SOFT20091: Software Design & Implementation

Sean Turner - N0643456

Dan Hughes - N0644583

Melvin Joy - N0640825

Thomas Moore – N0642032

Table of Contents

[Overview 3](#_Toc481517381)

[Requirements Specification 3](#_Toc481517382)

[Table 3](#_Toc481517383)

[Use Case 5](#_Toc481517384)

[Activity/sequence diagram 6](#_Toc481517385)

[Class diagrams 7](#_Toc481517386)

[Software Development Life-Cycle 8](#_Toc481517387)

[Software Functionalities 8](#_Toc481517388)

[MoSCoW 8](#_Toc481517389)

[Analytical Techniques 9](#_Toc481517390)

[Swot 9](#_Toc481517391)

[Design Patterns 10](#_Toc481517392)

[MVC 10](#_Toc481517393)

[Data Structure 11](#_Toc481517394)

[Sort algorithm 11](#_Toc481517395)

[Implementation 12](#_Toc481517396)

[Testing 13](#_Toc481517397)

[Approach 13](#_Toc481517398)

[Performance 13](#_Toc481517399)

[Conclusion 14](#_Toc481517400)

[Appendices 15](#_Toc481517401)

[Use Cases 15](#_Toc481517402)

[Testing Table 18](#_Toc481517403)

[Results 19](#_Toc481517404)

[References 20](#_Toc481517405)

# Overview

This report is on the Pokémon Pokedex software, which is created to understand how the design and development of Pokedex Software took place. The Pokémon Pokedex is a software that will be able to store all the details of the Pokémons and when the user initially uses the Pokedex has no Pokémon details stored in the Pokedex, this is so that when they come across new Pokémon they can be added to big database of Pokémon’s with all their details. The Pokedex’s main functions will be user can register, search, update, delete, sort and view evolutionary data of all Pokémon.

# Requirements Specification

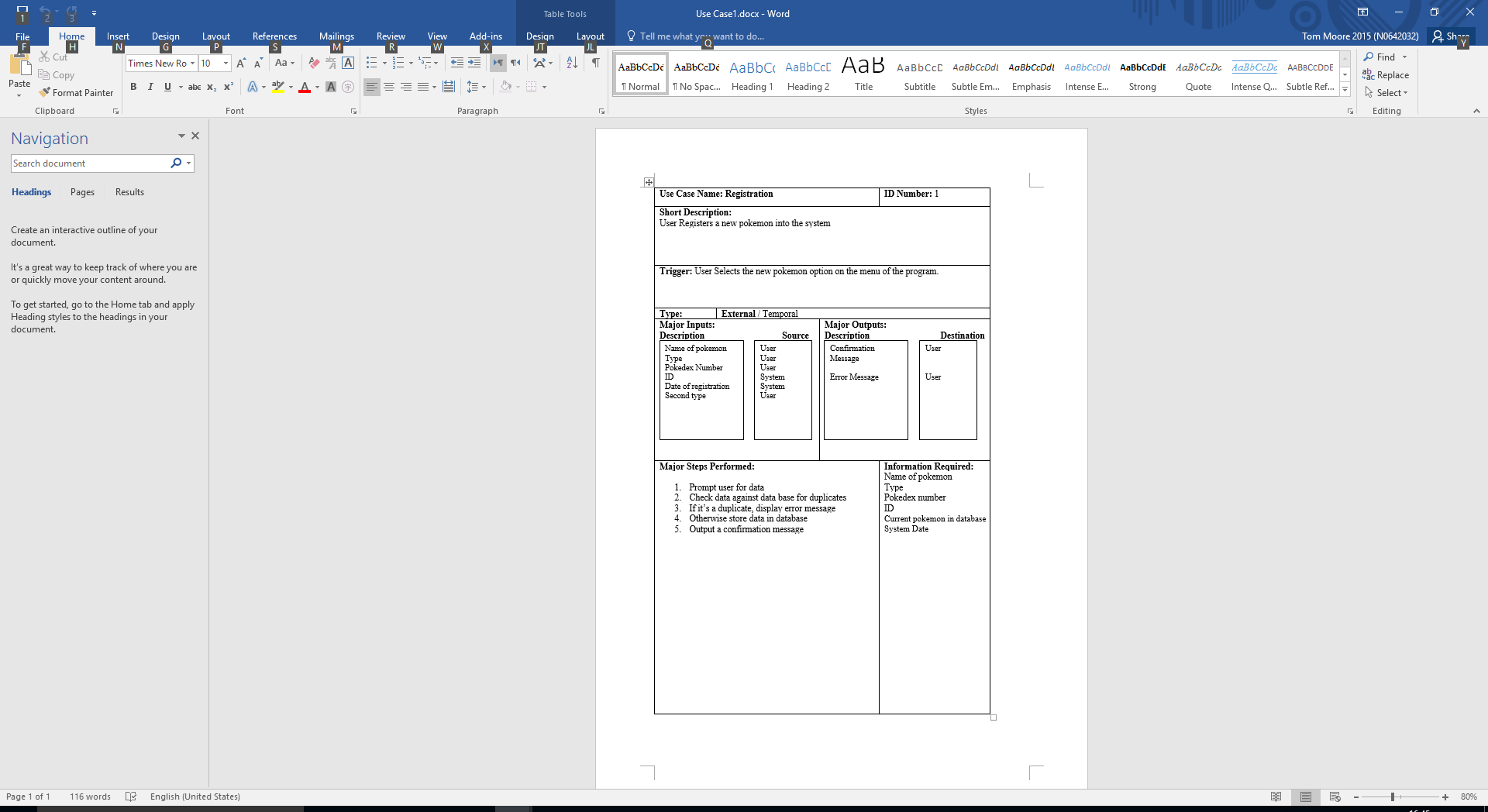
## Table

|  |  |  |  |
| --- | --- | --- | --- |
| Number | Type | Requirement Description | Implication |
| 1 | Functional | The ability to add a new Pokémon record to the storage. | User interface to enter Pokémon details |
| 2 | Functional | The ability to hold the data and information of Pokémon. | Use an array to store Pokémon data. Allow it to be dynamic. |
| 3 | Functional | The ability to search through stored data by National Dex Number, ID number or by Pokémon Types (Primary/Secondary). | Implement an appropriate searching algorithm based on a given field. |
| 4 | Functional | The ability to sort records by National Dex Number, ID Number or by Type. | Implement an appropriate sorting algorithm given a field to sort on. |
| 5 | Performance | Chosen sorting algorithm is most efficient for the selected data type(s). | Algorithm chosen must suit a dynamic array of classes. |
| 6 | Performance | Chosen data type(s) store the data in the most efficient way possible. | Limit the amount of unused space in data types by selecting the best data type. |
| 7 | Physical and Operational | A suitable user interface is chosen to allow for the user to easily navigate the Pokedex. | Create an interface that is easy and instinctive. |
| 8 | Functional | Allow Pokémon registered to have multiple typing. | Use of string within class and Boolean to show whether there is or is not a second type with string to show second if it is. |
| 9 | Functional | Allow for all Pokémon to be viewed at once, in order of ID Number. | Implement a sort and display the results in a 10 per page window |
| 10 | Functional | Allow the ability to view Pokémon that evolve and Pokémon that do not in separate lists. | Allow a search of Pokémon and only show ones with an evolution or prevolution |
| 11 | Functional | Allow the viewing of evolutions lines of Pokémon that do evolve. | Allow selection of Pokémon with evolutions and display that evolution tree |
| 12 | Operational | The date that a Pokémon is registered is recorded. | When a Pokémon is entered get system date and store it with the Pokémon |
| 13 | Operational | Keep records of Pokémon registered by the user, to be viewed whenever needed. | Allow entered Pokémon to be stored in data structure. |
| 14 | Operational | Allow Pokémon to be deleted from the system. | Allow the user the choice of deleting a Pokémon record. Then remove this record from our system. |
| 15 | Operational | Allow for the modification or updating of an existing record | Allow the user to make changes to each record then store this changes in the system |
| 16 | Functional | Stop duplication of previously entered Pokémon | If a Pokémon entered is already in the system don’t allow it to be entered twice. Ask if it needs to be edited |

## Use Case



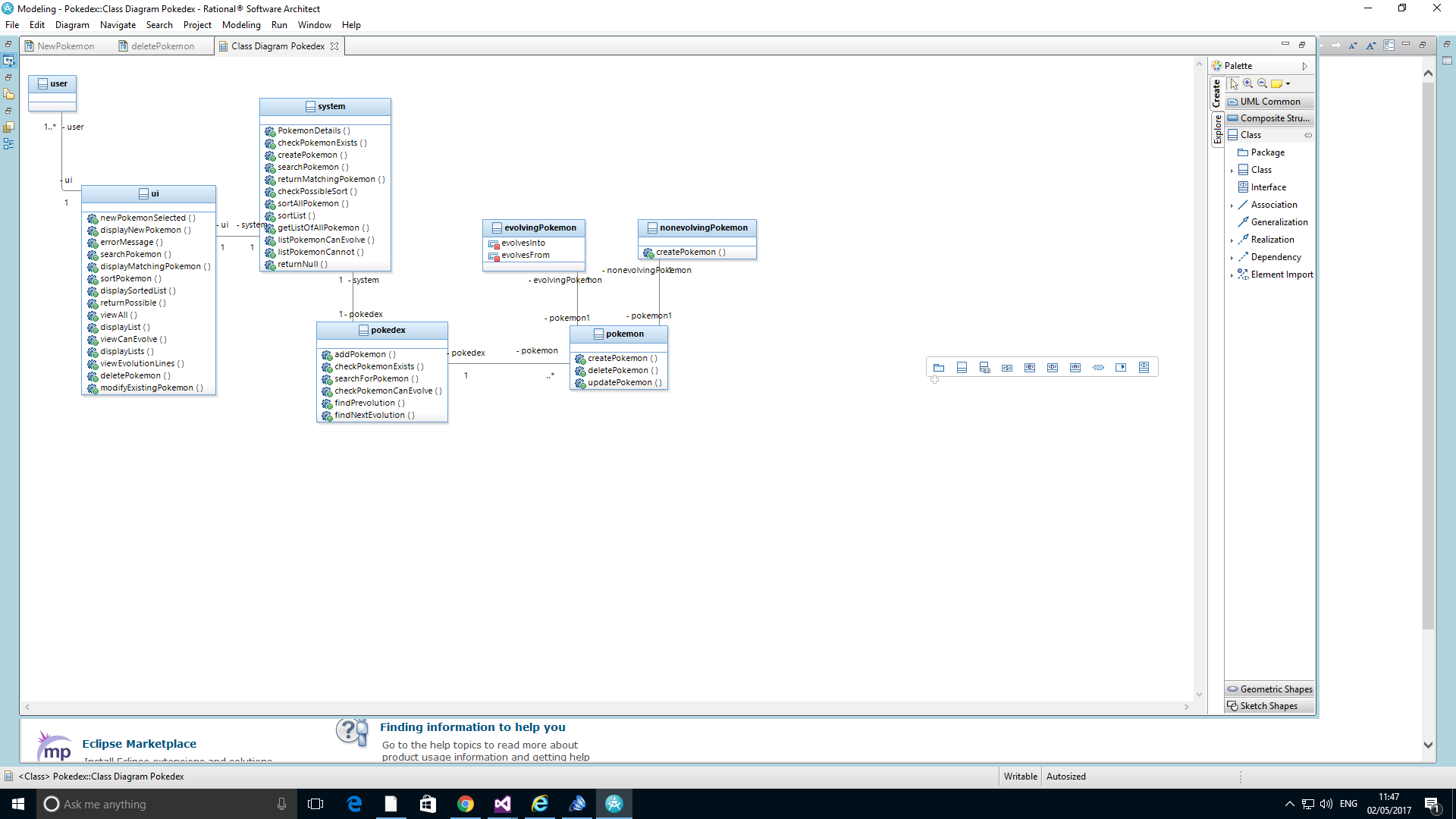
Figure 1



Other Use Case Tables are in Appendices

## Activity/sequence diagram

## Class diagrams



# Software Development Life-Cycle

The Software Development Life-Cycle that is used for the development of the project is the agile framework called Scrum. Scrum is used in development for cross functional or self - organising teams of about 4 - 9 people, and uses fixed – length iterations called sprints to produce a potential product every sprint through a series of organised meetings. Scrum has roles of development team, product owner and scrum master. The product owner mainly looks at the requirements for the project, development team builds a potential product every sprint and scrum master acts as the facilitator. Scrum contains artifacts of product backlog and sprint backlog. Product backlog is everything that could be done by priority and sprint backlog is what has been agreed to do for each sprint. (James, 2010)

In the group all of the members will be acting as the development team role, one of the group member will also be given the product owner role and the module tutor will be acting as the scrum master. The advantage of using scrum for the development of the Pokedex is that the group is only size of 4 people and people can easily self – organise through a series of meetings and as there is leader people can assign each other tasks through meetings where issues can be resolved quickly. The sprints and constant feedback means that it will be easy to cope with changes while having a working product throughout, which makes it easier to deliver quality product in the given time.

# Software Functionalities

## MoSCoW

|  |  |
| --- | --- |
| MUST | * Registration of new Pokémon’s. * Sorting of all the Pokémon’s in the system. * Be able search specific Pokémon’s with their national Pokedex number. * Modify details for existing Pokémon’s. * Delete Pokémon’s details. * Be able to see detailed evolutionary line of the Pokémon. |
| SHOULD | * Make sure that there aren’t any duplicate details of Pokémon. * Pokedex Software should be coded efficiently to give the best performance. * User friendly and easy to navigate. |
| COULD | * GUI interface so it makes it is more appealing and user friendly. * Images can be added of each Pokémon and displayed. * Export and import into file. * Each Pokémon data can be compared with each other. |
| WONT | * Graphs displayed for each details of the Pokémon. * Abilities of Pokémon can be displayed. * User registration and be able to share Pokémon details. |

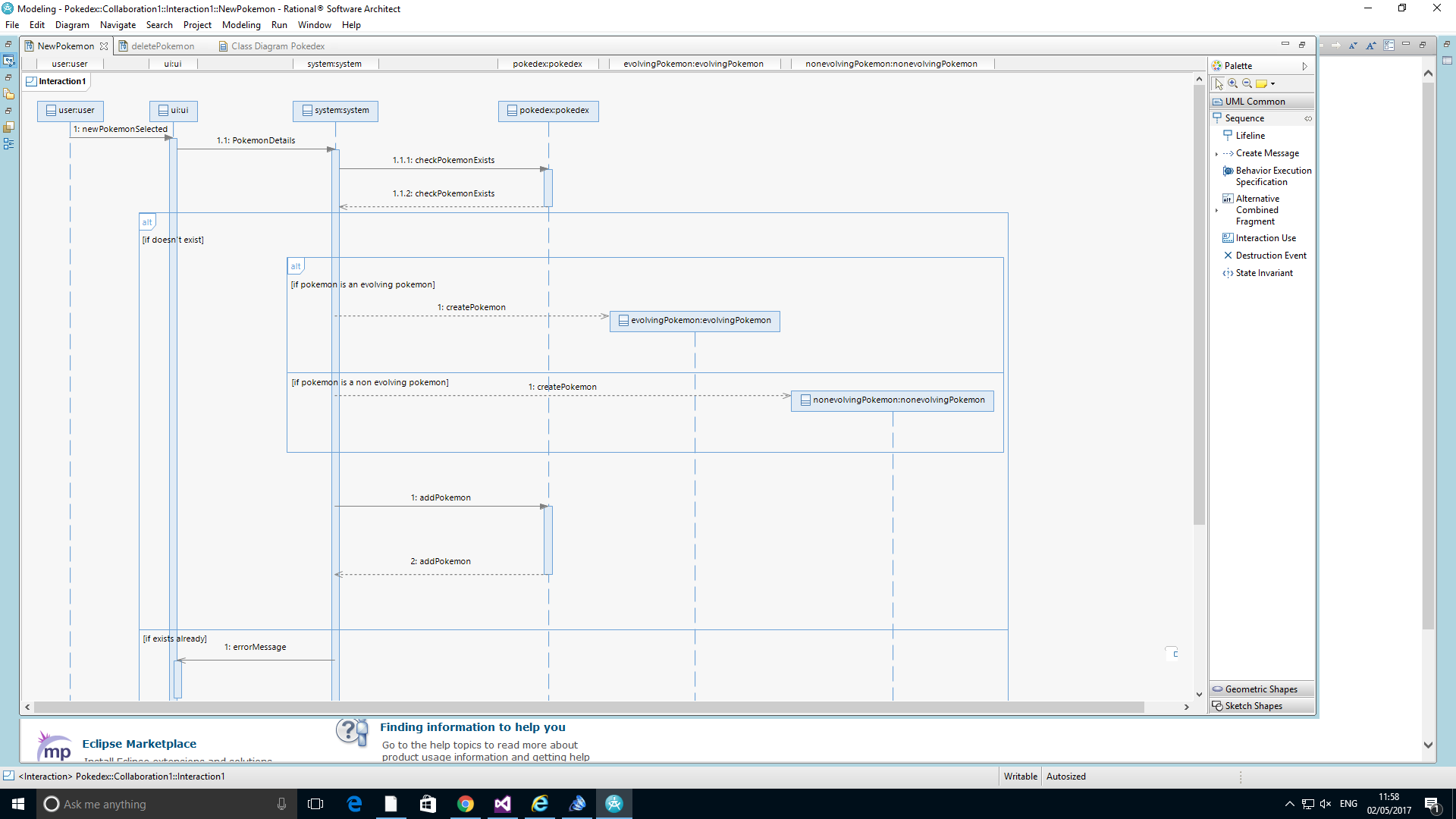
# Analytical Techniques

## Swot

|  |  |
| --- | --- |
| Strength | Weaknesses |
| * Flexible because the user can manipulate the data as they require. * User friendly by easily being able to navigate through the pokedex. * The software performs efficiently by making sure that code uses efficient algorithm and design pattern. | * When user adds Pokémon there is no check system to see if what user added is actually a Pokémon. * Compared to current Pokedex systems in the market, the Pokedex system that is being implemented has no statistical details of the Pokémon’s in the forms of graphs etc. |
| Opportunities | Threats |
| * GUI user interfaces used to enable easier navigation for the user and better appeal to the software. * The details entered in the Pokedex could be compared with the actual big database of Pokemon online to see if entered details are correct. * Be able to register multiple users to the system and enable them to share details of Pokémon’s. | * Storage of data might have a limit in space if a large about of details are added. * Need to make sure that there are no memory leaks in the program. |

# Design Patterns

## MVC



View

Control

Model

For the Pokedex system the design pattern that will be used is the Model View Controller(MVC), where the User Interface class is the view, system class is the control and Pokedex is the model. The user selects new Pokémon to be inserted and the UI sends the details to the system and the system then checks if the Pokémon exists. If the Pokémon exists the error is sent to the user from the system to the UI otherwise the Pokémon is added to the Pokedex. Using the Model View Controller makes the design more flexible and extensible and it will make it easier when Pokedex is developed further in the future.

# Data Structure

A vector data type is a type of dynamic array. The data structure allocates a set number of spaces In memory which can hold data for the user. As a vector is dynamic it does not need a specified size. If the underlying array is full and another piece of data is added, the vector is copied into a larger underlying array, with the new data added. As vectors are dynamic when being declared the programmer can create a vector of a certain size or an empty vector with 0 elements, both can have elements added and removed. This allows the program to store an unknown number of data elements. This is more efficient as less memory is needed, for instance to store an unknown amount of data with an array would require an extremely large array to store a possibly large amount of data. Each element of the array requires an allocated piece of memory even if blank. Therefore if only a small number of elements where required it would be a large waste of memory. This is avoided by vectors as a vector only requires enough allocated pieces of memory to store all the values, or the number of values declared by the programmer. Removing the need for an extremely large number of allocations which may not be used.

In our program, we utilise vectors in a couple of ways. Firstly, we use vectors to store our Pokemon class, so that users can add and delete Pokemon to the Pokedex. Two vectors are used in this manner- One to store all Pokemon, and one to store only Pokemon that evolve, as the class for the Evolving Pokemon provide additional functionality, which can be utilised in the system’s various functions. Another way that vectors are utilised in our system is through the sorting algorithm, Merge sort. This sort breaks down a Vector into a series of small vectors in order to make sorting them easier. As a result, a number of small vectors are created that can be accessed by the system and rearranged with ease.

We decided to implement the vector data Structure for its efficiency benefits, the system allows users to Add and delete pokemon, as we do not know the number of pokemon the user will be entering into the system a vector makes sense as it avoid wasted memory allocation, reducing the overall resources needed by the program. This is desirable as it is important for a program to be as resource light and efficient as possible. The ability to iterate through a vector to search, and also the ability to delete a specific piece of allocated data at a specific point in a vector was also desirable when considering vectors as it would help make the ‘search’ and ‘delete’ functionalities easier to implement as the vector can be indexed and iterated through.

# Sort algorithm

The searching algorithm we have chosen to use within the system is a Merge sort. *Merge sort is a sorting technique based on divide and conquer technique.* (TutorialsPoint, year unknown). What this means is that, rather than tackling the challenge of sorting the data all at once, the data is broken down into smaller lists, and then these smaller lists are sorted. Doing this allows for large data stores to be sorted in a consistent and approachable manner.

We chose to use Merge Sort as we believe the complexity of the sort matches our system. As our system will consist of a large list of datatypes with functions (Pokemon), breaking these lists down and then accessing the functions when needed rather than tackling a large portion at once is a far more system friendly approach. Merge sort also provides a consistent result throughout, as its Best, Worst and Average case scenarios for Merge sort are all the same (O n log n). Because of this, our system will always perform the same, regardless of best, worst or average case.

In our system, our Pokedex class contains a merge Sort and merge function. These functions work in conjunction after being called. The Pokedex search menu calls the merge Sort function, passing it the vector it needs to sort and the way it should be sorted by (Name, ID, National Dex Number, Type). Following this, the merge Sort function will split the vector down into smaller vectors recursively, calling itself until the vectors are too small to split further. Then, the vectors are passed to the merge function. This function will merge together and sort these lists based on the chosen sort type, until there is only one list left. Then, the sorted list will be returned to be displayed.

# Implementation

Overall, we feel that the implementation approach we took to the problem we faced was good, though there are a number of improvements that we could make. The structure of breaking down our C++ files based on the system utilities they provide allowed for us to quickly find functions that may be causing issues within our code when it came to implementing and testing them, as we could narrow down the possibilities based on the area of system we were operating upon. We broke down our functionality into submenus that a user can work through, refining the amount of information a user faces and allowing for them to understand how to use the system fully. This approach worked well, as it allowed for us to create an environment that we feel that a majority of users could use with very little issue.

However, a small amount of functionality that we could have implemented, such as an import or export function or modifying the user’s inputted Pokedex files, had to be scrapped towards the end of the project. This was due to unforeseen difficulties implementing these functions into our code, which, as a result, lead to us having less time than necessary to complete this functionality. In the future, we feel it best to research potential implementations of additional functionality in advance, so we can foresee these issues early and move to tackle them accordingly.

We also feel that a more prototype-based approach may have suited our system better, due to issues implementing evolving and standard Pokemon into the same vector. This was an issue we could not have foreseen, and a number of functions within the system had to be redesigned in order to accompany for a solution, which ate up time and resources. Through a prototype system, a more efficient system could have been created which tackled these issues, allowing for us to progress on from the problem, rather than having to return to old functionality and update it.

# Testing

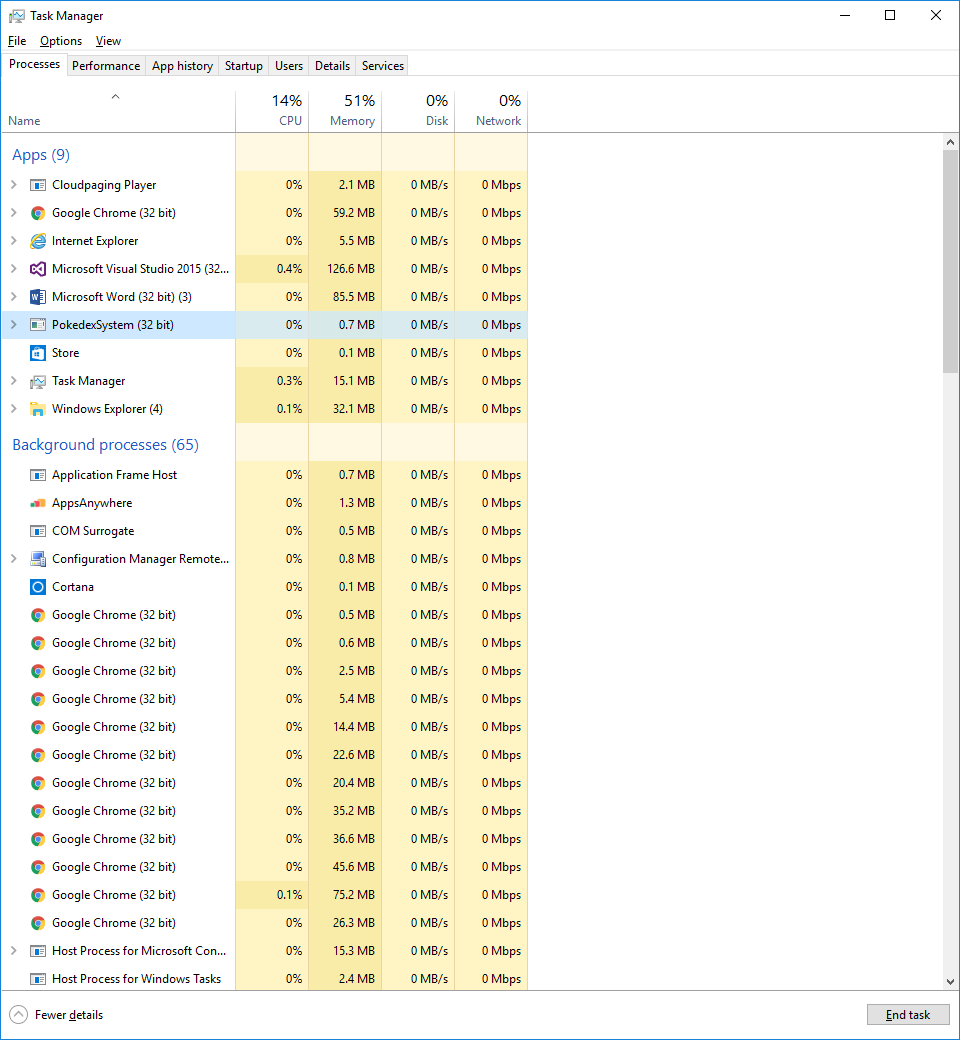
## Approach

Write shit about how we’re testing

For the project we employed black box testing as the testing methodology for the system. We chose black box as it allows us to ensure the experience of the system is as smooth for the user as we intend, as we are using the system as if we were users however we can compare the output with the expected output. On reflection our testing approach was effective as black box testing is efficient, allowing us to ensure the program works from the view of the user quickly and ensure all interaction with the system is as we intend. This allowed us to quickly and easily identify any bugs in our system.

## Performance

Upon reflection of the system we are happy with the systems response to input, the system responds in almost real-time to the user, suggesting a high computation efficiency; giving us high confidence in its performance as the system does not feel sluggish in response. Which would reduce the user experience. The system is also scalable, as we have used vectors the system is efficient with any number of pokemon added by the user. Using a merge sort also helps to keep scalability as this is an efficient and fast method of sorting, meaning computation time for large datasets would be kept low, keeping speed and responsiveness of the system. The system is very reliable, although the data the system outputs may change depending on the order entered into the system, if entered exactly the same, in the same order and exactly the same data, the system will produce the exact same outputs, showing a high level of repeatability as the same input will always garner the same response, giving us confidence in the systems overall reliability. The low resource use and high performance of the system is shown in the screen shot below which shows the resources used by the program:

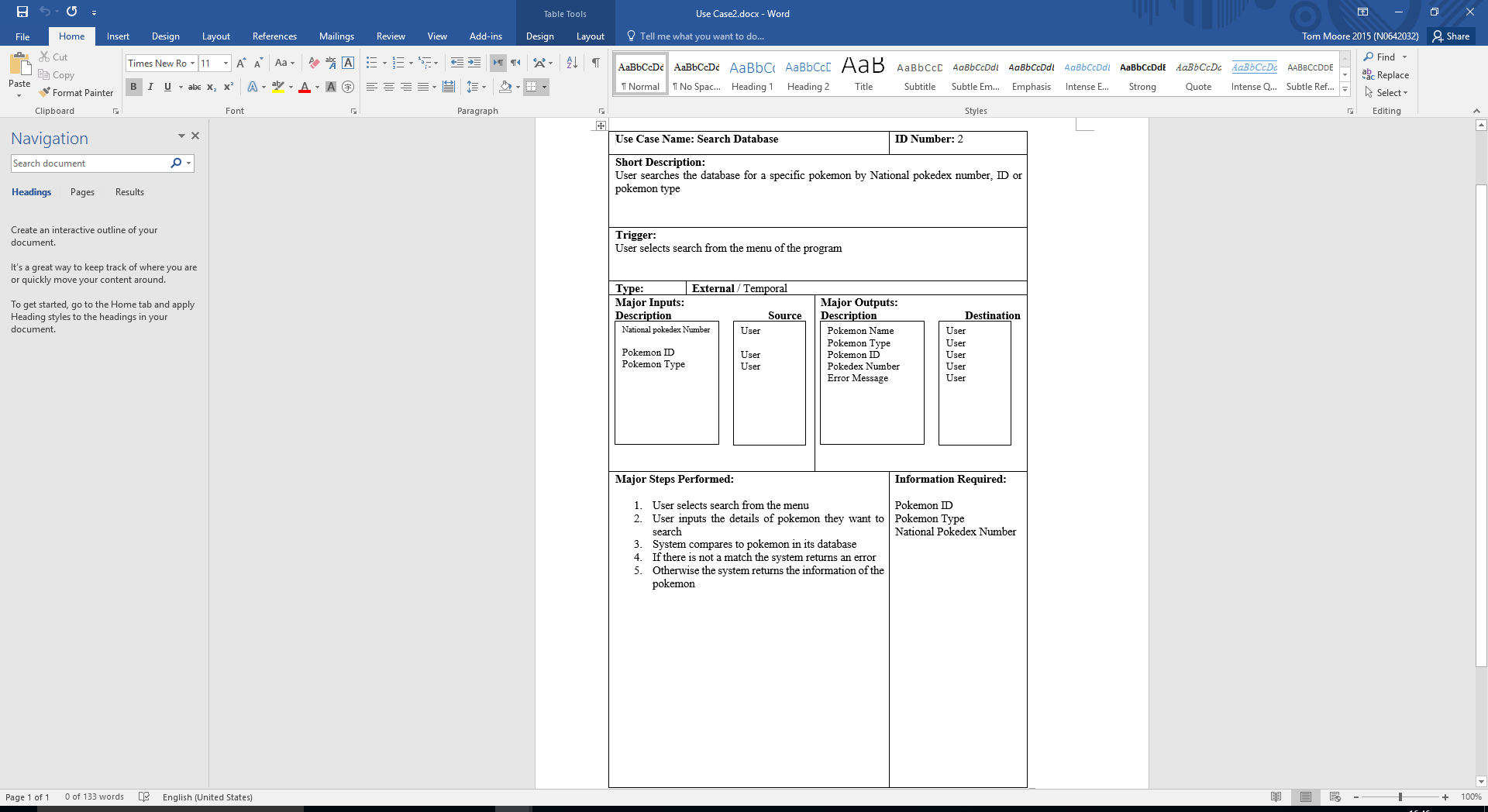


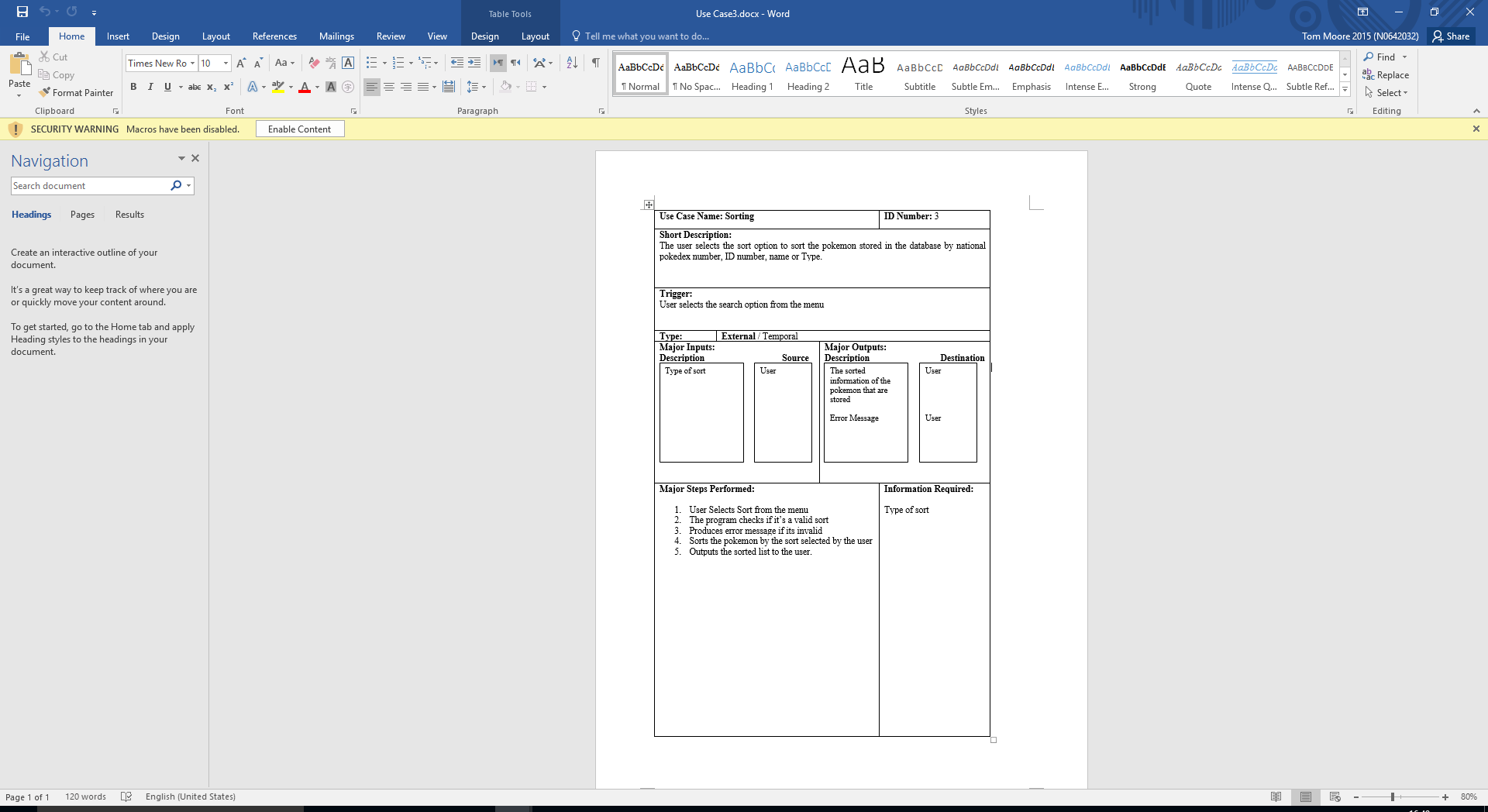
As you can see this is very low for a program on a modern machine.

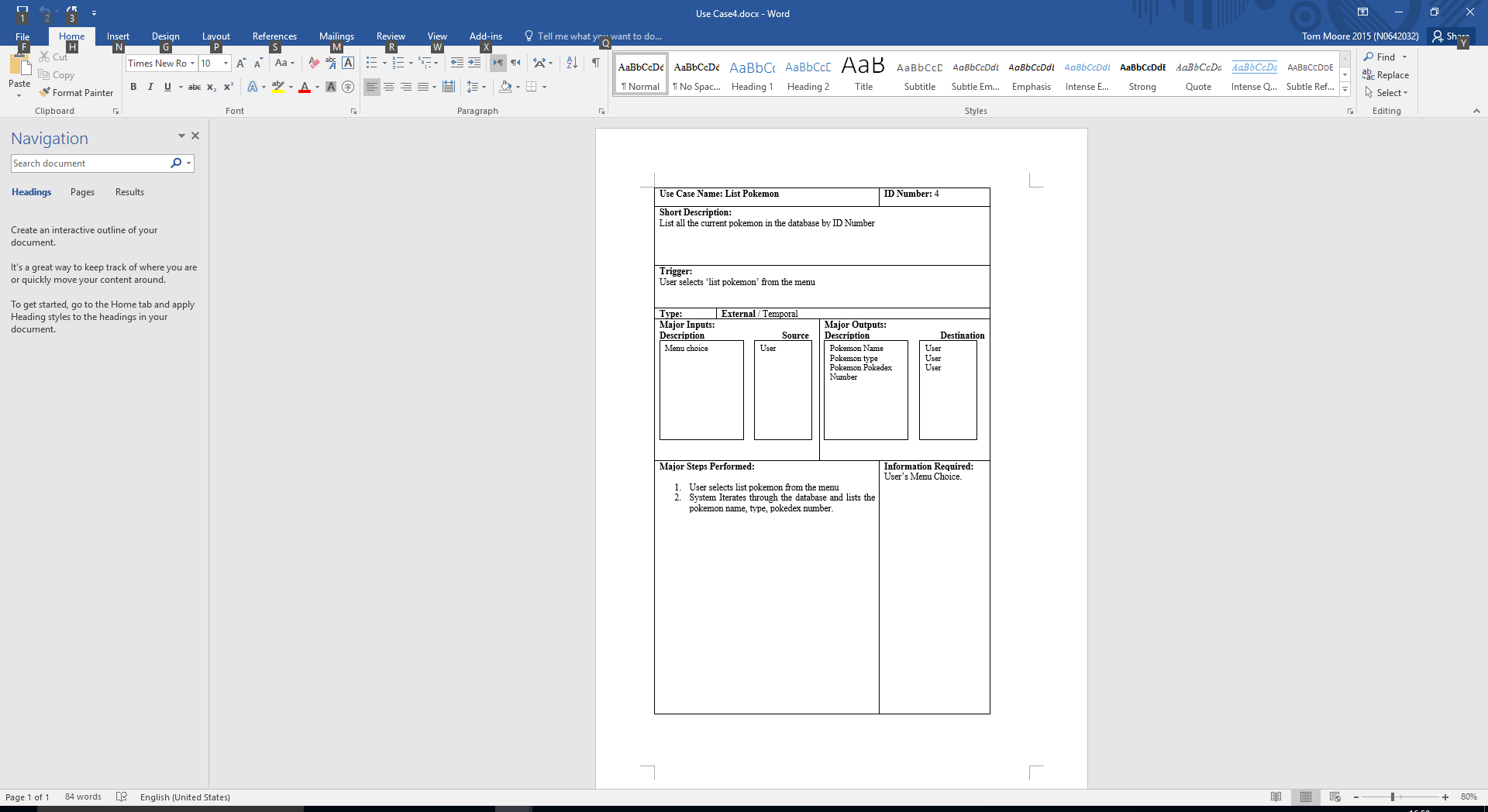
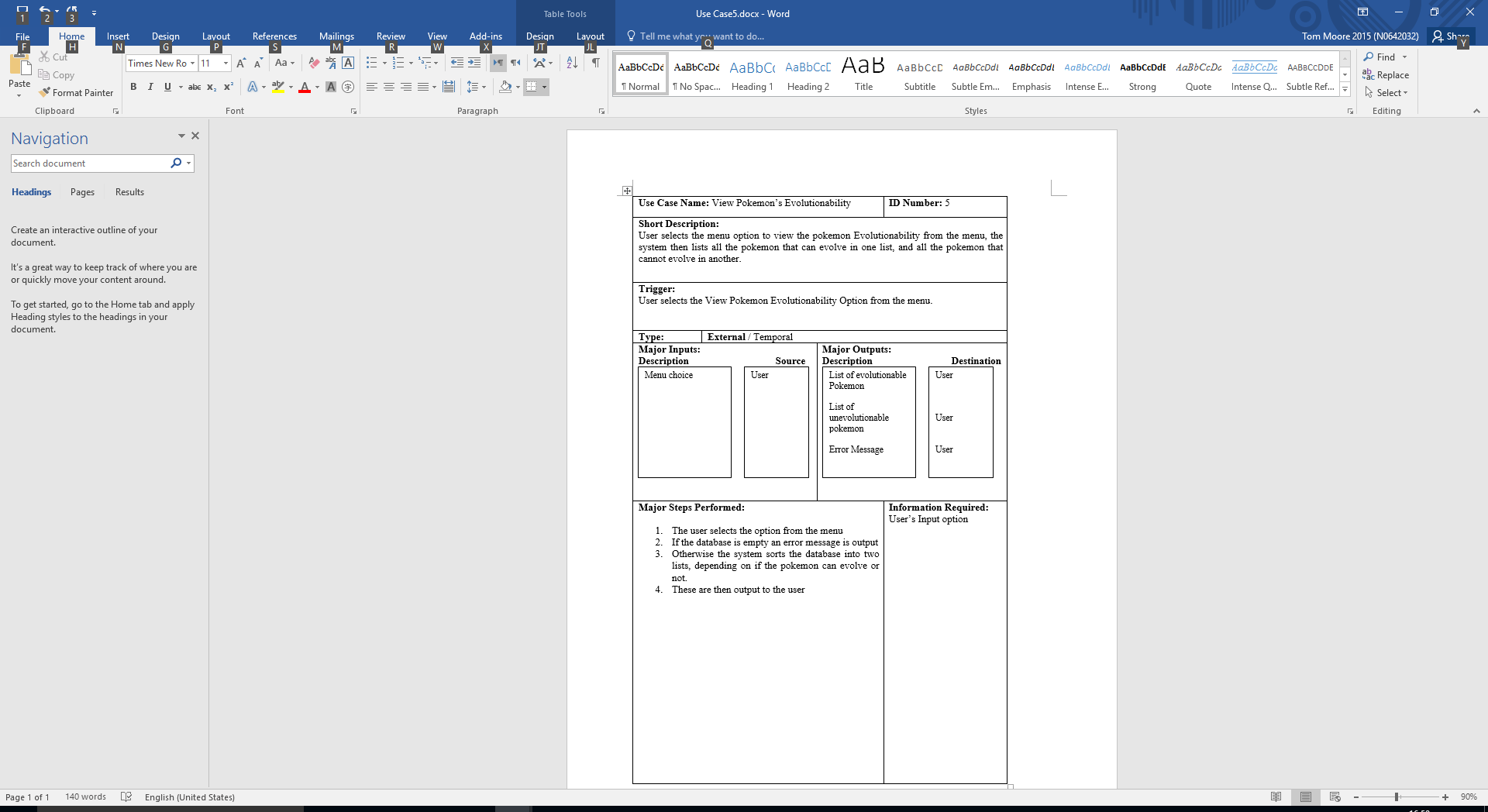
# Conclusion

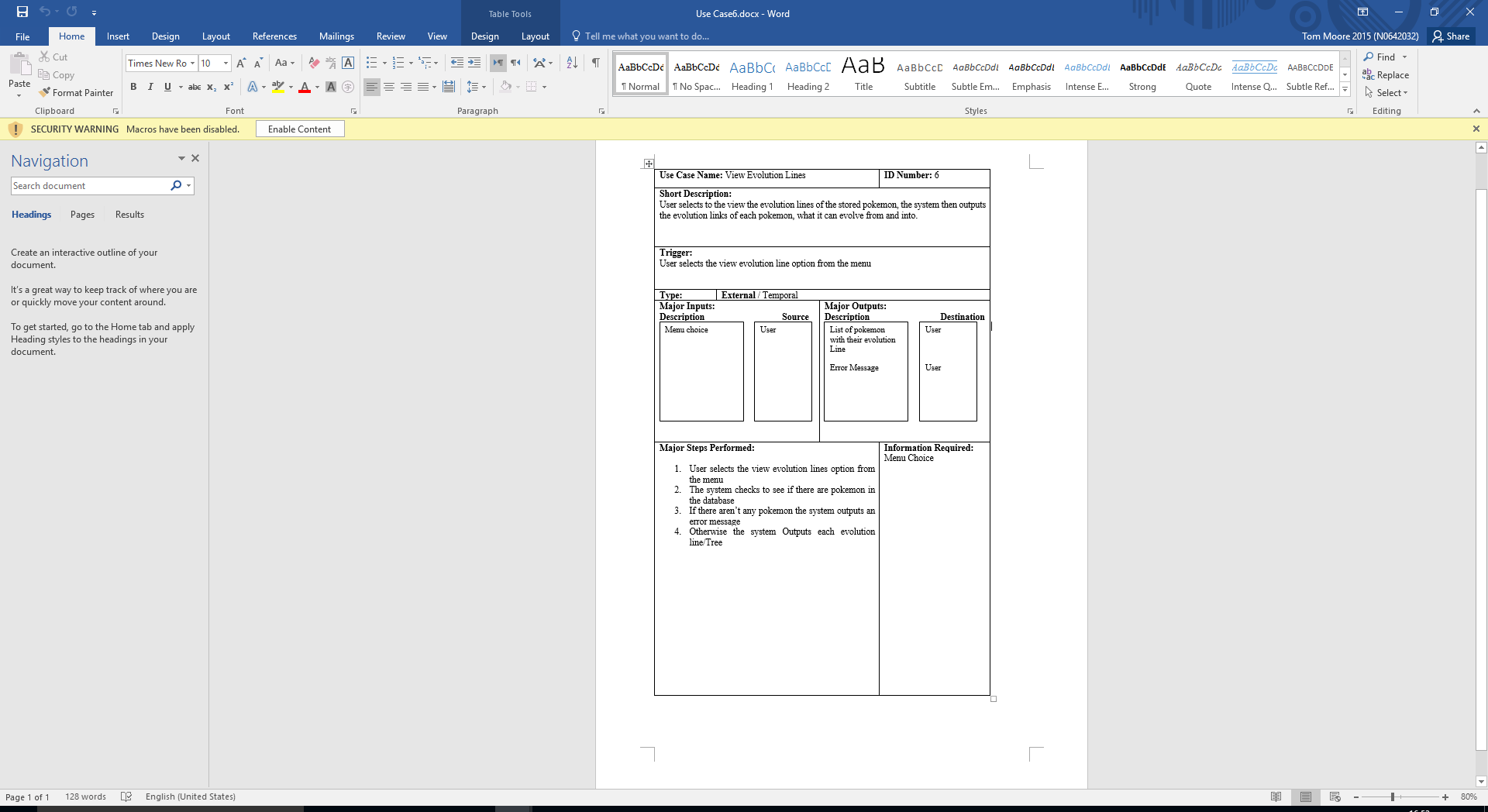
# Appendices

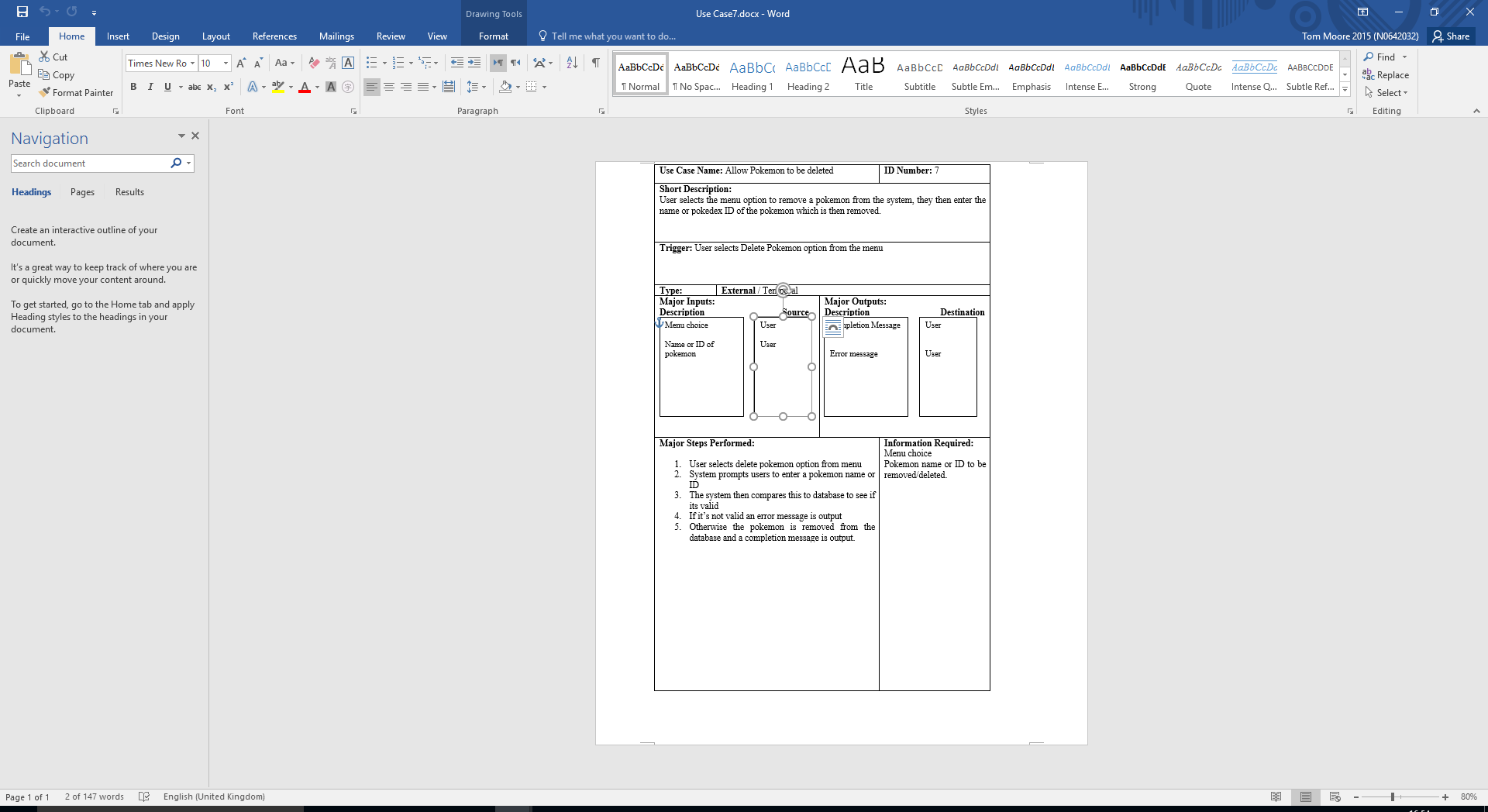
## Use Cases

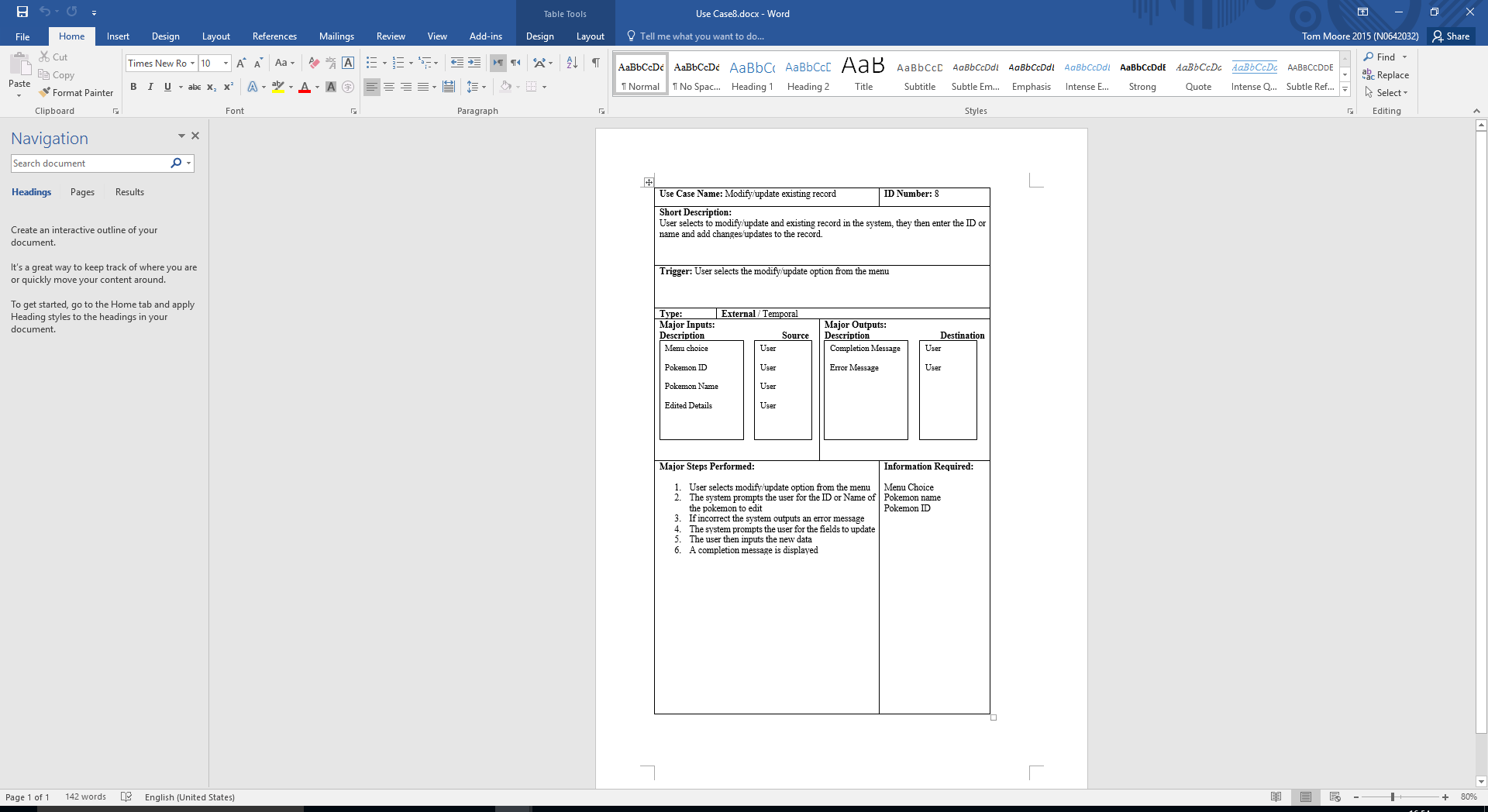








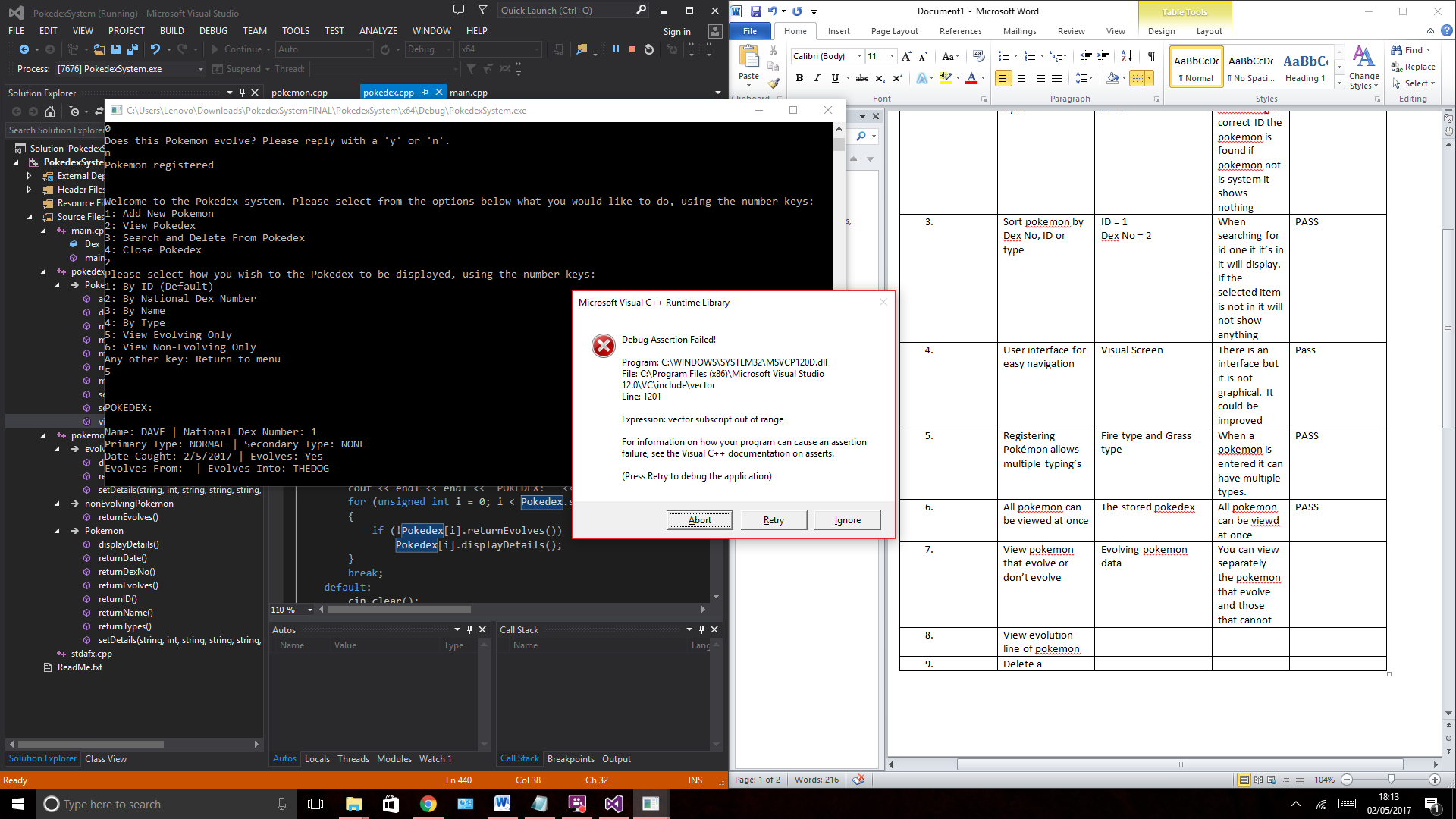
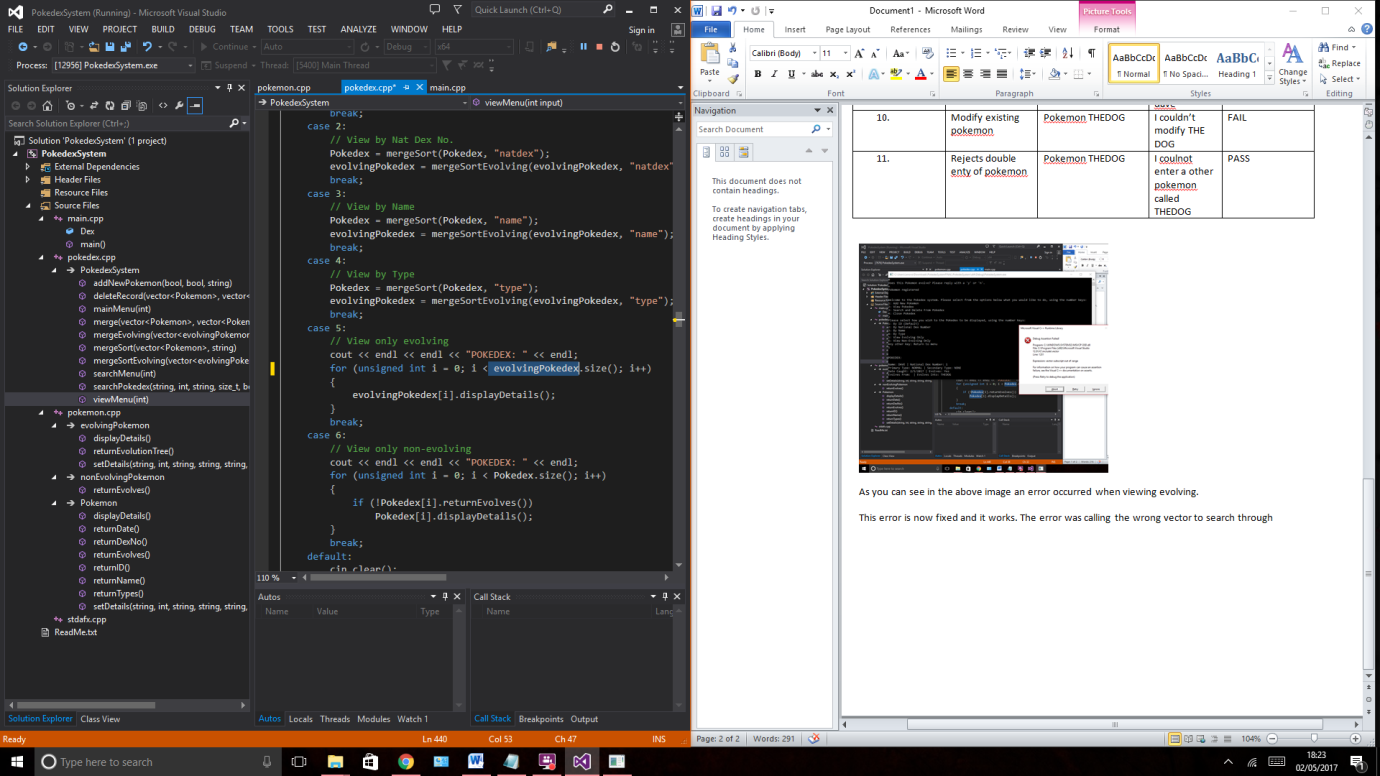




## Testing Table

| Test No. | What’s being tested | Data to test | Result | Pass/Fail |
| --- | --- | --- | --- | --- |
|  | Adding new pokemon | Name = Test  DEX NO = 1  PrimType = 2(fire)  SecondType = 0(none)  Evolve = n | With the entered data the pokemon is registered in the system | PASS |
|  | Search pokemon by Id | ID = 1  Id =3 | When entering a correct ID the pokemon is found if pokemon not is system it shows nothing | PASS |
|  | Sort pokemon by Dex No, ID or type | ID = 1  Dex No = 2 | When searching for id one if it’s in it will display. If the selected item is not in it will not show anything | PASS |
|  | User interface for easy navigation | Visual Screen | There is an interface but it is not graphical. It could be improved | Pass |
|  | Registering Pokémon allows multiple typing’s | Fire type and Grass type | When a pokemon is entered, it can have multiple types. | PASS |
|  | All pokemon can be viewed at once | The stored pokedex | All pokemon can be viewed at once | PASS |
|  | View pokemon that evolve or don’t evolve | Evolving pokemon data | When viewing evolving pokemon it created an error. See error 1 for details | FAIL |
|  | View evolution line of pokemon | Pokemon Dave | When searching for dave it shows me its evolution tree | PASS |
|  | Delete a pokemon | Pokemon dave | After searching I can delete dave | PASS |
|  | Modify existing pokemon | Pokemon THEDOG | I couldn’t modify THE DOG | FAIL |
|  | Rejects double enty of pokemon | Pokemon THEDOG | I couldn’t enter another pokemon called THEDOG | PASS |

## Results



As you can see in the above left image an error occurred when viewing evolving.

This allowed us to rectify the error, identifying that the code was getting the wrong vector length (Pokedex.size()) to loop through to search the evolving pokedex vector, creating the ‘vector subscript out of range’ error. This was fixed by changing the loop to use the correct vector length (evolvingPokedex.size()) which fixed the error, stopping the program from trying to access the evolvingPokedex vector outside its range.

## References

James, Michael. "An Empirical Framework For Learning". Scrum Methodology. N.p., 2010. Web. 1 May 2017.

TutorialsPoint, “Data Structures – Merge Sort Algorithm”, Author Unknown, Date Unknown, (https://www.tutorialspoint.com/data\_structures\_algorithms/merge\_sort\_algorithm.htm )