTE: PROPIA**  
**ELABORADO: DÚVAL CARVAJAL SUÁREZ**

Figura 6.20  
*XML generado por la librería.*



**FUENTE: PROPIA**  
**ELABORADO: DÚVAL CARVAJAL SUÁREZ**

Figura 6.21  
*Error detectado en la descripción del caso de uso*



**FUENTE: PROPIA**  
**ELABORADO: DÚVAL CARVAJAL SUÁREZ**