Esteban Ariza
Johan Giraldo
Mateo Valdés
Juan Restrepo

Engineering Method

Problem description:

Alfonso is a genius biologist who loves to eat various kinds of mushrooms every day. One day he was eating mushrooms while reading the news and he found an article in the Smithsonian Magazine website titled “You May Have Been Eating Mushrooms That Were Unknown to Science” (link to the article: https://www.smithsonianmag.com/smart-news/you-may-have-been-eating-mushrooms-unknown-science-180951974/. For more information about the article, please consult the bibliography included at the end of this document). According to the article, many mushrooms, including some we eat every day, are still unknown to science and may be poisonous or harmful to our health. Since he eats many mushrooms every day, Alfonso got scared and shared the article with his biologist colleagues. His biologist colleagues then asked our team, TWICE Spiderman (a team made up of Esteban Ariza, Johan Sebastián Giraldo, Juan José Restrepo and Mateo Valdés), for help.

The biologists say that we must first read the data already available regarding the mushrooms. Furthermore, the biologists advise us to make graphs and charts about all the available data in order to get a better grasp of the information. They advise us to make graphs of edible vs. poisonous mushroom quantity, the number of mushrooms with a certain odor, number of rings in each of the mushrooms, number of bruises in each of the mushrooms, and the number of mushrooms with a certain cap color. Finally, they want us to use all of the information and determine, given any mushroom’s attributes, whether the mushroom is edible or poisonous. They say that if we do this Alfonso will be able to enjoy mushrooms again without feeling scared and crying.

Identifying the problem:

The biologists want a way to determine if a mushroom is edible or poisonous given its attributes.

We must also create the graphs to represent the following quantities:

- Number of edible vs. poisonous mushrooms
- Number of mushrooms with a certain odor
- Number of mushrooms with a certain amount of rings
- Number of mushrooms with a certain amount of bruises
- Number of mushrooms with a certain cap color

Functional Requirements:

Name	**FR1:** Load data
Summary	The systems loads the data file of the mushrooms with the following attributes: ● Type ● Cap shape ● Cap surface ● Cap color ● Bruises ● Odor ● Gill attachment, ● Gill spacing ● Gill size ● Gill color ● Stalk shape ● Stalk root ● Stalk surface above ring ● Stalk surface below ring ● Stalk color above ring ● Stalk color below ring ● Veil type ● Veil color ● Ring number ● Ring type ● Spore print color ● Population ● Habitat
Input	The data file with all the mushroom attribute data
Output	The file has been loaded successfully

Name	**FR2:** Create the graphs
Summary	The system creates five graphs to represent the following quantities: ● Number of edible vs. poisonous mushrooms ● Number of mushrooms with a certain odor

	- Number of mushrooms with a certain amount of rings - Number of mushrooms with a certain amount of bruises - Number of mushrooms with a certain cap color
Input	The mushroom attribute data
Output	The graphs have been generated successfully

Name	**FR3**: Determine the type of a mushroom
Summary	The system determines the type (possible type: edible and poisonous) of a mushroom based on the following attributes about the mushroom: - Cap shape - Cap surface - Cap color - Bruises - Odor - Gill attachment, - Gill spacing - Gill size - Gill color - Stalk shape - Stalk root - Stalk surface above ring - Stalk surface below ring - Stalk color above ring - Stalk color below ring - Veil type - Veil color - Ring number - Ring type - Spore print color - Population - Habitat
Input	The attributes of the mushroom
Output	The type of the mushroom

<table>
<tr><td>Name:</td><td>FR4: Generate Table</td></tr>
<tr><td>Summary:</td><td>The system creates a table with the following mushroom attributes:

Type
Cap shape
Cap surface
Cap color
Bruises
Odor
Gill attachment,
Gill spacing
Gill size
Gill color
Stalk shape
Stalk root
Stalk surface above ring
Stalk surface below ring
Stalk color above ring
Stalk color below ring
Veil type
Veil color
Ring number
Ring type
Spore print color
Population
Habitat
</td></tr>
<tr><td>Input:</td><td>-</td></tr>
<tr><td>Output:</td><td>The table was created successfully</td></tr>
</table>

<table>
<tr><td>Name:</td><td>FR5: Filter the table</td></tr>
<tr><td>Summary:</td><td>The system allows filtering the table by any field value (mushroom attribute) chosen by the user. The possible fields are:

Type
Cap shape
Cap surface
Cap color
Bruises
Odor
Gill attachment,
Gill spacing
Gill size
Gill color
</td></tr>
</table>

	● Stalk shape ● Stalk root ● Stalk surface above ring ● Stalk surface below ring ● Stalk color above ring ● Stalk color below ring ● Veil type ● Veil color ● Ring number ● Ring type ● Spore print color ● Population ● Habitat
Input:	The attribute and the value
Output:	A table filtered by the value specified

Non-functional requirements

- The program has to be implemented using C#.
- The program has to load a classifiable database (normally used for machine learning) from a file.
- There must be an own implementation of decision trees.
- There must be at least five charts of different types (pie chart, bar chart, scatter chart, etc.).
- The program must show the decision tree in the graphic user interface.

Research:

Decision trees:

Decision trees are “a predictive model that divides the predictor space by grouping observations with similar values for the response or dependent variable” (Merayo, 2020).

The process of these is dividing the space of the predictors, grouping observations with similar values for a response or dependent variable, thus dividing the sample space into subregions, of which a series of rules or decisions applies, so that each subregion has the equal greater quantity of the individuals of one of the populations. If one of the sub-regions contains data different from the classes, it is divided into even smaller sub-regions until they are separated into smaller sub-regions that integrate data from the same class (Merayo, 2020).

Fungi:

Living beings that are part of the Fungi Kingdom, one of the groups in which life is classified, are commonly called fungi. There are more than 144,000 different species of fungi (caracteristicas.co, 2020). In turn, fungi are classified into four large groups: **Saprophytes** (They feed on the decomposition of organic matter), **Mycorrhizae** (Proliferates in plants, exchanging nutrients), **Lichenized** (Product of union with a cyanobacteria), **Parasite** (Poriferan inside the body of other living beings).(Máxima Uriarte,2020). We recognize when a mushroom is edible given its smell or taste, the shape of the hat. One of the edible mushrooms are the white cap mushrooms, characterized by their strong smell, short and thick feet, likewise truffles are another type of edible mushrooms that grow on the roots of plants. Finally we have the poisonous mushrooms, which their species are: Amanita Phalloides, Amanita Muscaria, Cortinarius Orellanus, Lactarius Scrobiculutus, etc. Many species of mushrooms, including some we eat every day, have not been discovered and classified by scientists yet, which, according to scientists, can be a problem because they could be poisonous or harmful to our overall health (Schultz, 2014).

Food and Agriculture Organization of the United Nations (FAO):

FAO is the agency charged with ending hunger. Its objective is to achieve food security for all and at the same time regulate sufficient food (...) with more than 194 member states, the FAO works in more than 130 countries. Finally, Develops international standards to guarantee quality food for all, maintains the largest and most complete statistical database in the world on food and agriculture (Food and Agriculture Organization, 2020).

.NET Framework:

.NET Framework The .NET Framework is a “software development framework for building and running applications on Windows” (Microsoft Corporation, 2020). There are two major components in the .NET Framework. One component is the Common Language Runtime (CLR), which is an execution engine used to run the applications with features such as garbage collections, threading and exception handling; the other component is the Class Library, which contains various APIs and types to perform actions such as reading and writing files and drawing objects on the screen (Microsoft Corporation, 2020).

Kaggle:

Kaggle is “a crowd-sourced platform to attract, nurture, train and challenge data scientists from all around the world to solve data science, machine learning and predictive analytics problems” (Usmani, 2017). The platform has hundreds of thousands of active members from all over the world and receives more than 100 thousand submissions per month (Usmani, 2017). It contains courses, contests, and a huge collection of datasets.

Windows Forms:

Windows Form is a UI Framework that is part of the .NET Framework. It can be used to create desktop applications and various features to process and visualize data; furthermore, it has a visual designer integrated in Visual Studio (Microsoft Corporation, 2020).

State of the art:

There are many ways to tell the difference between edible and poisonous. There are currently a plethora of books and online resources that help people identify poisonous mushrooms. These books and online resources help people classify the mushrooms into five different families: Agarics, Boletes, Milkcaps, Russulas, and Amanitas (Biggane, 2014). Once the family is identified, the person can easily check whether the mushroom is poisonous or not. This method, however, is not 100% accurate, and experts suggest not eating the mushroom unless you absolutely know what family they belong to.

Creative solutions:

<u>Brainstorming:</u>

Alternative A (The expert alternative):

Hire an expert who specializes in mushrooms to collect all of his knowledge and information available to him into a notebook or database, which he can organize and represent graphically in any way he chooses. Then, when we have a mushroom and need to classify it, we contact (in person or online) the expert and he uses his database, judgement and intuition to tell us whether the mushroom is poisonous or edible.

Alternative B (The lab rat alternative):

Obtain all the information about the mushrooms from Kaggle and use Excel to create all the tables and charts needed to solve the problem. Then, when we have a mushroom and need to classify it, we check to see if the mushroom is already in the database. If it is not, we feed the mushroom to a lab rat. If the lab rat dies, we can conclude that the mushroom is poisonous. If the lab rat survives, we can conclude that the mushroom is not poisonous.

Alternative C (Hill Climbing alternative):

Using the information from an online database on the experimentation of edible and toxic mushrooms that is updated every time an experiment is done, using the hill climbing algorithm, given the set of experiments on the experimentation of mushrooms, find a solution on which mushrooms are edible

Alternative D (Java alternative):

Obtain all the information on edible mushrooms from the FAO investigation document, updating it each time new experimentation data appears, classifying them using divide and conquer within an array, whether the mushroom is edible or toxic. Using JavaFX and displaying them in both table and graphs.

Alternative E (SQL alternative):

Using SQL (Server Query Language) as a language, read all the information from an online database about edible and toxic mushrooms, also update it every time an experiment is made to, classify them and connect them to a server to display it on a web page.

Alternative F (C# Decision Tree alternative):

Obtain the database from the University of California Irvine Machine Learning Repository, and then use C# and all available .NET libraries to read and save all the data and display all the graphs. Then create a decision tree algorithm that can determine if a mushroom is poisonous or edible given its attributes.

Alternative G (Human experiment alternative) :

Do an experiment in humans, making them consume the different types of edible and toxic mushrooms, to see their reaction and from there read all the information (updating it each time a new discovery arises in the experimentation) and manually enter the values one by one. Also, classify it given the criterion that the fungus is edible or toxic

Alternative H (Program from scratch alternative):

Obtain the database from Kaggle, then create a program from scratch in any programming language without using any external libraries to create a program that can read the data, display the graphs and use a machine learning algorithm to determine if a mushroom is poisonous or edible. This way, we can have more control over our program and will not have the risk of using external libraries with bugs.

Transition to preliminary designs:

Alternative A (The expert alternative):

This alternative could be very expensive because we might have to hire the expert for a long time. Also, depending on the circumstances, we might even have to hire more than one expert, which will make this alternative even more expensive. Therefore, we must discard this alternative and look for a better one.

Alternative B (The lab rat alternative):

This alternative seems to have many problems. None of us know very much about animals or biology, so we don’t know how to deal with lab rats. We also have to feed them and clean their waste. For these reasons we must discard this alternative and look for a better one.

Alternative C (Hill Climbing alternative):

The hill climbing algorithm is a solid algorithm that can solve the problem we are trying to solve. We must consider this alternative lest we might throw away a perfectly viable choice to help us accomplish our goal.

Alternative D (Java alternative):

Since we all have a solid grasp of the Java programming language and the JavaFX library, this is a very good choice. Java is a great language that can tackle all sorts of problems.Therefore we must consider this alternative.

Alternative E (SQL alternative):

This is a perfectly viable choice because we are dealing with lots of data, and SQL is especially good at managing and obtaining information from large databases. Thus we must consider this alternative.

Alternative F (C# Decision Tree alternative):

This alternative is great because we already have experience with C# and decision trees are perfect for our type of problem, and implementing this is not extremely difficult or time consuming. We must consider this alternative.

Alternative G (Human experiment alternative) :

This alternative is unethical, dangerous, and could land us into legal trouble. We must discard this alternative post-haste.

Alternative H (Program from scratch alternative):

This alternative will give us more control over our application, but it might prove to be too difficult and time consuming. For this reason we will discard this alternative and look for a better one.

Evaluation and selection of the best solution:

The final candidates are the following:

- Alternative C (Hill Climbing alternative)
- Alternative D (Java alternative)
- Alternative E (SQL alternative)
- Alternative F (C# Decision Tree alternative)

We will evaluate them and make our final choice based on the following criterions:

Criterion A: Time consumption

[3] The solution takes a team of four people less than a month to completely implement

[2] The solution takes a team of four people between one month and three months to completely implement

[1] The solution takes a team of four people more than three months to completely implement

Criterion B: Modeling of the problem

[3] The problem can be modeled with the solution very well

[2] The problem can be modeled with the solution

[1] The solution does not model the problem well

Criterion C: Versatility

[3] The solution is very versatile and works in many situations

[2] The solution is moderately versatile and only works in certain situations

[1] The solution is not versatile at all

Criterion D: Difficulty of implementing

[3] The solution is not difficult to implement well

[2] The solution is moderately difficult to implement well

[1] The solution is extremely difficult to implement well

Alternative	Criterion A (Time consumption)	Criterion B (Modeling of the problem)	Criterion C (Versatility)	Criterion D (Difficulty)	Total
C (Hill Climbing solution)	2	3	2	2	9
D (Java solution)	2	1	2	2	7
E (SQL solution)	2	2	2	2	8
F (C# Decision Tree solution)	2	3	3	2	10

Justification:

Alternative C (Hill Climbing solution): This solution gets a very high score (9 out of 12) because the hill climbing algorithm can model our problem very well, is moderately versatile, and is not extremely difficult or time consuming to implement. The only problem is that, while the C# Decision Tree solution can generate a decision tree that we can actually see and show to our client, this solution in particular is difficult to accurately visualize and obtain all the information about how it got to its result.

Alternative D (Java solution): This solution gets a medium score (7 out of 12) because, even though it is not too difficult or time consuming, it does not model the problem well. Java is too verbose and lacks many important features that C# has that can help us solve this problem. When compared to the other solutions, Java does not seem to be the right tool for the job.

Alternative E (SQL solution): This solution gets a medium-high score (8 out of 12) because SQL is great for creating large databases and handling all sorts of data. SQL is not hard to use and creating a database using SQL will not take very long in most cases. The problem is that, while we are dealing with a lot of data, we also need to create powerful algorithms and use external libraries that are found in programming languages such as C# or Python and not in SQL.

Alternative F (C# Decision Tree solution): This solution gets a very high score (10 out of 12) because it models the problem extremely well, it is very versatile, and it can be completely implemented in less than 3 months without much difficulty. An added bonus is that the decision tree can be visualized and shown to the client so that he understands each step required to classify the mushroom.

The C# Decision Tree solution wins.

Bibliography:

Biggane, Eric (2014). How to Tell the Difference Between Poisonous and Edible Mushrooms. Retrieved from https://www.wildfooduk.com/articles/how-to-tell-the-difference-between-poisonous-and-edible-mushrooms/

Food and Agriculture Organization (2020). About FAO. Retrieved from https://www.fao.org/about/es/

Máxima, P. (2020). Information and characteristics of fungi. Retrieved from https://www.caracteristicas.co/hongos/

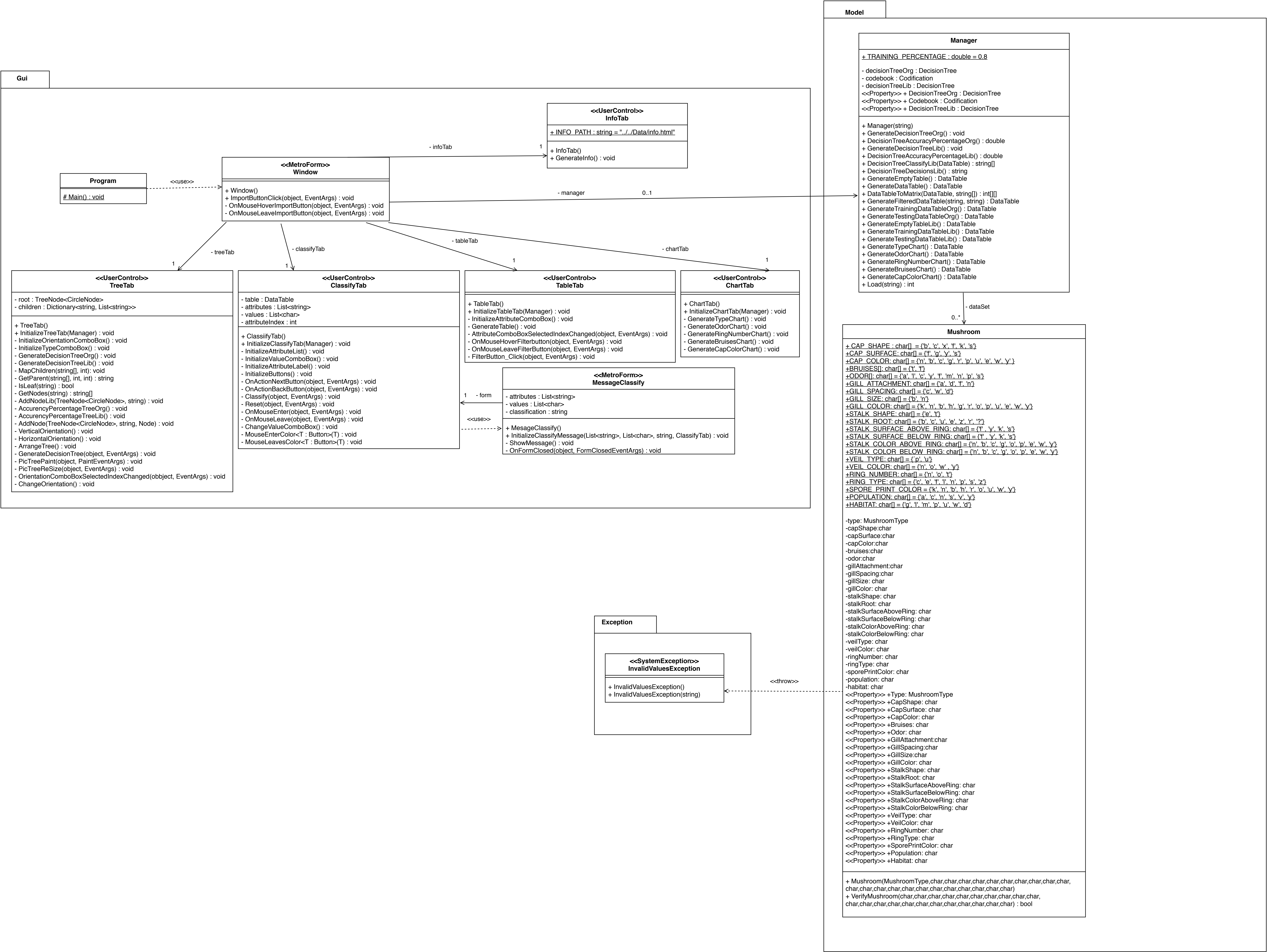
Merayo, P. (2020). What are decision trees and what are they for? Retrieved From https://www.maximaformacion.es/blog-dat/que-son-los-arboles-de-decision-y-para-que-sirven/

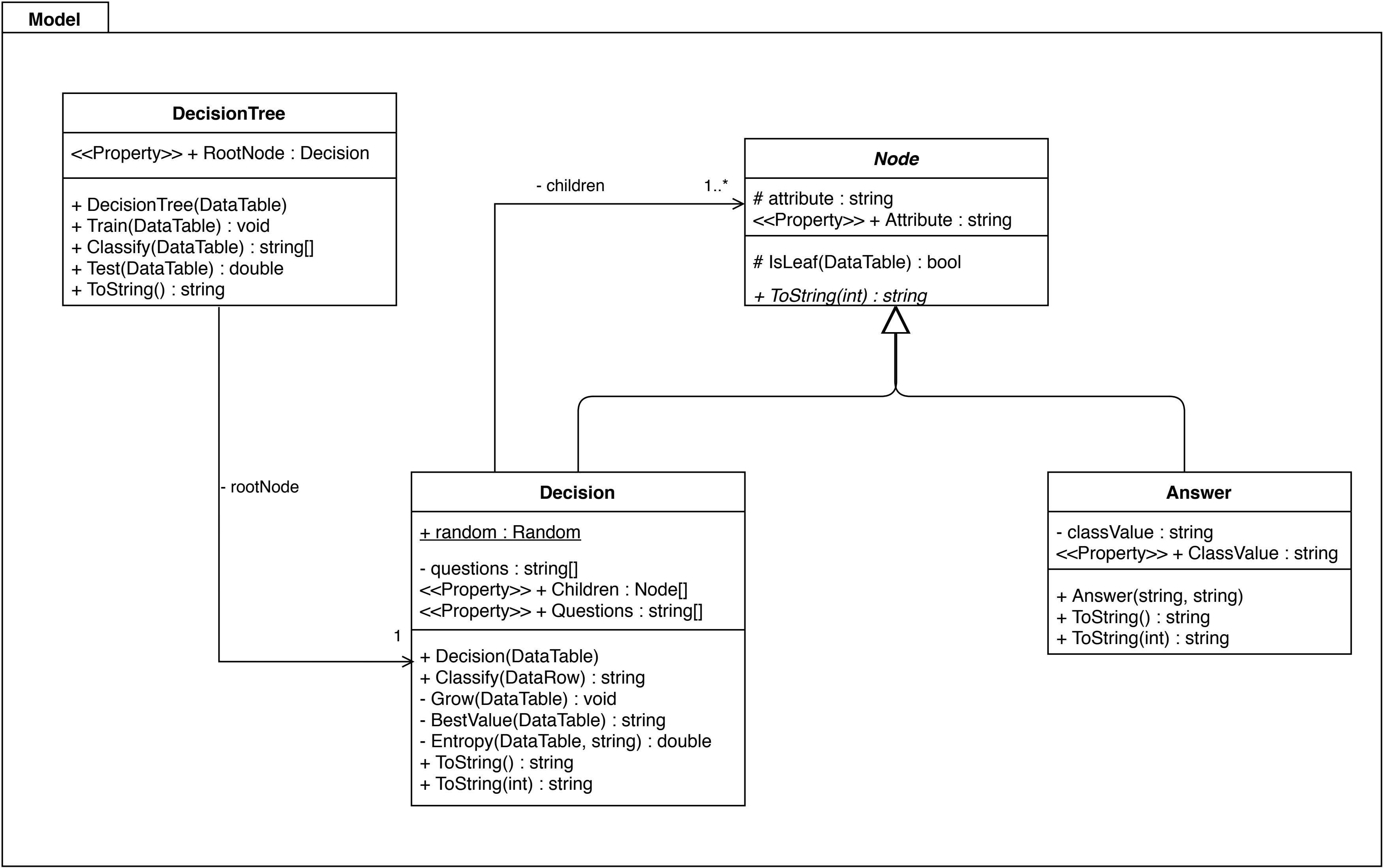
Microsoft Corporation (2020). What Is The .NET Framework? Retrieved from https://dotnet.microsoft.com/learn/dotnet/what-is-dotnet-framework

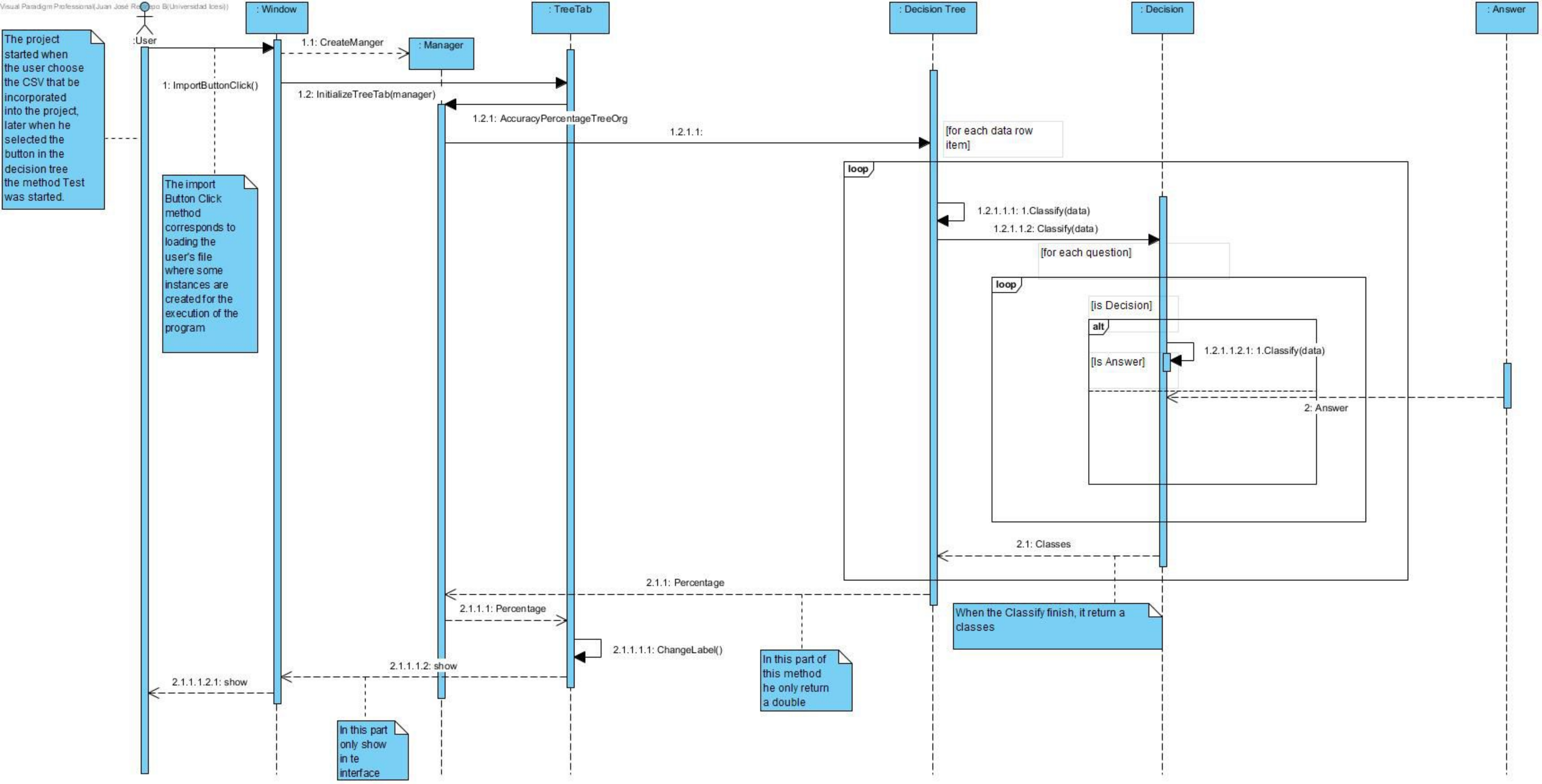
Microsoft Corporation (2020). Windows Forms for .NET 5 and .NET Core 3.1. Retrieved from https://docs.microsoft.com/en-us/dotnet/desktop/winforms/?view=netdesktop-5.0

Schultz, C. (2014). You May Have Been Eating Mushrooms That Were Unknown to Science. Retrieved from https://www.smithsonianmag.com/smart-news/you-may-have-been-eating-mushrooms-unknown-science-180951974/

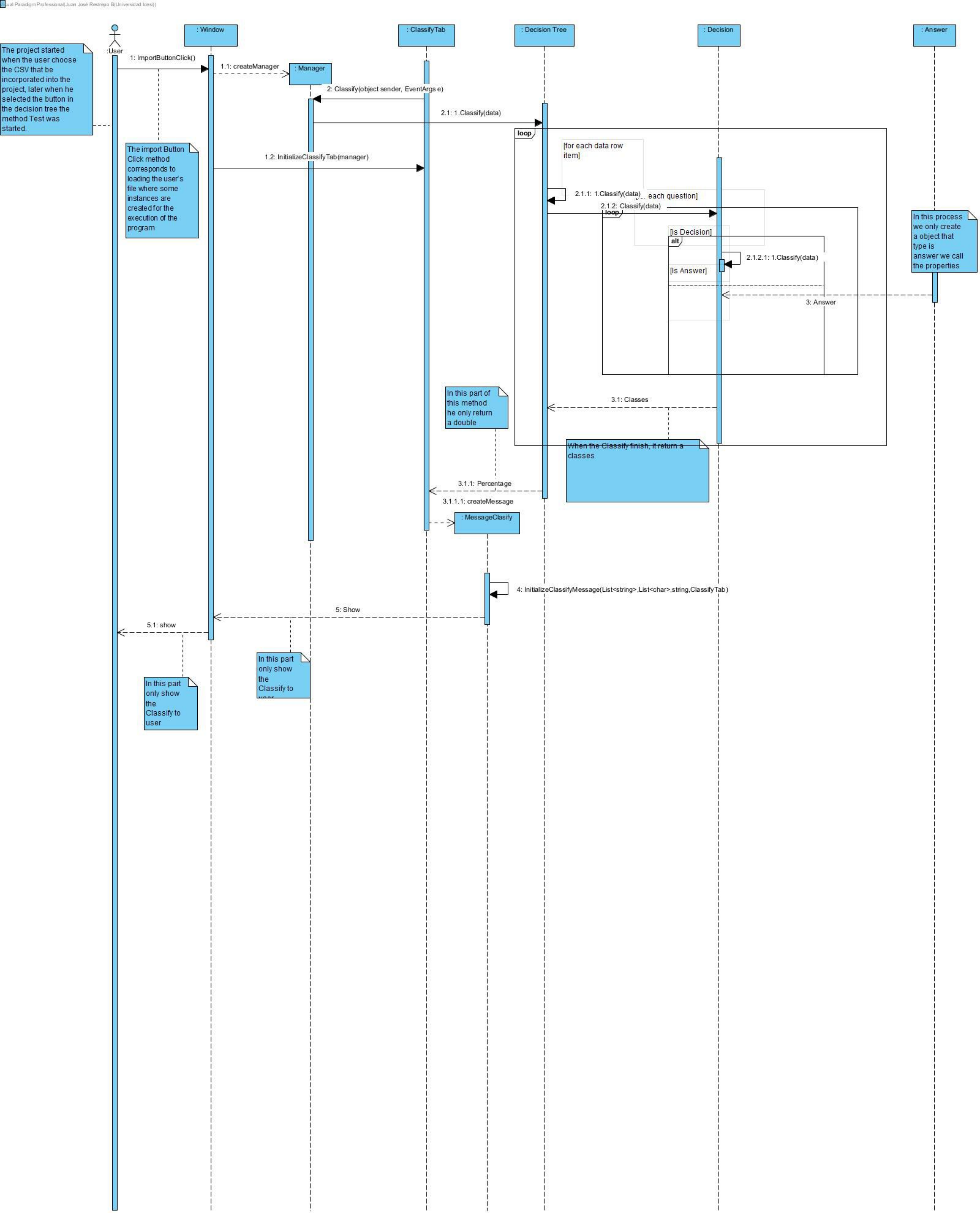
Uzman, Z. (2017). What is Kaggle, Why I Participate, What is the Impact? Retrieved from https://www.kaggle.com/getting-started/44916

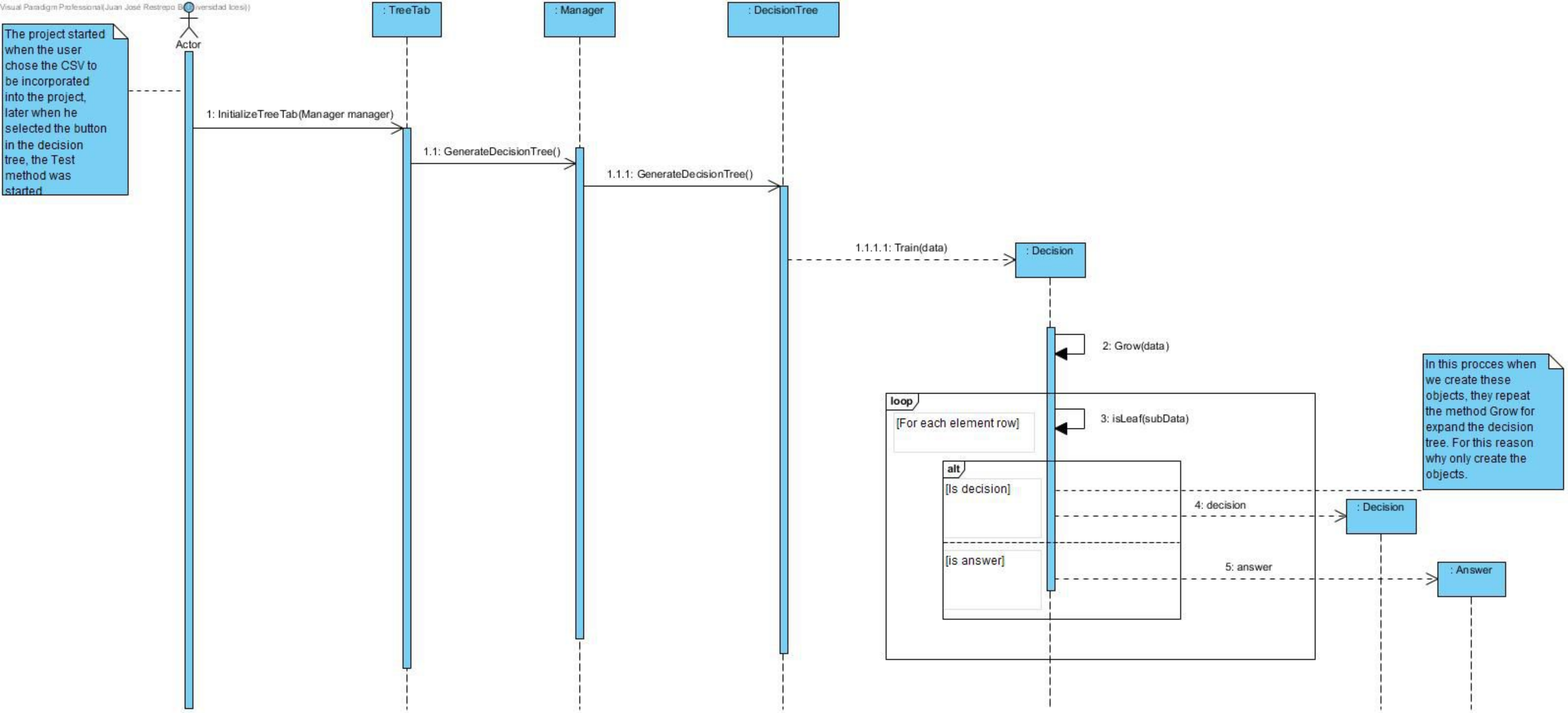

Model
Manager
+ TRAINING_PERCENTAGE : double = 0.8
- decisionTreeOrg : DecisionTree
- codebook : Codification
- decisionTreeLib : DecisionTree
<<Property>> + DecisionTreeOrg : DecisionTree
<<Property>> + Codebook : Codification
<<Property>> + DecisionTreeLib : DecisionTree
+ Manager(string)
+ GenerateDecisionTreeOrg() : void
+ DecisionTreeAccuracyPercentageOrg() : double
+ GenerateDecisionTreeLib() : void
+ DecisionTreeAccuracyPercentageLib() : double
+ DecisionTreeClassifyLib(DataTable) : string[]
+ DecisionTreeDecisionsLib() : string
+ GenerateEmptyTable() : DataTable
+ GenerateDataTable() : DataTable
+ DataTableToMatrix(DataTable, string[]) : int[][]
+ GenerateFilteredDataTable(string, string) : DataTable
+ GenerateTrainingDataTableOrg() : DataTable
+ GenerateTestingDataTableOrg() : DataTable
+ GenerateEmptyTableLib() : DataTable
+ GenerateTrainingDataTableLib() : DataTable
+ GenerateTestingDataTableLib() : DataTable
+ GenerateTypeChart() : DataTable
+ GenerateOdorChart() : DataTable
+ GenerateRingNumberChart() : DataTable
+ GenerateBruisesChart() : DataTable
+ GenerateCapColorChart() : DataTable
+ Load(string) : int
- dataSet
0..*
Mushroom
+ CAP_SHAPE : char[] = {'b', 'c', 'x', 'f', 'k', 's'}
+CAP_SURFACE: char[] = {'f', 'g', 'y', 's'}
+CAP_COLOR: char[] = {'n', 'b', 'c', 'g', 'r', 'p', 'u', 'e', 'w', 'y' }
+BRUISES[]: char[] = {'t', 'f'}
+ODOR[]: char[] = {'a', 'l', 'c', 'y', 'f', 'm', 'n', 'p', 's'}
+GILL_ATTACHMENT: char[] = {'a', 'd', 'f', 'n'}
+GILL_SPACING: char[] = {'c', 'w', 'd'}
+GILL_SIZE: char[] = {'b', 'n'}
+GILL_COLOR: char[] = {'k', 'n', 'b', 'h', 'g', 'r', 'o', 'p', 'u', 'e', 'w', 'y'}
+STALK_SHAPE: char[] = {'e', 't'}
+STALK_ROOT: char[] = {'b', 'c', 'u', 'e', 'z', 'r', '?'}
+STALK_SURFACE_ABOVE_RING: char[] = {'f' , 'y', 'k', 's'}
+STALK_SURFACE_BELOW_RING: char[] = {'f' , 'y', 'k', 's'}
+STALK_COLOR_ABOVE_RING: char[] = {'n', 'b', 'c', 'g', 'o', 'p', 'e', 'w', 'y'}
+STALK_COLOR_BELOW_RING: char[] = {'n', 'b', 'c', 'g', 'o', 'p', 'e', 'w', 'y'}
+VEIL_TYPE: char[] = { 'p', 'u'}
+VEIL_COLOR: char[] = {'n', 'o', 'w' , 'y'}
+RING_NUMBER: char[] = {'n', 'o', 't'}
+RING_TYPE: char[] = {'c', 'e', 'f', 'l', 'n', 'p', 's', 'z'}
+SPORE_PRINT_COLOR = {'k', 'n', 'b', 'h', 'r', 'o', 'u', 'w', 'y'}
+POPULATION: char[] = {'a', 'c', 'n', 's', 'v', 'y'}
+HABITAT: char[] = {'g', 'l', 'm', 'p', 'u', 'w', 'd'}
-type: MushroomType
-capShape:char
-capSurface:char
-capColor:char
-bruises:char
-odor:char
-gillAttachment:char
-gillSpacing:char
-gillSize: char
-gillColor: char
-stalkShape: char
-stalkRoot: char
-stalkSurfaceAboveRing: char
-stalkSurfaceBelowRing: char
-stalkColorAboveRing: char
-stalkColorBelowRing: char
-veilType: char
-veilColor: char
-ringNumber: char
-ringType: char
-sporePrintColor: char
-population: char
-habitat: char
<<Property>> +Type: MushroomType
<<Property>> +CapShape: char
<<Property>> +CapSurface: char
<<Property>> +CapColor: char
<<Property>> +Bruises: char
<<Property>> +Odor: char
<<Property>> +GillAttachment:char
<<Property>> +GillSpacing:char
<<Property>> +GillSize:char
<<Property>> +GillColor: char
<<Property>> +StalkShape: char
<<Property>> +StalkRoot: char
<<Property>> +StalkSurfaceAboveRing: char
<<Property>> +StalkSurfaceBelowRing: char
<<Property>> +StalkColorAboveRing: char
<<Property>> +StalkColorBelowRing: char
<<Property>> +VeilType: char
<<Property>> +VeilColor: char
<<Property>> +RingNumber: char
<<Property>> +RingType: char
<<Property>> +SporePrintColor: char
<<Property>> +Population: char
<<Property>> +Habitat: char
+ Mushroom(MushroomType,char,char,char,char,char,char,char,char,char,char, char,char,char,char,char,char,char,char,char,char,char,char)
+ VerifyMushroom(char,char,char,char,char,char,char,char,char,char,char, char,char,char,char,char,char,char,char,char,char,char) : bool
Gui
Program
Main() : void
<<use>>
<<MetroForm>>
Window
+ Window()
+ ImportButtonClick(object, EventArgs) : void
- OnMouseHoverImportButton(object, EventArgs) : void
- OnMouseLeaveImportButton(object, EventArgs) : void
- infoTab
1
<<UserControl>>
InfoTab
+ INFO_PATH : string = "../../Data/info.html"
+ InfoTab()
+ GenerateInfo() : void
- manager
0..1
- treeTab
1
- classifyTab
1
- tableTab
1
- chartTab
1
<<UserControl>>
TreeTab
- root : TreeNode<CircleNode>
- children : Dictionary<string, List<string>>
+ TreeTab()
+ InitializeTreeTab(Manager) : void
- InitializeOrientationComboBox() : void
- InitializeTypeComboBox() : void
- GenerateDecisionTreeOrg() : void
- GenerateDecisionTreeLib() : void
- MapChildren(string[], int): void
- GetParent(string[], int, int) : string
- IsLeaf(string) : bool
- GetNodes(string) : string[]
- AddNodeLib(TreeNode<CircleNode>, string) : void
- AccurencyPercentageTreeOrg() : void
- AccurencyPercentageTreeLib() : void
- AddNode(TreeNode<CircleNode>, string, Node) : void
- VerticalOrientation() : void
- HorizontalOrientation() : void
- ArrangeTree() : void
- GenerateDecisionTree(object, EventArgs) : void
- PicTreePaint(object, PaintEventArgs) : void
- PicTreeReSize(object, EventArgs) : void
- OrientationComboBoxSelectedIndexChanged(obbject, EventArgs) : void
- ChangeOrientation() : void
<<UserControl>>
ClassifyTab
- table : DataTable
- attributes : List<string>
- values : List<char>
- attributeIndex : int
+ ClassiifyTab()
+ InitializeClassifyTab(Manager) : void
- InitializeAttributeList() : void
- InitializeValueComboBox() : void
- InitializeAttributeLabel() : void
- InitializeButtons() : void
- OnActionNextButton(object, EventArgs) : void
- OnActionBackButton(object, EventArgs) : void
- Classify(object, EventArgs) : void
- Reset(object, EventArgs) : void
- OnMouseEnter(object, EventArgs) : void
- OnMouseLeave(object, EventArgs) : void
- ChangeValueComboBox() : void
- MouseEnterColor<T : Button>(T) : void
- MouseLeavesColor<T : Button>(T) : void
<<UserControl>>
TableTab
+ TableTab()
+ InitializeTableTab(Manager) : void
- InitializeAttributeComboBox() : void
- GenerateTable() : void
- AttributeComboBoxSelectedIndexChanged(object, EventArgs) : void
- OnMouseHoverFilterbutton(object, EventArgs) : void
- OnMouseLeaveFilterButton(object, EventArgs) : void
- FilterButton_Click(object, EventArgs) : void
<<UserControl>>
ChartTab
+ ChartTab()
+ InitializeChartTab(Manager) : void
- GenerateTypeChart() : void
- GenerateOdorChart() : void
- GenerateRingNumberChart() : void
- GenerateBruisesChart() : void
- GenerateCapColorChart() : void
1
- form
<<use>>
<<MetroForm>>
MessageClassify
- attributes : List<string>
- values : List<char>
- classification : string
+ MesageClassify()
+ InitializeClassifyMessage(List<string>, List<char>, string, ClassifyTab) : void
- ShowMessage() : void
- OnFormClosed(object, FormClosedEventArgs) : void
Exception
<<SystemException>>
InvalidValuesException
+ InvalidValuesException()
+ InvalidValuesException(string)
<<throw>>

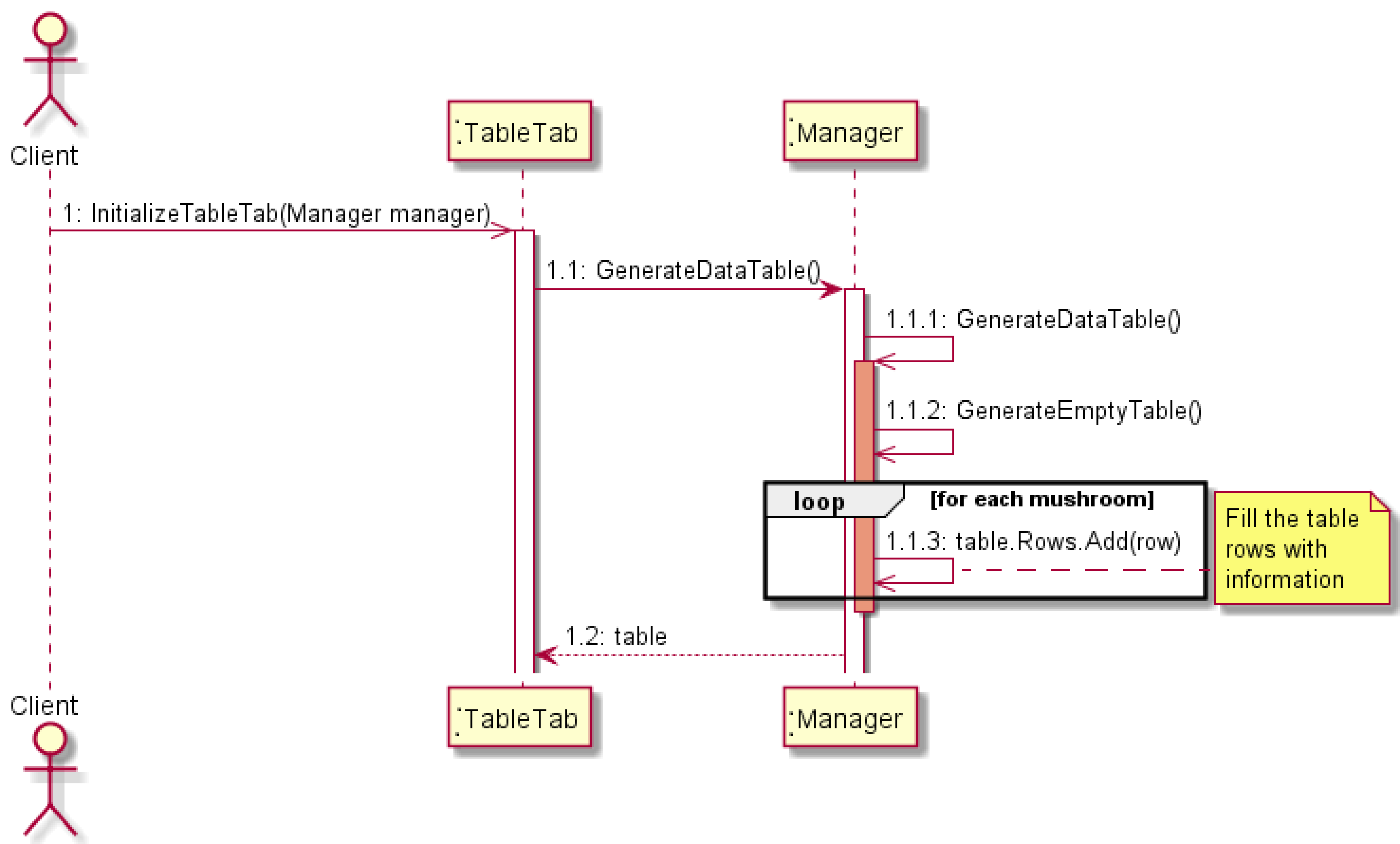

Model
DecisionTree
<<Property>> + RootNode : Decision
+ DecisionTree(DataTable)
+ Train(DataTable) : void
+ Classify(DataTable) : string[]
+ Test(DataTable) : double
+ ToString() : string
- children
1..*
Node
attribute : string
<<Property>> + Attribute : string
IsLeaf(DataTable) : bool
+ ToString(int) : string
- rootNode
1
Decision
+ random : Random
- questions : string[]
<<Property>> + Children : Node[]
<<Property>> + Questions : string[]
+ Decision(DataTable)
+ Classify(DataRow) : string
- Grow(DataTable) : void
- BestValue(DataTable) : string
- Entropy(DataTable, string) : double
+ ToString() : string
+ ToString(int) : string
Answer
- classValue : string
<<Property>> + ClassValue : string
+ Answer(string, string)
+ ToString() : string
+ ToString(int) : string

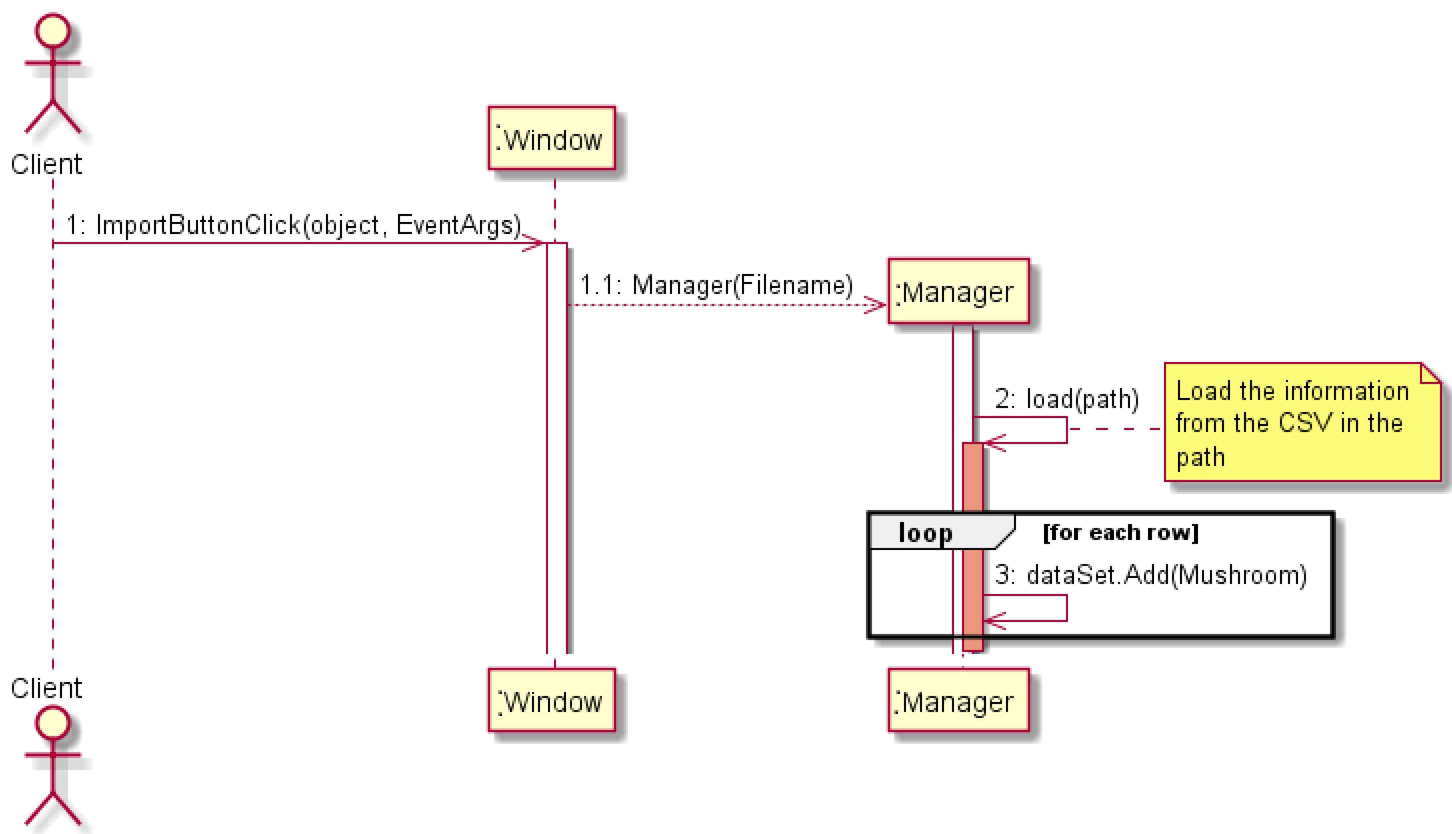


Visual Paradigm Professional(Juan José Restrepo B(Universidad Icesi))
: Window
: TreeTab
: Decision Tree
: Decision
: Answer
:User
: Manager
The project started when the user choose the CSV that be incorporated into the project, later when he selected the button in the decision tree the method Test was started.
1.1: CreateManger
1: ImportButtonClick()
1.2: InitializeTreeTab(manager)
1.2.1: AccuracyPercentageTreeOrg
1.2.1.1:
[for each data row item]
The import Button Click method corresponds to loading the user's file where some instances are created for the execution of the program
loop
1.2.1.1.1: 1.Classify(data)
1.2.1.1.2: Classify(data)
[for each question]
loop
[is Decision]
alt
1.2.1.1.2.1: 1.Classify(data)
[Is Answer]
2: Answer
2.1: Classes
2.1.1: Percentage
2.1.1.1: Percentage
When the Classify finish, it return a classes
2.1.1.1.1: ChangeLabel()
In this part of this method he only return a double
2.1.1.1.2: show
2.1.1.1.2.1: show
In this part only show in te interface


ual Paradigm Professional(Juan José Restrepo B(Universidad Icesi))
:User
: Window
: ClassifyTab
: Decision Tree
: Decision
: Answer
The project started when the user choose the CSV that be incorporated into the project, later when he selected the button in the decision tree the method Test was started.
1: ImportButtonClick()
1.1: createManager
: Manager
2: Classify(object sender, EventArgs e)
2.1: 1.Classify(data)
loop
[for each data row item]
The import Button Click method corresponds to loading the user's file where some instances are created for the execution of the program
1.2: InitializeClassifyTab(manager)
2.1.1: 1.Classify(data)
each question]
2.1.2: Classify(data)
loop
[is Decision]
alt
2.1.2.1: 1.Classify(data)
[Is Answer]
In this process we only create a object that type is answer we call the properties
3: Answer
In this part of this method he only return a double
3.1: Classes
When the Classify finish, it return a classes
3.1.1: Percentage
3.1.1.1: createMessage
: MessageClasify
4: InitializeClassifyMessage(List<string>,List<char>,string,ClassifyTab)
5: Show
5.1: show
In this part only show the Classify to user
In this part only show the Classify to user


The project started when the user chose the CSV to be incorporated into the project, later when he selected the button in the decision tree, the Test method was started
Actor
: TreeTab
: Manager
: DecisionTree
1: InitializeTreeTab(Manager manager)
1.1: GenerateDecisionTree()
1.1.1: GenerateDecisionTree()
1.1.1.1: Train(data)
: Decision
2: Grow(data)
loop
[For each element row]
3: isLeaf(subData)
alt
[Is decision]
4: decision
: Decision
[is answer]
5: answer
: Answer
In this procces when we create these objects, they repeat the method Grow for expand the decision tree. For this reason why only create the objects.


Client
:TableTab
:Manager
1: InitializeTableTab(Manager manager)
1.1: GenerateDataTable()
1.1.1: GenerateDataTable()
1.1.2: GenerateEmptyTable()
loop
[for each mushroom]
1.1.3: table.Rows.Add(row)
Fill the table rows with information
1.2: table
Client
:TableTab
:Manager


Client
:Window
1: ImportButtonClick(object, EventArgs)
1.1: Manager(Filename)
:Manager
2: load(path)
Load the information from the CSV in the path
loop
[for each row]
3: dataSet.Add(Mushroom)
Client
:Window
:Manager