CS5351 - Project / 1920A

Group 3, Online UML Class Diagram Drawing Tool

LAW Chun Kit (54113847), YIP Chi Ming (50859156), TSANG Ching Wai (55040990), YIU Wai Chung (55194087), HO Kin Leung (55638430), CHOR Siu Man (53707719), WONG Hoi Man (55866609)

**Abstract—**In order to reduce the effort when documenting software projects, this project aims at developing a text-based input tool for drawing Class Diagrams. With the help of agile development model, a team of seven has successfully designed and built the tool in 10 weeks. The system is composed of two major modules: text parsing and diagram visualization. The team is split into two sub-teams to implement the modules in parallel. This report shows the project plan and the detail of the development flow. It includes the proposed system architecture, design, work breakdown structure, project schedule and test plans.

—————————— ◆ ——————————

# INTRODUCTION

S

oftware Engineering (SE) - the systematic approach towards developing maintainable software. It requires continuous and on-going effort to maintain the system documentation so as to facilitate system maintenance and future system enhancements. In order to provide a clear visualization of the design of a system, Unified Modeling Language (UML) is commonly used to model the design in the System Analysis and Design (SA&D) stage of a system.

UML is a standard way for system documentation. Nonetheless, most of the UML tools on the market are of beautiful GUI which requires repetitive drag-and-drop to draw the diagrams. Although online UML tools are sometimes provided, it takes time to produce different types of UML diagram. In order to facilitate developers to draw a UML diagram in a timely manner, we are inspired to build a simple tool to facilitate the UML diagram drawing process.

When conducting research into existing SE tools, we found an online sequence diagram drawing tool (<https://sequencediagram.org/>). Using text input to generate sequence diagram is much faster than repetitively drag and drop in GUI. Developers are usually more familiar with using command-line interface (CLI). Thus, text input will be a more efficient way to draft UML diagrams during the SA&D stage.

When conducting research into existing SE tools, we found an online sequence diagram drawing tool (<https://sequencediagram.org/>). Using text input to generate sequence diagram is much faster than repetitively drag and drop in GUI. Developers are usually more familiar with using command-line interface (CLI). Thus, text input will be a more efficient way to draft UML diagrams during the SA&D stage.

The aforementioned considerations are reviewed and assembled into our proposed solution. We choose the most vital diagram, Class Diagram, as our project target. Class Diagrams not only shows the system design, but also enforces object oriented programming, which minimizes the efforts for system maintenance and future upgrade.

# RELATED WORK

A

lthough there are different online SE tools on the market, only a few of them are for generating UML diagrams, and even less among them can generate multiple diagram types.

We found that the free online tools, such as Regular Expression Tester, SHA key generation, Base64 decoder, JSON editor etc, are helpful in the implementation phase. However, we found only one free sequence diagram tool that is dedicated for the design phase in software development. Although Visual Paradigm, which is a globally used enterprise solution, recently joint the web-based applications galaxy, it only provides GUI which we believe that is not the most efficient way for developers to create UML diagrams.

[SequenceDiagram.org](https://sequencediagram.org/) has demonstrated an effective solution to cope with the problems. We decided to build a similar tool that generates Class Diagram from text-input, so as to gradually provide a comprehensive solution on top of the online sequence diagram.

We also aim at enhancing the versatility by introducing more commonly used features. For example, direct printing, direct mail, result exportation etc.

# Preliminaries

T

he drawing tool we developed is based on the structure of UML class diagram. UML stands for Unified Modeling Language which aims to help software developer to visualize the architecture of the software system including classes, attributes, types, modifiers and the relation among the different diagrams. For those object oriented software project, it is an important port on the design phase to express the objects in graphical view and do standardized the input for further validation.

Multi-tier system architecture helps modularize the system design into layers. The components would be loosely coupled such that each of them could be tested separately, and be developed parallelly with less impact. It provides a model for application in which the developer can more easily reuse program codes with a flexible manner.

Model-View-Control (MVC), one of the most commonly used multi-tier system architectures, is adopted in our system. On one hand, team members could focus on developing the two controllers with less conflicts since the Model, which works as a communication interface, is already well defined. On the other hand, owing to the reduced size of each piece in the system design and the functionally grouped modules, it would be easier to enhance or debug the system in the future. Furthermore, the model has centralized the data structure, the rules and the logic for the application. View works on information representation like bar chart, table and diagram where the dynamic data are updated from the model. The controller handles the user ‘s input and translate it as commands both for model or view. The user manipulates the model through controller by input the commands and get the result from view.

In order to speed up the development, VueJS, an Model-View-ViewModel (MVVM) framework, is utilised for View generation. Since VueJS will bind the data to the UI automatically, we can reduce the efforts for translating the data output to the user interface, aka View. The view model controls the logic that how the data to be managed from model and provide data to view with its own properties. The view model is design for the view, express the status of the view and the method of process logic of the view.

JavaScript ES6 is a coding standard of JavaScript aka ECMASCRIPT2015. It brings us with new features and syntax for modern code which is more readable for developers. ES6 allows us to do more with less program code.

Test driven development (TDD) is a technique of agile software development. It encourages developers to firstly create the test scripts, and test the software with the scripts during the development. It could be utilized for automated test script and make the new passed codes would not breaking other parts. Otherwise, the codes must be re-written subject to the requirements until the tests are all passed.

Scalable Vector Graphics (SVG) is an XML based format that support two dimensional graphics for interactivity and animation on web environment. It is simple to use text files to describe curves, lines, text and colors. We adopted this technique as the output (SVG image) of our UML tool.

Regular expression is an object or special text string to represent a pattern of characters. We used some RegExp in our JavaScript for searching patterns. It helps extract information through any text by searching the matched patterns.

We designed an operation flow as the work process into the project. It is a sequence of certain process that includes the input of string, the string will be transferred for parsing and define objects after analyzing the rules or data structure. Finally it draws the SVG image as output for view.

# SOLUTION

T

he project uses Agile Development model and is divided into 8 weekly scrum sprints. A work breakdown structure (WBS) is drafted to identify tasks to be assigned to the sprints. The project also adopts diverse tools to facilitate the development cycle, bug findings, communications and testing.

The following table shows the usage of SE tools adopted:

|  |  |  |
| --- | --- | --- |
| **SE Tool** | **Technique** | **Usage** |
| Scrum | Agile Development Technique | Trace project progress |
| Pair Programming | Agile Development Technique | Come up with better solutions by discussion  Facilitate skill transfer |
| Slack | Communication Tool | Explain code throw online discussion  Facilitate skill transfer |
| VueJS | Model-View-ViewModel (MVVM) | Quick development with best practices. |
| Jasmine | Test-Driven Development (TDD) | Automated test script for unit testing |
| Git | Distributed Version Control | Keep tracking source update. Distributed backup. |

(Table 1)

## Work Breakdown Structure

### Text parser Design

1. Design syntax
2. [#TT-01] Design object type identifier
3. [#TT-02] Design attribute identifier
4. [#TT-03] Design method identifier
5. [#TT-04] Type identifier
6. [#TT-05] Design association identifier
7. [#TT-06] Create test cases
8. [#TT-07] Implement Unit test scripts
9. [#TT-08] Design tokenizing flow
10. Implement tokenizer
11. [#TT-09] Implement object type identifier
12. [#TT-10] Implement attribute identifier
13. [#TT-11] Implement method identifier
14. [#TT-12] Implement association identifier

### Drawing diagram

1. [#TD-01] Design test cases
2. [#TD-02] Design box-location assignment algorithm
3. [#TD-03] Feasibility Study on using VueJS for SVG diagram generation
4. [#TD-04] Implement box drawing
5. [#TD-05] Implement box-size estimation
6. [#TD-06] Implement box-location assignment algorithm
7. [#TD-07] Implement association link drawing

### User Interface

1. [#TI-01] Design HTML layout
2. [#TI-02] Implement HTML layout
3. [#TI-03] Implement image export

### System integration

1. [#TI-04] Integrate the UI to the system
2. [#TI-05] Integrate the text-analysis and the drawing modules

### Deployment

1. [#TY-01] Setup server VM
2. [#TY-02] Setup Apache server
3. [#TY-03] Deploy application

### Road-map

1. [#RM-01] Print function by click function
2. [#RM-02] Email image by click function

## Workload Estimation

|  |  |  |  |
| --- | --- | --- | --- |
| **Task ID** | **Estimated man-hour** | **Task ID** | **Estimated man-hour** |
| #TT-01 | 1 | #TD-01 | 16 |
| #TT-02 | 1 | #TD-02 | 42 |
| #TT-03 | 1 | #TD-03 | 24 |
| #TT-04 | 1 | #TD-04 | 16 |
| #TT-05 | 1 | #TD-05 | 8 |
| #TT-06 | 16 | #TD-06 | 40 |
| #TT-07 | 32 | #TD-07 | 44 |
| #TT-08 | 32 | #TI-01 | 8 |
| #TT-09 | 32 | #TI-02 | 16 |
| #TT-10 | 8 | #TI-03 | 8 |
| #TT-11 | 8 | #TI-04 | 4 |
| #TT-12 | 16 | #TI-05 | 16 |
| #TY-01 | 8 | #RM-01 | 8 |
| #TY-02 | 2 | #RM-02 | 8 |
| #TY-03 | 2 |  |  |

(Table 2)

## Scrum

It is estimated that 419 hours is required for all tasks. Each sprint should finish around 53 man-hours of tasks.

|  |  |  |  |
| --- | --- | --- | --- |
| **Sprint** | **New Tasks** | **Progress** | **Done** |
| 0 | #TT-01 ~ 05  #TD-01 ~ 02 |  |  |
| 1 | #TT-08  #TD-03 | #TD-01 (-4)  #TD-02 (-6) | #TT-01 ~ 05 |
| 2 | #TT-06 ~ 07  #TD-04 | #TT-08 (-12) | #TD-01 ~ 03 |
| 3 | #TD-05 ~ 06  #TI-01 ~ 02 | #TT-07 (-16)  #TT-08 (-4)  #TD-04 (-4) | #TT-06 |
| 4 | #TD-07  #TY-01 | #TT-07 (-4)  #TD-06 (-24)  #TI-02 (-6) | #TT-08  #TD-04 ~ 05 |
| 5 | #TT-09  #TY-02  #TI-03 | #TD-06 (-4)  #TD-07 (-32)  #TI-02 (-14) | #TT-07  #TY-01 |
| 6 | #TT-10 ~ 12 | #TT-09 (-4)  #TD-07 (-16)  #TI-02 (-6) | #TD-06  #TY-02  #TI-02 |
| 7 | #TI-04 ~ 05  #RM-01 ~ 02 | #TT-12 (-8) | #TT-09-11  #TD-07  #TI-03 |
| 8 |  |  | #TT-12  #TY-02 ~ 03  #TI-04 ~ 05  #RM01 ~ 02 |

(Table 3)

Refer to attachment [1] for detail.

## System Design

### Class diagram

Refer to attachment [2] which shows part of the class diagram.

### Text Identifiers and validations

There are two types of validations for input text lines: header line validations for class and interface definitions, and detail line validations for attribute and operation definitions.

We use “>>” to indicate inheritance and “||” to indicate implementation. Other notations, such as visibility, follow that in UML Class Diagram.

ValidateAssociationLine is used to check if any violation of the string format on the header line. The following validation rules are defined for an object definition.

For all objects:

1. The object name must not contain any special characters except \_ and $
2. If there is an association relationship defined (i.e. “>>” or “||”) for this object, the referenced object must be defined in the input text prior to this object’s definition
3. Duplicate object name is not allowed in the input text

For class objects:

1. Either “>>” or “||” can appear on the line string
2. At most one “>>” is allowed on the line string
3. Multiple “||”s are allowed on the line string
4. “>>” or “||” must be separated by a name in between
5. ">>" must go first (if any), followed by 0 to n “||”s

For interface objects:

1. Only “>>” is allowed on the line string
2. Must be separated by a name in between “>>” or “||”

ValidateLine is used to check if any violation of the string format occurs on the detail line of an object. The following validation rules are defined for an attribute or an operation.

For all objects:

1. A modifier must exist and be the first character of the line string
2. A separator (i.e. a colon (“:”)) must exist
3. All names, i.e. method name, attribute name parameter names (if any), and the method return type must not contain any special characters except \_ and $

For class objects:

1. A modifier must be + - # ~

For interface objects:

1. A modifier must be +

### Graph coordination algorithm

Some researches have been conducted to figure out a better implementation for calculating the coordinates of the nodes. We found some of the methods, such as spectral layouting method by Y. Koren; which are, however, not as simple to be understood and implemented within the project time. We decided to use a tree structure to layout the graph and migrate the coordinating algorithm in the future.

However, why tree structure can be an alternative for the nodes layouting method? Class diagram is in fact a directed acyclic graph (DAG) if we only consider inheritance and implementing interfaces. As long as the inheritance can never be cyclic, we can find out the root ancestor, which is also the root of a tree structure. As multiple roots can be found, we introduced an invisible root that will not be displayed, such that the calculation can be done in a single recursive function.

### Graph association lines coordination algorithm

The association lines are the edges connecting nodes. The lines should be headed on the edge of targeted class box, while the tail should also touch the edge of the source class box. The problem is: which edges are the nearest neighbour.

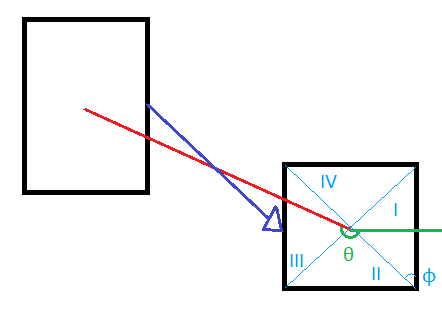
Our initial solution was using the angle θ to identify which the region is the line in. The range of angle for each of the region varies with the width-height ratio of the box.

Which are:

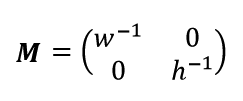
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| θ Range | Region I | Region II | Region III | Region IV |
| From | 3π/2 + φ | π/2 – φ | π/2 + φ | 3π/2 - φ |
| To | π/2 - φ | π/2 + φ | 3π/2 - φ | 3π/2 + φ |

Where φ is the angle of the diagonal at Region I = arctan( h/w ).

The following image shows the setting.



Checking of angles in the table is quite exhausting. Thus, we twisted the calculation a little bit. The final solution considers the source box as a perfect square by considering the width and height of the source box as unit vectors. The transformation is as follow in matrix form:

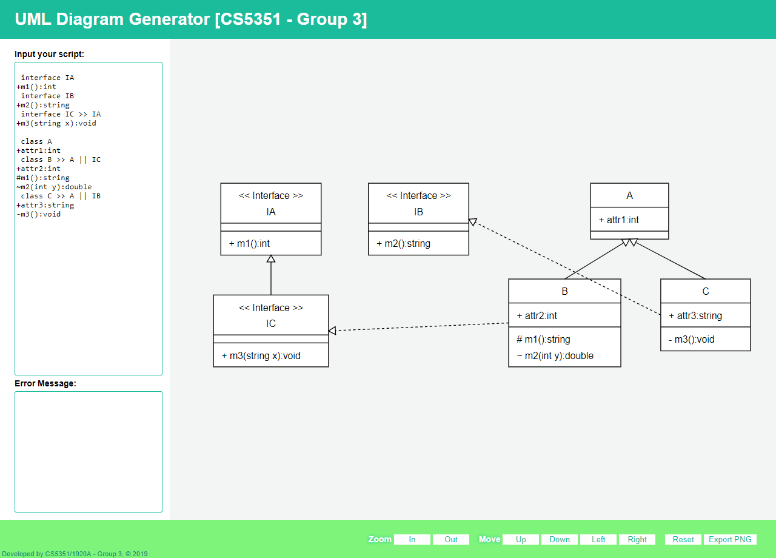


Where ***M*** is the transformation matrix, w is the width, and h is the height. After the transformation, the angle θ can be checked in a fixed interval of π/2. By rounding 2θ/π mod 4, the range would be 0, 1, 2, and 3, which represents the 4 edges correspondingly.

### Roadmap Functions

Print function by click - when user want to print the output to printer who just click the button, then can print it out.

Email image by click - when user want to send an email the output to outside who just click the button and type valid email address, then image can send it out to designated email address.

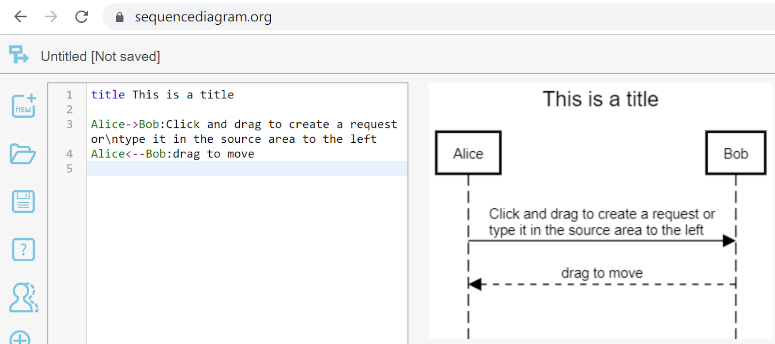
(Attachment [3])

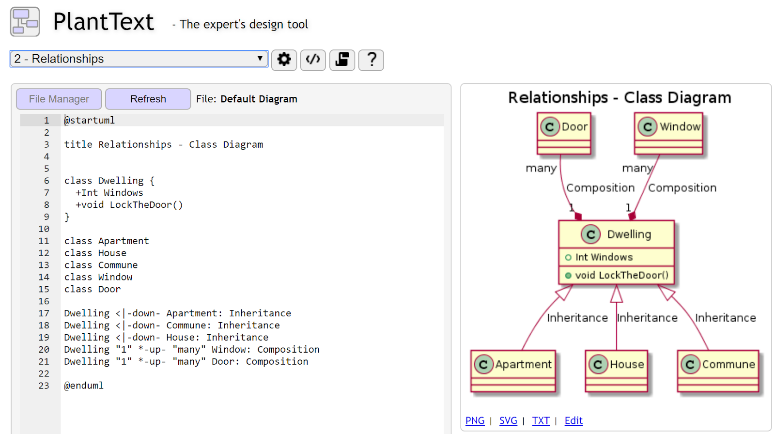
# Evaluation

T

here are similar web based tools aim at generating sequence diagram, flowchart directly from text input. Such tools are useful to generate diagrams on the fly and suitable for creating simple diagrams without installing complicated modelling software. Some existing tools are listed below:

1. Sequence diagram editor (<https://sequencediagram.org/>)

(Attachment [4])

1. PlantText (<https://www.planttext.com/>)

(Attachment [5])

The reason why we choose to create this project is the lack of simple text to UML class diagram modelling tool on the internet. For this project, the syntax of the text input is simple enough to remember and provides faster performance compare with drag-and-drop. The tool provides more functions, print service, mail the diagram and result export, which does not be provided by existing similar tool.

The tool is written in JavaScript which can run in browser directly, without installation, purely provides offline running. It means the tool is independent of OS type, browser type, Internet connection. For offline running, open “index.html” file via browser locally, the HTML file has preconfigured to call JavaScript file.  For online running, only upload the HTML & JavaScript files to root document directory of webserver, browser the index.html file via http(s) protocol, complex web-server configuration is not required. Developer can only focus on program code review and enhancement.

JavaScript can be read by browser directly, code compiling is not required. With this nature, code development process, debugging and further code maintenance can be carried out solely in the browser. JavaScript is not like another programming language, for example Java, needs to run within container (Tomcat/Weblogic). Developer can focus on code design only, no need to manager extra application server. If in Production Environment, using JavaScript is better for CI/CD processes as the code can be deployed immediately to Production Environment after testing locally and is without service interruption.

# Conclusion

W

e have introduced a development tool for developers who are going to initial build a new project in an efficient way. This tool not only minimize the cost of creating the diagram on design phase, but also can easily create UML class diagrams which are useful when modelling business data. It can accurately model attributes and associations of class entities, developers or users can map these class diagram specifications to entity easily. The process of the business modelling could be developed faster as well as the next generation. Moreover, the developer can flexibly handle the user feedbacks and the requirements, our diagram can be updated instantly and output the new result. By abstract concept, Class attributes map to access method for persistent/static fields, and association roles map to access methods for relationship fields. A wrong type of attributes or character could be identified when user inputs. The validation can help to correct the data as early as possible. Without the tool, an error may be carried to next phase and incur more complicated problem.

Basically, we have developed fundamental tools for developer to create UML class diagram. For future work, some models/ functions can be enriched to make it more fruitful. Firstly, we can extend to text parser to understand multiplicity notation in the input text. The text parser can determine the correct type for relationship fields and delete cascading characteristics. Secondly, there is no “print and forwarding” function for the UML Class diagram result. We will create Print and Forwarding function (e.g. Email / Whatsapp / Facebook) to redirect the result to other parties. Likely output the result as a documentation both for client review and store as part of our backlog. Lastly, the result of the class member cannot be editable, we can modify it which can be selectable and editable managed within a class shape.

# References

[1] Tutorialspoint.com. (2019). MVC Framework - Introduction - Tutorialspoint. [online] Available at: https://www.tutorialspoint.com/mvc\_framework/mvc\_framework\_introduction.htm [Accessed 3 Sep. 2019].

[2] En.wikipedia.org. (2019). Model–view–controller. [online] Available at: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller [Accessed 3 Sep. 2019]

[3] Visual-paradigm.com. (2019). What is Unified Modeling Language (UML)?. [online] Available at: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/ [Accessed 3 Sep. 2019].

[4] Tutorialspoint.com. (2019). UML - Class Diagram - Tutorialspoint. [online] Available at: https://www.tutorialspoint.com/uml/uml\_class\_diagram.htm [Accessed 3 Sep. 2019].

[5] Deepak Kumar (2019), ES6 | A Comprehensive Guide to Learn ES2015(ES6). [online] Available at: <https://dev.to/dipakkr/es6-a-comprehensive-guide-to-learn-es2015-es6-2ao1> [Accessed 3 Sep. 2019].

[6] Said (2018). JavaScript ES6 — write less, do more. [online] Available at: https://www.freecodecamp.org/news/write-less-do-more-with-javascript-es6-5fd4a8e50ee2/ [Accessed 3 Sep. 2019].

[7] Galinsky, A. (2016). JavaScript Unit Testing with TDD, Jasmine and Karma. [online] TestProject. Available at: https://blog.testproject.io/2016/08/14/javascript-unit-testing-with-tdd-jasmine-and-karma/ [Accessed 3 Sep. 2019].

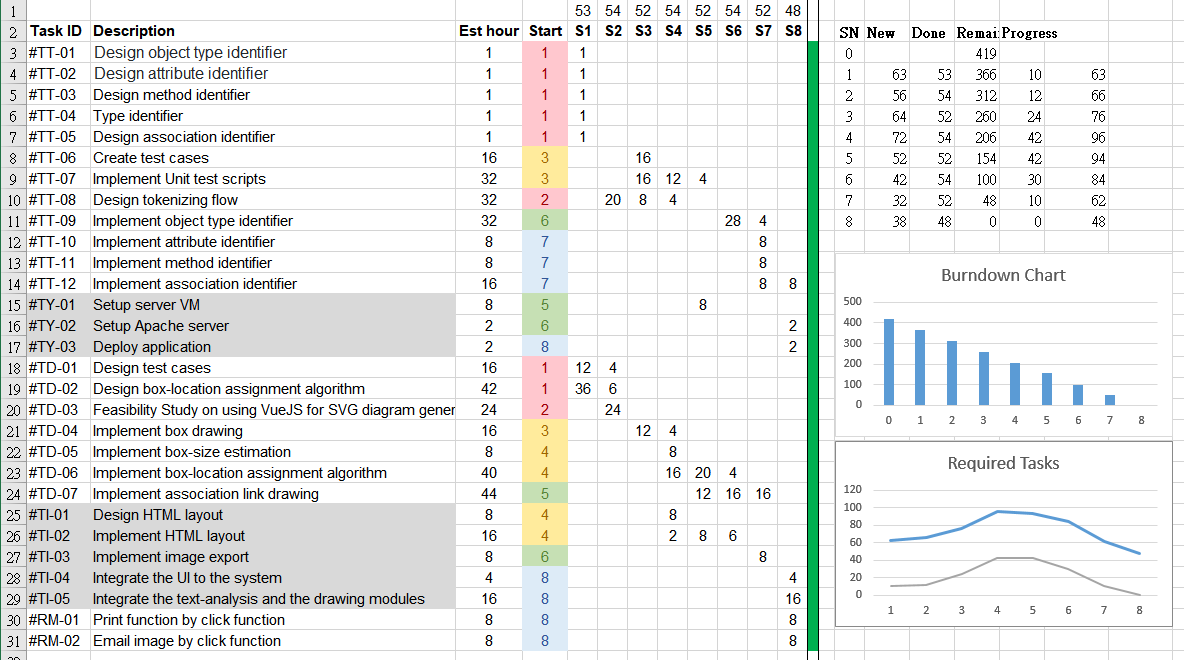
[8] En.wikipedia.org. (2019). Scalable Vector Graphics. [online] Available at: https://en.wikipedia.org/wiki/Scalable\_Vector\_Graphics [Accessed 3 Sep. 2019].

[9] Y. Koren, “Drawing graphs by eigenvectors: theory and practice,” Computers & Mathematics with Applications, vol. 49, no. 11-12, pp. 1867–1888, 2005.

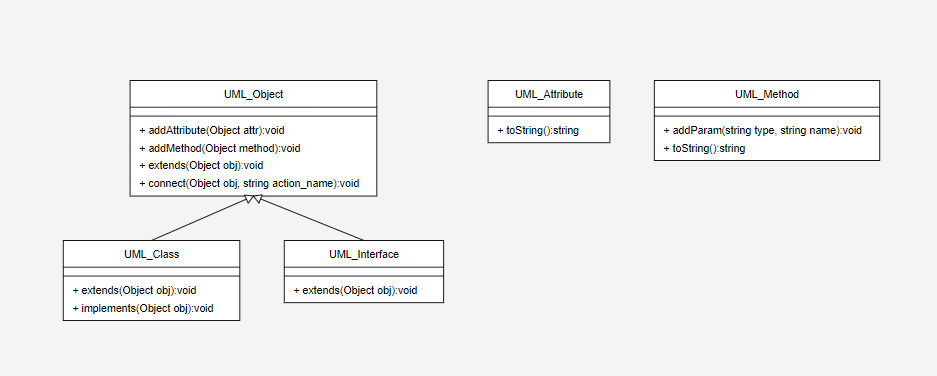
# Students Bio

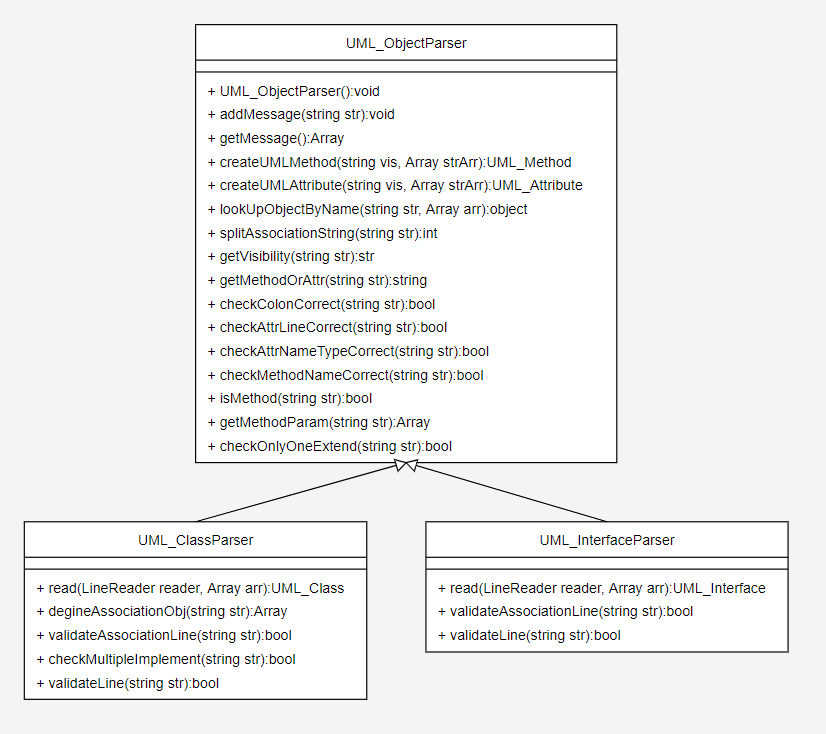
# Attachment

## Attachment [1]

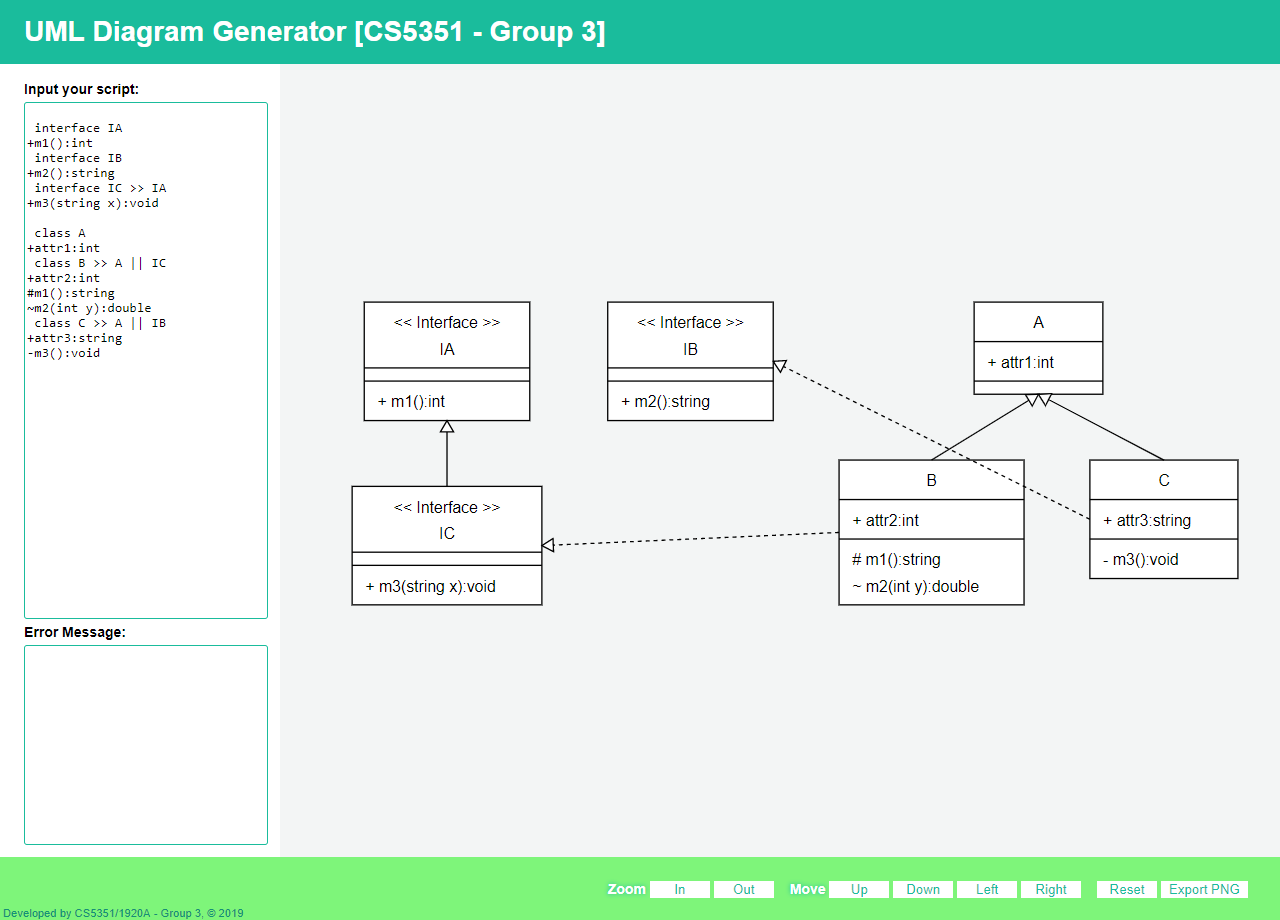


## Attachment [2]

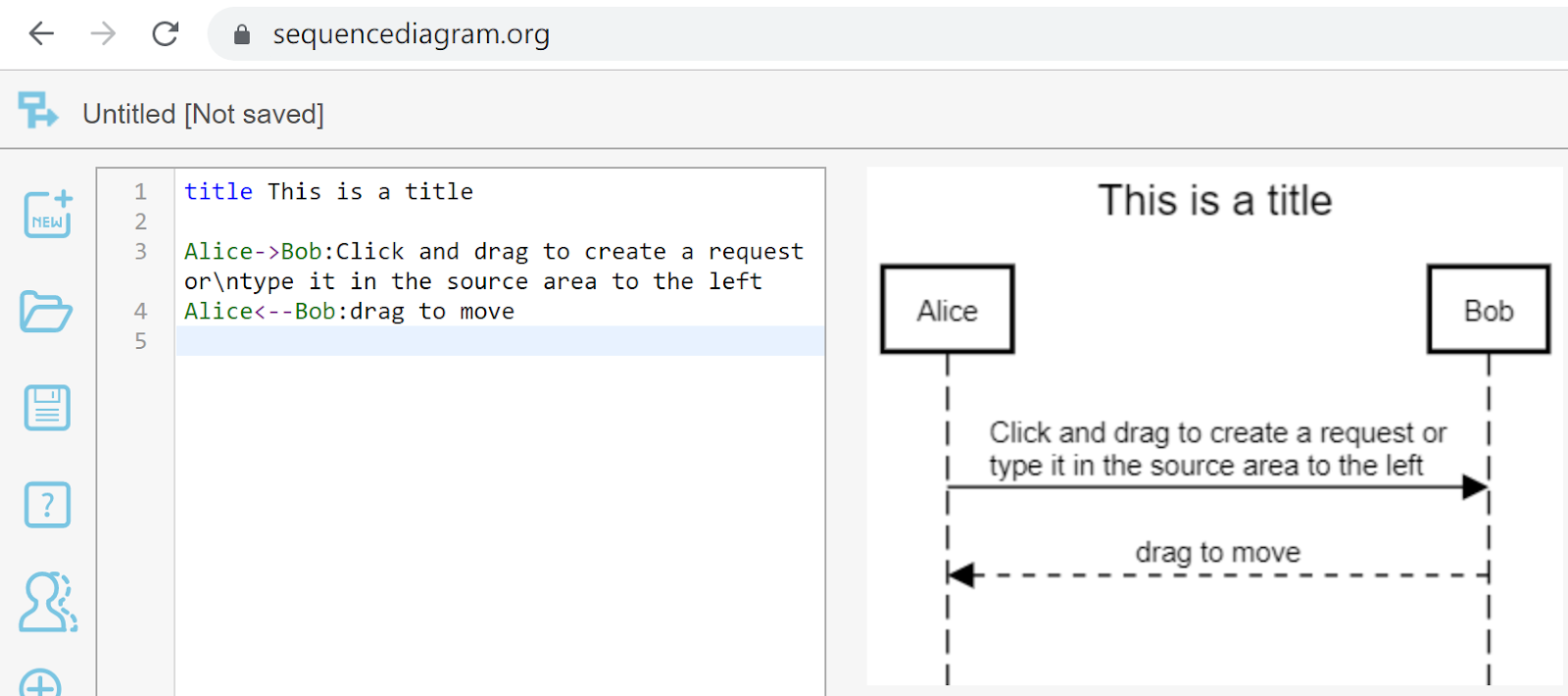




## Attachment [3]



## Attachment [4]



## Attachment [5]

