NOTTINGHAM TRENT UNIVERSITY

SOFTWARE DESIGN & IMPLEMENTATION

Project Report

Project Manager	–	Richard Stone N0782914
Software Architect	–	Thomas Harrison N0833220
Software Developer	–	Samuel Harrison N0787108
Software Tester	–	Stephen Anderson N0786072
Lab Tutor	–	Pedro Machado

Abstract

Group 1 was tasked with planning, designing, implementing, and testing an annotation program for use with convolution neural networks. This will therefore be an application which provides an interface for annotations of a dataset to be fed directly into a neural network. This could, for instance, be annotating the trunks of a tree, or the leaves of a tree and separating them into their appropriate classes in order to train the neural network. This document details all stages of the development, discussing all the tools used libraries used alongside the UML diagrams which describe the function and implementation of the system. The system was implemented to an adequate standard for the purposes that were set out initially. Alongside this, the system was thoroughly tested which will also be documented in this report. The application was written using C++ in Microsoft Visual Studio using the .NET framework and CLR/CLI to integrate the .NET code into the C++ environment. This project can be further improved and built upon by members of the scientific community wishing to deepen the understanding of artificial intelligence and the ways in which it can be applied to modern life through the use of strong programming practices.

Revision History

Table 1 - Revision History

Version	Issue Date	Stage	Changes	Author
0.0.1	20/02/20	Alpha	Added basic Windows Forms Functionality, GUI, and basic file handling.	Samuel Harrison
0.0.2	24/02/20	Alpha	Improved image padding/centring and general GUI fixes.	Samuel Harrison
0.1.0	01/03/20	Beta	Added images being loaded to a list in the left side of the GUI, implemented basic annotation drawing, separated into headers and source files.	Samuel Harrison
0.1.1	01/03/20	Beta	Added the test plan and added the test framework to the project with stub definitions.	Stephen Anderson
0.1.2	29/3/20	Beta	Added more headers, attempting to resolve an issue where test framework would not work.	Samuel Harrison
0.2.0	04/04/20	Beta	GUI updates, added the ability to click and drag to create annotations, added the ability to load a .names file for annotation classes. Added remove annotation button.	Samuel Harrison
0.3.0	05/04/20	Beta	Added search functionality for images, classes, and annotations.	Samuel Harrison
0.3.5	06/04/20	Beta	Added new .names files to be tested and made existing .names more appropriate.	Richard Stone
0.3.6	06/04/20	Beta	Restructuring of the project in an attempt to fix test framework issues.	Stephen Anderson
0.3.7	06/04/20	Beta	Added more accurate header files, resolved conflicts.	Samuel Harrison
0.4.0	22/04/20	Beta	Fixed threading issue by adding '[STAThread]' to main.cpp.	Richard Stone

0.5.0	24/04/20	Beta	Added JSON functionality.	Samuel Harrison
1.0.0	25/04/20	Release	Fixed many GUI bugs, added sort functionality, added resizing, fixed linked list functionality allowing tests to be run.	Samuel Harrison
1.0.1	25/04/20	Release	Completed unit tests.	Stephen Anderson

GitHub link: https://olympuss.ntu.ac.uk/N0786072/SDI-Project

Contents

Tables

Figures

Introduction

Background

In the modern world of computing, artificial intelligence holds a great prominence with its range of uses and applications. Its primary purpose is to automate tasks in the same fashion as a human operator, increasing productivity while maintaining the same degree of precision. To achieve this functionality, a neural network may be implemented – this is a system which uses a set of sample data, called a dataset, to predict future results. Hirose (2013) discusses one such use in image processing; neural networks can be used to identify certain genes or organs in an MRI scan, for example, while Dybowski and Gant (2001) suggest the use of this technology to indicate the presence of diseases via ultrasound examination.

For a convolutional neural network to work, the dataset must first be built from data that has been manually collected. The objective of this project is to design a program that would streamline the process of building a dataset to be as simple and user-friendly as possible. Such a program would be highly applicable to the training of a convolutional neural network; labels would be used to define certain elements in an image, and a greater scope project would use this sample data to identify the same element in other data without manual input. In the case of Hirose's application of neural networks in an MRI scan, the program could allow a user to indicate specific organs in a scan, allowing a larger system to recognise instances of that same organ in other images.

During this project, the group aims to enrich their knowledge of the creation and operation of neural networks, as well as improving team building and working together on a group project.

Aims

The group aims to create an annotation drawing program which will meet all of the requirements outlined in the analysis phase. The functionality of the program should be evident to the user very quickly, which will be achieved through the creation of an intuitive user interface which will be easy for a person to use and understand.

Objectives

For the project to succeed, all of the following criteria must be met:

- Create an application in which labels of various shapes can be added to an image.
- Save data about each annotation.
- Save data about the classes each annotation belongs to.
- Sort and search through annotations, images, and classes.
- Store data using linked lists and other necessary data structures.
- Manipulate already existing annotations (resize, delete)
- Provide a user-friendly interface.

Background Research

Table 2 - Background Research

Tool/Library	Purpose	Notes
Microsoft Visual Studio	IDE	The selected IDE used by the group. This is due to the group having the most experience using Visual Studio as an IDE.
Windows Forms System.Collections.Generic Systems.ComponentModel Systems.Data Systems.Drawing Systems.Linq Systems.Text Systems.Threading.Tasks System.Windows.Forms	GUI/Functionality	Uses the .NET framework. The vast majority of the program will be created using these tools, including the GUI and creation of annotations. These have been selected as the software developer has the most experience with these frameworks and libraries.
Common Language Runtime (CLR) & Common Language Infrastructure (CLI)	Implementation	CLI allows the integration of various programming languages on various machines in order to improve reliability. CLR is the common runtime for all .NET languages.
Microsoft Unit Testing Framework for C++	Testing	Standard Visual Studio unit testing framework, this will be used on back-end testing and is written in C++.
GitHub	Version Control	Use of GitHub in order to keep track of updates and help the group identify where problems may have arisen. This also ensures that
Slack	Communication	Use of Slack in order to split communication into sections and allow for pinned items which may be important. The channels will be structured as such: general, planning, design, development, testing, announcements.
Doxygen	Documentation	Help to create reliable documentation quickly.
JSON For Modern C++	Formatting & Parsing JSON	Used to format and parse JSON in a C++ programming environment.

Project Analysis

Requirements Analysis

Table 3 - Requirements Analysis

Requirement Number	Requirement Description	Implications	Tasks
1	Must be able to open image files.	Will have to make use of file handling and using libraries to allow an image file to load into the interface.	**T1/T4** Implement file handling to allow the user to load an image file into the interface.
2	Must be able to open annotation files.	Will have to make use of file handling and using libraries to allow an annotation file to load into the interface.	**T1/T4** Implement file handling to allow the user to load an annotation file into the interface.
3	Must be able to close image files from the current session.	Will have to make use of file handling within the GUI to allow an image file to be closed without closing the current session.	**T7** Implement GUI elements to allow the user to close an image file from the interface.
4	Must be able to close an annotation file from the current session.	Will have to make use of file handling within the GUI to allow an annotation file to be closed without closing the current session.	**T7** Implement GUI elements to allow the user to close an annotation file from the interface.
5	Must be able to save an annotation file.	Make use of file handling to save the annotation file to the user's local storage.	**T4/T7** Implement GUI elements and file handling functionality to allow the user to save a file.
6	Must be able to append to an existing annotation file.	Make use of file handling, ensure that the file's previous	**T4** Implement file handling functionality to allow the user to append a file.

Requirement Number	Requirement Description	Implications	Tasks
		contents are not deleted.	
7	Should be a panel in the GUI which lists compatible image files.	Will have to create a method of file handling within the GUI.	**T4/T7** Develop a portion of the UI dedicated to listing compatible file types.
8	Must be able to sort annotation files by name or date.	Must be able to make use of both file handling and sorting algorithms.	**T5** Create a suitable sorting algorithm for the files.
9	Must be able to create annotations by drawing a rectangle, triangle, trapezium, or free form polygon with up to eight vertices.	Must use libraries and frameworks to create selection tools annotations and interact with the GUI using the mouse pointer.	**T1/T2** Create necessary classes and functions for implementing annotations of various shape. Use appropriate data structures.
10	Must be able to remove annotations in an intuitive way.	Must integrate well with the function of creating annotations and be cleared from the GUI, therefore will need to learn in-depth GUI handling.	**T1/T2** Create necessary functions within classes for removing annotations from the interface and data structures.
11	Must be able to edit annotations by increasing the size or moving vertices.	Make use of libraries to edit already made annotations, this must remove the original unedited annotation in favour of the new edited annotation.	**T1/T2** Implement appropriate functions for editing annotations.
12	Must be able to select each different shape to draw an annotation.	Must be able to select between a rectangle, triangle, trapezium, and free form with up to eight points. Make use of drawing libraries.	**T7** Must implement functionality in the user interface to allow switching between different shapes in an intuitive way.
13	Must be able to copy and paste annotations.	Must make use of suitable copy and paste integration with the application.	**T7** Ensure that through conventional methods (ctrl+c, ctrl+v) the interface allows the

Requirement Number	Requirement Description	Implications	Tasks
			user to copy and paste annotations.
14	Must visualise the name of the class on top of the shape.	Must learn how to add labels to annotations.	**T2/T3** Create a label which tells the user the name of the annotation's class.
15	Must be able to search for a annotation.	Will have to make use of a suitable search algorithm to search for an annotation.	**T5** Select a suitable search algorithm for this task and implement the code for it.
16	Must be able to sort all annotations by date and title.	Will have to make use of a suitable sorting algorithm to sort annotations.	**T5** Select a suitable sorting algorithm and implement the code for it.
17	Must be able to convert data to/from JSON file format.	Will have to make use of libraries for writing to a .JSON file.	**T4** Select suitable libraries for file conversion and implement the code for it.
18	Must be able to store annotations in a suitable way, most likely linked lists.	Make use of data structures such as linked lists.	**T3** Implement code to store annotations in a linked list.
19	Data about each annotation must be stored.	Making use of file handling to store data of number of images, image file name, number of shapes per image. For each shape: shape type, Point_1(x,y), Point_N(x,y)...	**T3** Store data about each annotation in an appropriate data structure, and then make use of file handling to save the information to the user's storage device.
20	Must be able to store images in a suitable way, most likely arrays.	Make use of data structures such as arrays.	**T3** Implement code to store images in an array.
21	Must provide a suitable and intuitive user interface.	Must select an appropriate GUI library/framework.	**T7** Create a user interface which is

Requirement Number	Requirement Description	Implications	Tasks
			simple to use and not overly verbose.
22	Must be able to classify the annotations to group similar annotations.	Will need to have annotations created in groups and have them identifiable according to similar characteristics.	**T2** Implement relevant architecture to allow for annotations to be grouped with other annotations

Risk Assessment

Table 4 - List of Risks

Risk	Likelihood (1 – Very Unlikely, 5 – Very Likely)	Impact (1 – Very Low, 5 – Very High)	Impact on Project	Mitigation Plan
A group member suspends their study	2	4	Group members suspending their study may lead to a vastly increased workload on other members.	Ensure that members are informed as early as possible, and that this information is passed onto the module leader.
Absence due to illness	3	2	Project time may be wasted, particularly if the absence is during a group meeting.	Ensure that the absent group member is fully briefed on any meetings or aspects of the process which may have been missed.
Allocated tasks are not completed, or not completed on time.	3	5	The project of the quality may be hindered due to rushed work, or the workload of other members may increase	Internal deadlines will be set; however work will be expected to be submitted to the project manager well before this deadline. The project manager will ask for frequent, measurable updates using yes/no questions. E.g., can the program do this yet. Github will be used to keep track

Risk	Likelihood (1 – Very Unlikely, 5 – Very Likely)	Impact (1 – Very Low, 5 – Very High)	Impact on Project	Mitigation Plan
				of members' contributions.
Lack of communication	3++	4	May lead to poorer quality of work or even incorrect work.	Frequent face-to-face meetings, and a group chat has been set up on WhatsApp in order to keep up with all members. Slack will also be used.
Poor time management	4	5	Not managing time effectively may lead to an incomplete project.	Measurable goals should be set according to the date, the project manager will frequently communicate with all members for updates.
Project Assumptions	4	3	Underestimating the difficulty of particular aspects may lead to an incomplete or rushed product.	Every aspect will be rigorously discussed and preferably researched to ensure that all members fully understand the difficulty of each aspect.
Technical Failure	1	4	Technical issues such as a computer shutting down may lead to a loss of work.	Ensuring that frequent local backups and cloud backups will be taken to vastly reduce the amount of work lost.

Risk	Likelihood (1 – Very Unlikely, 5 – Very Likely)	Impact (1 – Very Low, 5 – Very High)	Impact on Project	Mitigation Plan
Merge Conflicts	3	3	Two developers attempting to commit changes to the same file at the same time may lead to loss of work.	Developers will communicate when they need to commit, and in the event that a merge conflict should happen, a commit will never be forced and instead will be merged through strong communication.
Member conflicts.	3	3	Excessive member conflicts and lack of consensus may slow down the development process or lead to unhappy developers possibly resulting in lower quality work.	The group will be democratic, if three out of four members agree, that will be taken as the consensus. If it is split fifty-fifty, further discussion will take place until the project manager decides. Compromise will be necessary.

<u>Time Plans</u>

All stages will be committed to a GitHub repository where possible. Slack will be used for communication between members.

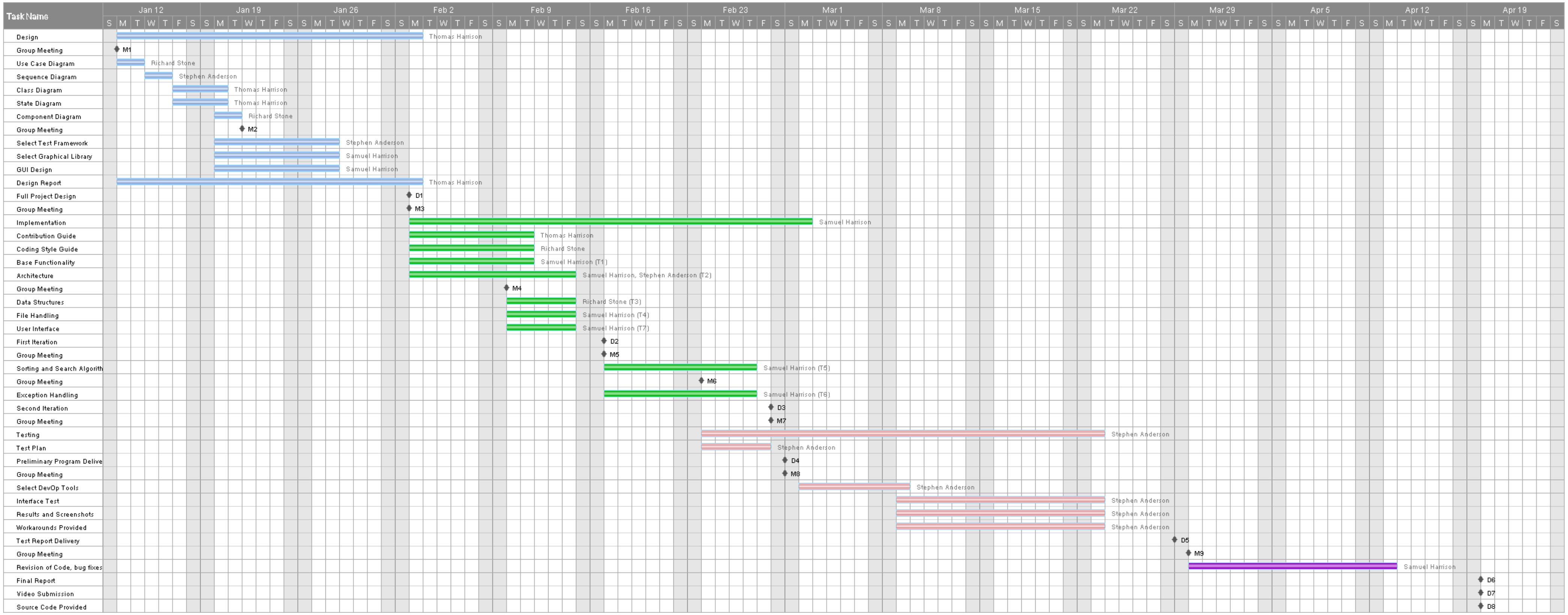


Figure 1 Gantt Chart

Assumptions

The group assumes that the user will have a computer running Windows 10 64 bit with at least semi modern hardware, this is necessary as Windows 10 is the only operating system which has been fully tested and the program may require more than 4GB of memory, which a 32 bit operating system will not support. The group also assumes that the user has a functional keyboard, mouse and monitor in order to interact with the user interface and allow full functionality of the program.

Adopted Coding Standards

The group adopted various coding standards prior to the implementation of the application. All such standards which have been adopted can be found in 'Appendix 1 – Adopted Coding Standards' in the form of the coding style guide and the contribution guide. The group created a coding style guide which outlines a very clear method of creating readable and maintainable code which will make the experience for future developers who might build upon the application a much more streamlined one. Relevant information such as the C++ version used, commenting etiquette, and variable naming conventions among other aspects of the implementation style. This ensures that the program will be coherent and robust.

Alongside this, a contribution guide was created which details the correct protocol for behaviour and committing work to the GitHub repository. The importance of the ability to work in a team without discrimination or arguments is outlined in this document, and therefore ensures that all team members will be fully aware of the repercussions should they break the rules of the contribution guide, including how the project manager should handle such situations an how further action may be taken as necessary.

Project Design

Use Case Diagram

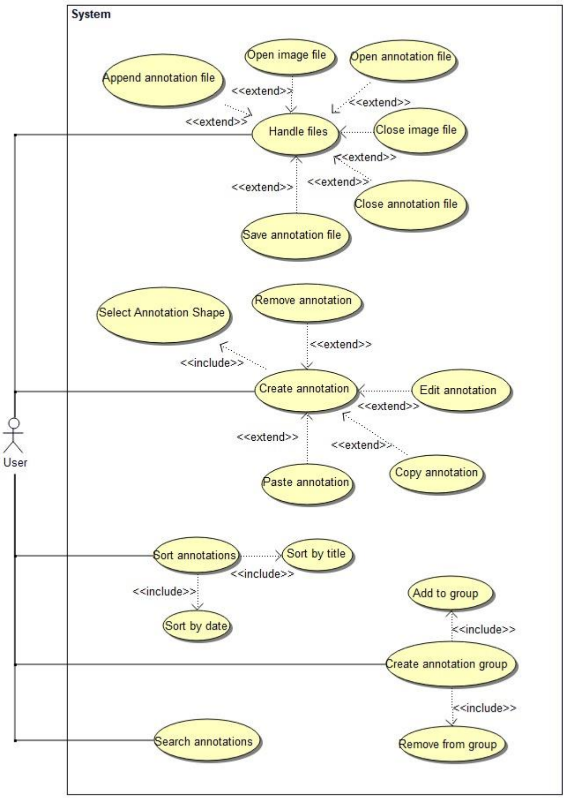


Figure 2 Use Case Diagram

In this use case diagram, the user is the sole actor; they are the only one that interacts directly with the program. The user can handle files, manipulate annotations, search annotations, and manipulate annotation groups as base functionality.

Handle Files allows the user to perform various types of file handling to be able to use the program. This extends to append annotation file, open image file, open annotation file, close image file, close annotation file, and save annotation file. <<extend>> is present because it is extended functionality of the base part (handling files).

Create annotation lets the user create an annotation. These annotations can also be removed, edited (resized or repositioned), copied, and pasted. Select annotation shape is included as this cannot be done without first creating the annotation. Create annotation extends to remove, edit, copy, and paste; these can in theory be done without creating an annotation and do not depend on the base use case.

Sort annotation allows for the sorting of annotations by title and date. Sort by title and sort by date depend on the base use case and cannot be done without the ability to sort in the first place.

Create annotation group allows the use to group annotations together. This includes add to group and remove from group, which are dependent on the base use case.

Search annotations lets a user search through the annotations.

Class Diagram

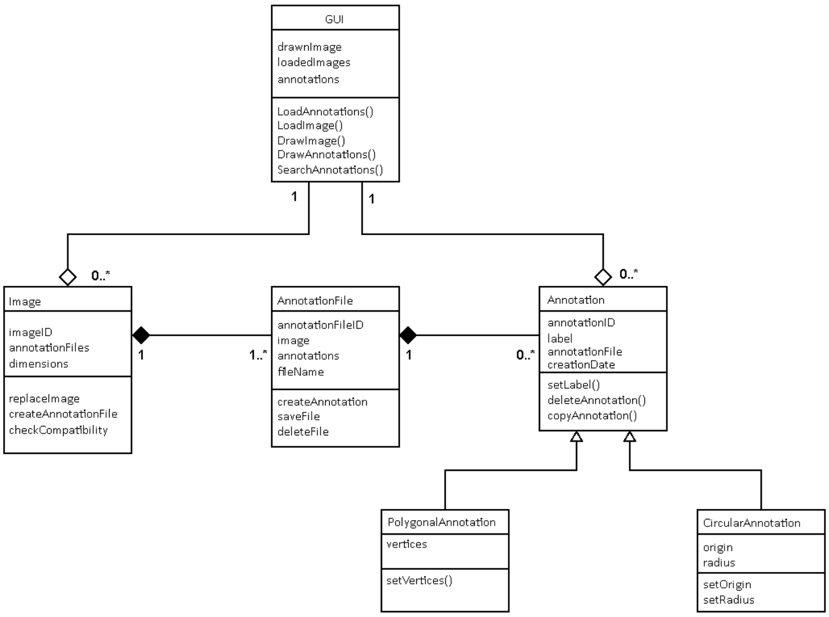


Figure 3 Class Diagram

This diagram consists of the key classes for the program, along with their respective attributes and operations.

The GUI represents the canvas on which images and annotations are displayed. Each instance of the program has one GUI, which has multiple images loaded and multiple annotations. The GUI has attributes for the image currently being drawn, any images that are loaded into memory, and annotations. From the GUI, the program may load images, annotations, and display them.

Each image has a unique ID to differentiate it from other images, a list of annotation files associated with that image, and the image size and dimensions. Operations include the ability to replace the image, check compatibility when the image is loaded, and create an annotation file to be associated with the image. An image may have multiple annotation files associated with it.

Annotation files have a unique ID, a relation to its associated image, and a list of annotations contained within the file. Its operations include creating annotations, saving the annotation file, and deleting the file. Each annotation file is only associated with one image but can be associated with many different annotations.

All types of annotation have a unique ID, a label (or name) describing the annotation in a string format, and the ID of the associated annotation file. With all annotations, the user may set and edit the label, copy and paste the annotation, or delete it. Each annotation is only associated with the one GUI, and one annotation file.

There are two subclasses of annotation: polygonal and circular. Polygonal annotations use a linked list of vertices which can be set in the relevant operation, while circular ones have an origin coordinate and radius, with operations that allow the user to define these.

Sequence Diagrams

In the program environment, the user is the only entity capable of interacting directly with the program. As such, they are the only actor in these sequence diagrams.

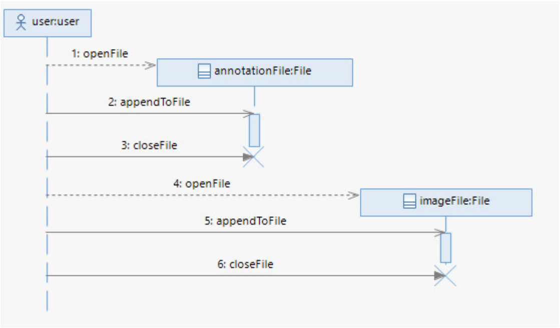


Figure 4 Sequence Diagram for file handling

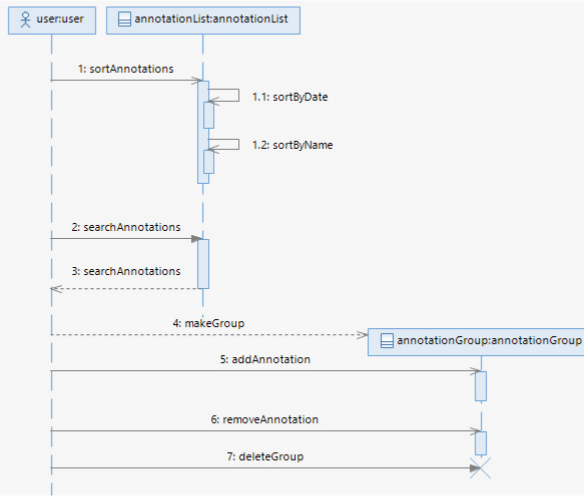


Figure 5 Sequence diagram for annotation file manipulation

Figure 3 demonstrates a user's file handling ability for images and annotation files. The user first must open the relevant file; from here, they can make any amendments to the file (for example, in the image's case, replacing the source file), before closing it again to prevent a memory leak.

Figure 4 shows the actions the user can perform on the annotation file. Users can sort through the existing annotations that are stored in a list. This list can be sorted by mot recent, oldest, A-Z or Z-A. The user would also be able to use a search box to find a particular annotation in the list. The user can further classify the annotations by creating or deleting group which they can add and remove annotations to.

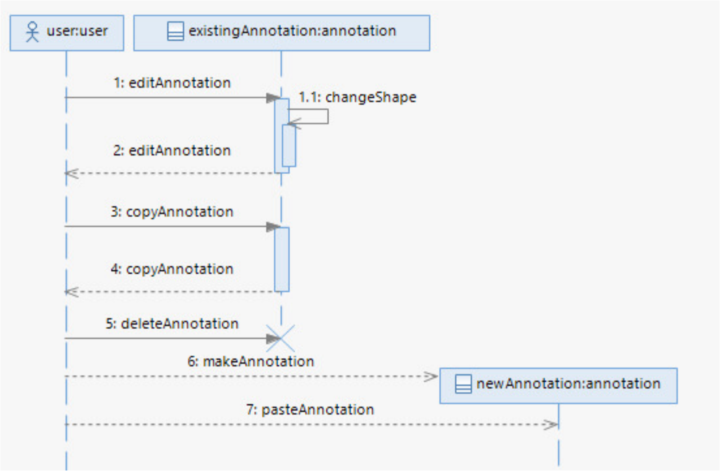


Figure 6 Sequence diagram for annotation manipulation

The sequence diagram in Figure 5 represents the process of editing an annotation. Users can edit any existing annotations (i.e. changing its shape or name) and can copy and paste these annotations as well as being able to delete them. When making a new annotation, the user can specify the type as well as the position and name of this annotation.

State Diagram

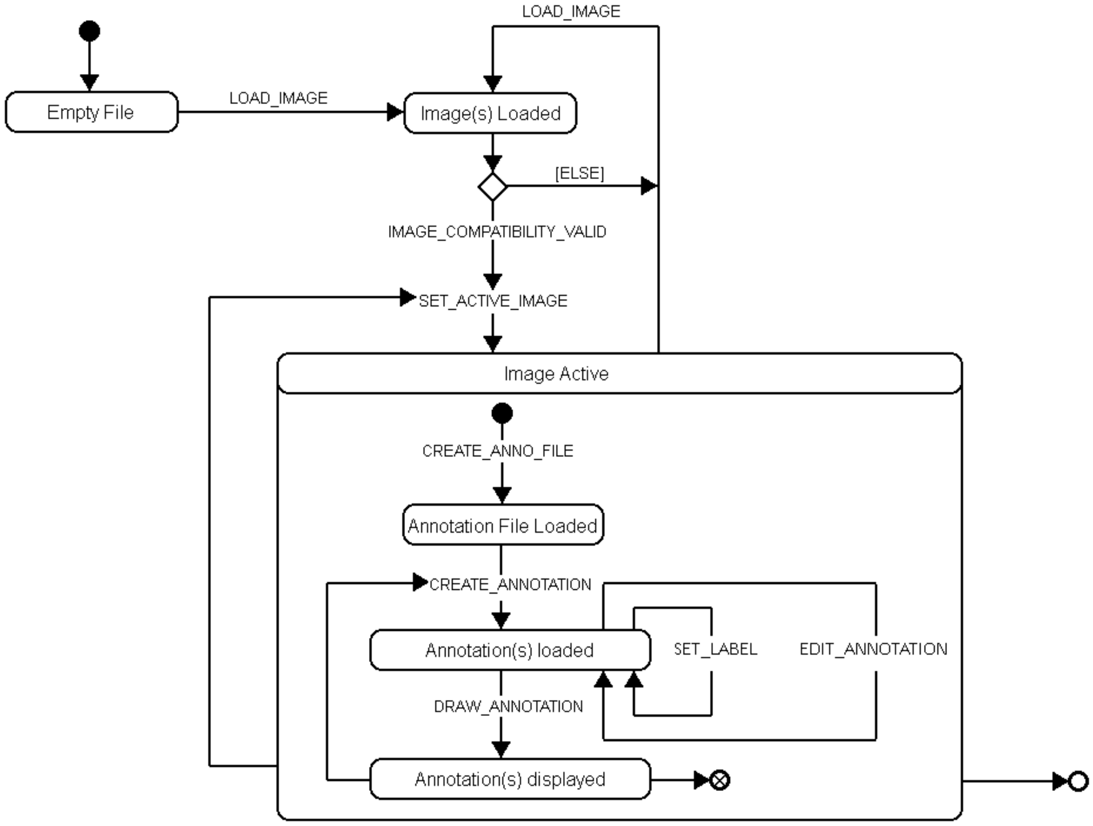


Figure 7 State Diagram

This state diagram demonstrates the process of starting the program, loading an image, and creating annotations. When the program is first started, the user must load an image. From here, the compatibility is checked; if it passes, the image may be set to active and displayed, otherwise the user must load a different image.

Within an active image, an annotation file can be created, with annotations either being loaded or created within the program. When an annotation is loaded, the label can be set/replaced and the annotation region itself may be modified. The user may swap the current active image or load new ones at any time. The final state is an image active in the interface, which may include annotation files and annotations.

Component Diagram

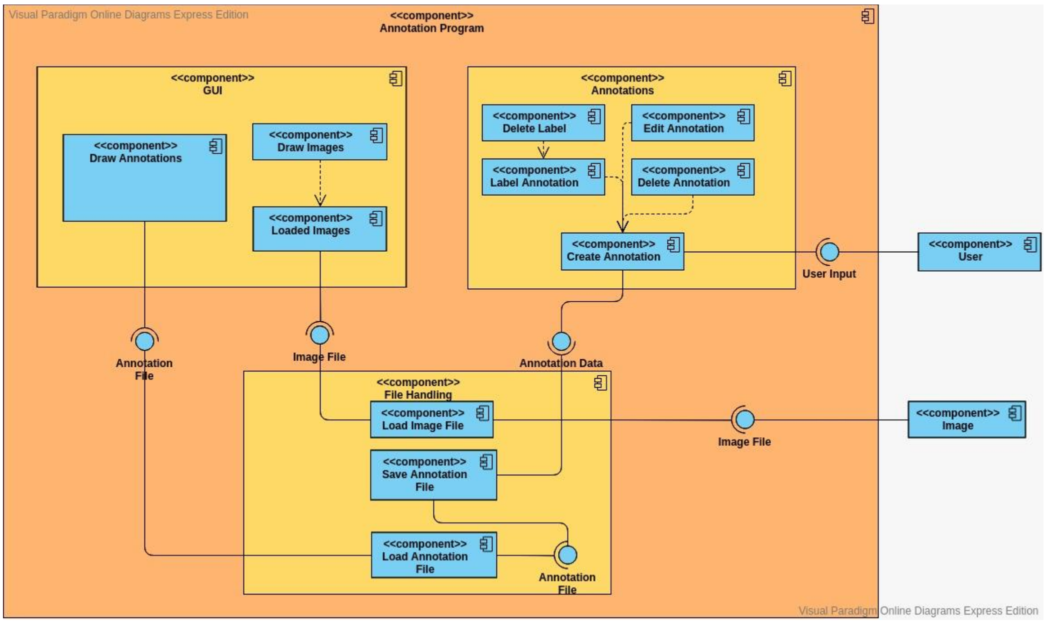


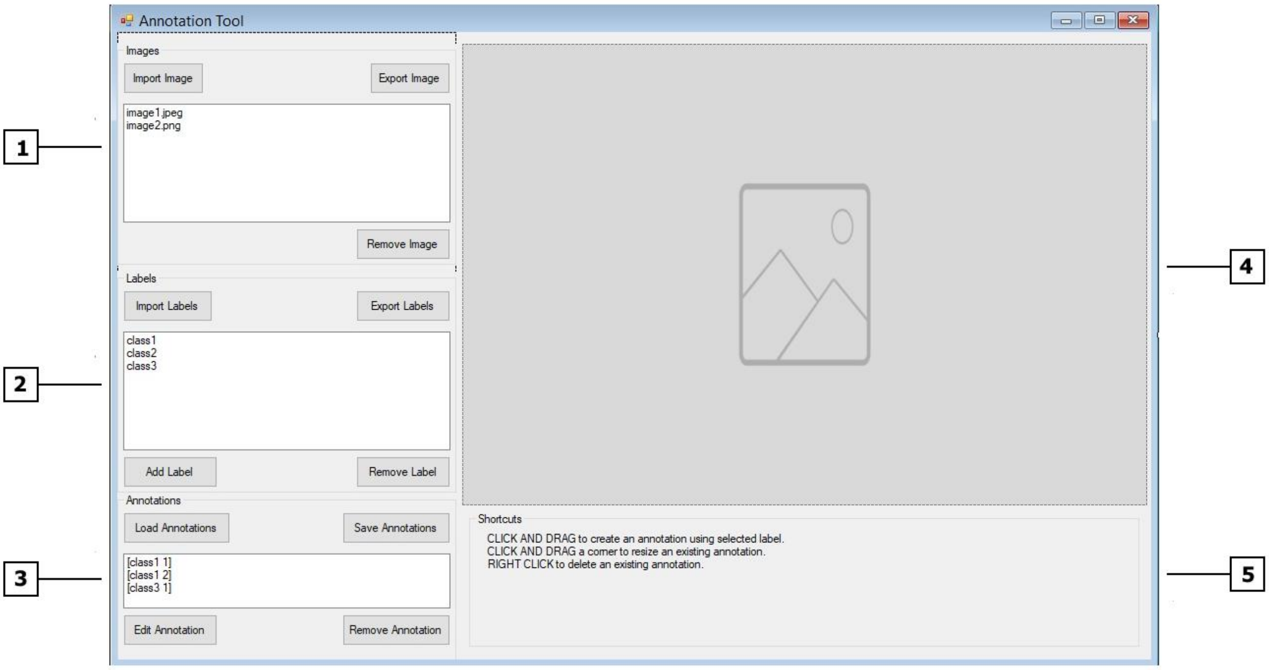
Figure 8 Component Diagram

A user provides the user input interface. This is then used by the annotations component to create an annotation. Delete annotation, edit annotation, label annotation, and delete label all depend on the create annotation function and therefore requires the user input interface. When an annotation is manipulated, annotation data is created as an interface.

This interface is used by the file handling component, which can then use that annotation data to save to an annotation file and produces an annotation file as an interface. The load annotation file component will then use that interface to load the annotation file. The load image file component then takes the image file interface provided by the image component and produces an image file to be loaded.

The GUI component uses both the annotation file and image file interfaces provided from the file handling component to load the image and annotation file to drawn them to the screen.

GUI Mock-up



CLICK AND DRAG to create an annotation using selected label.
CLICK AND DRAG a corner to resize an existing annotation.
RIGHT CLICK to delete an existing annotation.

Figure 9 GUI Mock-up

Figure 8 shows a mock-up design of the User Interface for the program.

1. **Image List:** A list of loaded images. Images can be imported, exported, and removed using the buttons located around the list. This can include .jpeg, .png, .bmp files.
2. **Label List:** A list of labels included in the currently active image, with buttons which allow the user to import, export, create, and remove labels from a .names file.
3. **Annotation List:** A list of annotations present in the currently active image. These annotations can be loaded from a file, saved to a file, edited, and removed using the buttons surrounding the list.
4. **Canvas:** The area in which an image from the image list is displayed on-screen. This is where the user can create a manipulate annotations.
5. **Shortcut Information:** Information is displayed at the bottom of the screen about command inputs the user can perform to create and edit annotations.
   - Clicking and dragging a space on the canvas creates an annotation.
   - Clicking and dragging the corner of an existing annotation resizes it.
   - Right clicking an annotation deletes it.

Test Plan

All test scenarios will be defined using the following test table:

Table 5 - Sample Test Table

ID:		Description:	
Test Type:		Success Criteria:	
Number of Attempts:		Comments:	
List of Equipment/requirements			
Setup Instructions			
Failure Correction Procedure			
Engineer/Technician			
Individual Results:			
Screenshots			

The tests will be a mixture of black box testing, white box testing, unit testing (found in the project's source files), integration, and acceptance testing. All of these used in conjunction with each other will provide a much more robust product which will in theory meet the criteria defined by the client and provide a more user-friendly experience when interacting with the software. All test tables and scenarios can be found in Appendix 2.

Conclusions and Future Work

Overall, the group communicated well throughout the entire process of the project. For the first deliverable, the requirements and risks were very easily and readily identified by the group with very few issues. In the second deliverable, the tasks were divided equally amongst all members including UML diagrams and background research. For the third deliverable, a large amount of the development work had been achieved and achieved to a good standard.

The only deliverable which became problematic was the fourth deliverable, the software tester's deliverable. The issues arose because the selected test framework (Boost) was designed for C++ code, which the group's project was written in, however many of the libraries used (particularly the System library) were written with .NET code rather than C++, using CLI/CLR to bridge the gap between them and allow the .NET code to interact with the group's C++ code. In this way, the test framework selected did not recognise the .NET code as it was designed to work solely on C++, and therefore created many issues in terms of unit testing. Despite this, the software tester and software developer created a solution by converting anything which uses System type data types into standard C++ types, which allowed the bulk of the unit testing to work, while other manual tests were created for the interface and front-end functionality.

Thus, although the fourth deliverable was by far the least straight forward, this was not through a lack of effort on any group member's part, as the tester attempted to use five different frameworks before settling upon Microsoft Unit Testing Framework and the developer had to restructure a large proportion of the code. In future, this could have been potentially been mitigated with stronger research into which libraries and frameworks to use, however given the niche nature of this project this likely would have proven difficult.

In terms of the design phase, the software architect was instrumental in the creation and description of various UML diagrams and the GUI mock-up, while also helping strongly with the final report, while the project manager made sure to keep track of meetings and request frequent updates from each member while also contributing to every deliverable in some way and ensuring that workload was reasonably balanced.

Overall, despite the group running into many issues, through communication and strong effort the team overcame these problems and was able to achieve most of what was set out in the initial design phase.

References

Hirose, A. (2013). *Complex-Valued Neural Networks: Advances and Applications*. Wiley-IEEE Press. [Accessed 20/04/2020]

Dybowski, R., & Gant, V. (Eds.). (2001). *Clinical Applications of Artificial Neural Networks*. Cambridge: Cambridge University Press. [Accessed 20/04/2020]

GitHub. 2020. *Nlohmann/Json*. [online] Available at: https://github.com/nlohmann/json [Accessed 21 April 2020].

Individual Contributions Per Member

Richard Stone – Project Manager

- Overall report structure and piecing together
- Abstract
- Background Research
- Requirements Analysis
- Risk Assessment
- Time Plans
- Assumptions
- Coding Style Guide
- Use Case Diagram
- Component Diagram
- Conclusions and Future Work

Thomas Harrison – Software Architect

- Introduction
- Background Research
- Contribution Guide
- Class Diagram
- State Diagram
- GUI Mock-up
- Written aspect of the design

Samuel Harrison – Software Developer

- GUI Mock-up
- Entirety of development
- Doxygen
- Assisted with testing

Stephen Anderson – Software Tester

- Requirements Analysis
- Sequence Diagrams
- Test Plan
- Test Report
- Unit testingJust
- Assisted with development

Overall Reflection of the Work Done

Overall, the group communicated well throughout the entire process of the project. For the first deliverable, the requirements and risks were very easily and readily identified by the group with very few issues. In the second deliverable, the tasks were divided equally amongst all members including UML diagrams and background research. For the third deliverable, a large amount of the development work had been achieved and achieved to a good standard.

The only deliverable which became problematic was the fourth deliverable, the software tester's deliverable. The issues arose because the selected test framework (Boost) was designed for C++ code, which the group's project was written in, however many of the libraries used (particularly the System library) were written with .NET code rather than C++, using CLI/CLR to bridge the gap between them and allow the .NET code to interact with the group's C++ code. In this way, the test framework selected did not recognise the .NET code as it was designed to work solely on C++, and therefore created many issues in terms of unit testing. Despite this, the software tester and software developer created a solution by converting anything which uses System type data types into standard C++ types, which allowed the bulk of the unit testing to work, while other manual tests were created for the interface and front-end functionality.

Thus, although the fourth deliverable was by far the least straight forward, this was not through a lack of effort on any group member's part, as the tester attempted to use five different frameworks before settling upon Microsoft Unit Testing Framework and the developer had to restructure a large proportion of the code. In future, this could have been potentially been mitigated with stronger research into which libraries and frameworks to use, however given the niche nature of this project this likely would have proven difficult.

In terms of the design phase, the software architect was instrumental in the creation and description of various UML diagrams and the GUI mock-up, while also helping strongly with the final report, while the project manager made sure to keep track of meetings and request frequent updates from each member while also contributing to every deliverable in some way and ensuring that workload was reasonably balanced.

Overall, despite the group running into many issues, through communication and strong effort the team overcame these problems and was able to achieve most of what was set out in the initial design phase.

Appendix

Appendix 1 – Adopted Coding Standards

Coding Style Guide

C++ Version

All code should be submitted using C++ 17, the latest version available.

Where possible, only standard libraries should be used, however due to the nature of this assignment graphical frameworks and test libraries from outside of the standard library will need to be used.

Header Files

Every .cpp file should have its own .h file, the .h file should contain declarations for classes and functions used in the associated .cpp file.

This improves the usability and modularity of the code.

Header files should compile on their own and therefore be self-contained. They should also end in .h for ease of understanding.

Order of Includes

Header files should be included in the following order: Related headers, C system headers, C++ standard library headers, other libraries' header, this project's header.

Separate each included header file type with a line. For example:

```
#include “relatedheader1.h”


#include <CSystemHeader1.h>
#include <CSystemHeader2.h>

#include <StdLibHeader1.h>
#include <StdLibHeader2.h>

#include “this/ProjectsHeader.h”
```

Scoping

Namespaces

Namespaces should be used to separate distinct scopes and will therefore prevent conflicts in the global scope, such as global variables with the same name. Namespaces should have unique, descriptive name of what the scope achieves.

Non-member, Static Member, and Global Functions

Always place non-member functions in a relevant namespace, avoid using a class to group static member functions, and avoid using entirely global functions.

Variables

Use variables as locally as possible. Prefer declaring variables in a function, then a class, then a namespace, and avoid using global variables overall.

Classes

Try to use classes as frequently as possible where appropriate.

Implicit Conversions

Avoid using implicit conversions, instead use the explicit keyword for conversions.

Structs or Classes

In general, prefer to use a class over a struct. As classes and structs behave very similarly in C++, it is best to avoid confusion and opt to use one consistently. As classes are more conventionally used, use a class rather than a struct.

Access Control

Make attributes of a class private in general. As an exception, attributes can be public if they are constants.

Order of Declaration

Declare public class members first, then protected, then private. Each different level of access should be grouped together, with all public together, all protected, and all private. Generally, related attributes should be declared together to minimise confusion.

Functions

Generally, return values should be used rather than output parameters for functions.

Short Functions

Functions should be kept as short as reasonably possible. This improved code readability and generally keeps the code tidy. Functions should not exceed 40 lines, though this is not a strict rule. If a function exceeds 40 lines, the programmer should consider where the function can be broken up into smaller function. For instance, if adding values to an array and then sorting the array in the same function, perhaps sorting the array can be separated into its own function. This should be left to the judgment of the developer.

Naming

Naming should be consistent for each entity. This is so that the naming style can be used to correctly identify what the entity is, for instance, it should be immediately obvious whether the developer is using a constant value or a variable simply at a glance from the naming convention used by each.

General Naming Rules

Names should be as readable as possible, regardless of what the entity being named is. It should be as obvious as possible to other people reading the code what the purpose of the named item is based on a glance at the name. With that being said, names shouldn't be overly long and clunky, as this can clutter the code and decrease the overall quality and therefore it is important for the developer to find a balance between descriptive naming and concise naming. Where possible, magic numbers should be avoided and instead have a descriptive but concise variable name assigned to them. Widely accepted, common names (for example, the use of "i" for an iterator in a for loop) are acceptable when used in the correct context.

File Names

File names should be all lowercase using underscores to separate words. For example, a good file name would be: "trees_image.png" whereas a bad file name would be: "Treesimage.png".

C++ files should end in .cpp and header files should end in .h, and all files should adhere to the general naming rules with reference to being descriptive and concise.

Class and Struct Names

Classes and structs should be named with a capital letter for each new word, without underscores. For example: "GoodClassName" or "GoodStructName". This can therefore differentiate them from variables names, which use a different naming convention.

Variable Names

Variable names should be named using camel case, and therefore have the first word starting with a lower-case letter, and every word following starting with an upper-case letter. An example of this would be: “goodVariableName”. This helps to differentiate from other named entities.

Data members included in types such as classes or structs should follow the same naming convention as regular variables.

Constant Names

Constant should be named in the same way as variables; however, the first ‘word’ will always be a lower-case letter ‘k’, with every following word beginning with an upper-case letter. An example of this: “kMonthsInAYear”.

Function Names

Function names should follow the same naming convention as classes and structs, and therefore all contain an upper-case letter at the start of each word, for instance: “GoodFunctionName”. Parameters should also be given sensible names, using variable formatting.

Namespace Names

Namespaces should be in all lower case with no separating characters, for instance: “givennamespace”. This helps to differentiate namespaces from classes, functions, and variables.

Comments

Comments should not be used overly frequently. Good code should not require the use of excessive comments but should rather be readable on its own as much as possible. However, there will be times where comments are necessary, but prior to including a comment the developer should ask themselves if the code has descriptive variable names and is formatted well.

Both // and /* */ are acceptable styles of commenting.

Function Comments

Comments should be included before the definition of each function. The comments should describe what the function does and how to use it. Comments may be used inside the function to describe a particularly complex block of code (for example a long mathematical function) which would make it easy to understand but adhere to general comment guidelines as described above.

An example of a well commented function:

```
// This function takes in an array and sorts it using an insertion sort.
// Example:
//        double [] sortedArray = InsertionSort(nonSortedArray);

double InsertionSort(givenArray) {

 …
};
```

Variable Comments

The name of a variable, if using the naming guidelines described above, should be enough in explaining what the variable is for. On rare occasions, comments may be required such as in the case of global variables. When a global variable is declared, they should have a comment alongside them explaining what the global variable does and why it must be global rather than local.

This also help the developer to think about whether the global variable needs to be global or not.

Spelling, Punctuation, and Grammar

All comments should be given with correct spelling, punctuation, and grammar which should be easily read and descriptive. Comments should be written in prose and use as simple language as possible to help the next developer to understand what the code is doing. If the comment does not make sense, the comment is likely not very useful.

Formatting

Line Length

Lines of code (excluding comments) should be no longer than 100 characters including spaces, this helps to keep the code as readable as possible.

Spaces or Tabs

Tabs should be used rather than spaces to format block of code such as if statements, functions, and loops.

Function Declarations and Definitions

Function definitions should be done all on the same line, including the return type and the parameters, but excluding the curly braces. An example of this:

```
ReturnType ClassName::FunctionName(Type parOne, Type parTwo)
{

	…

}
```

Curly braces should begin on the next line, in line with the start of the function while closing in line with the first curly brace, as shown above.

Calling Functions

Functions should where possible be called on one line only, rather than wrapping onto another line. If functions must be called across multiple line, separate the parameters rather than the rest of the definition. For example:

```
Type result = FunctionName(aRidiculouslyLongArgument, parTwo,
                           parThree, parFour);
```

Conditionals, Loops, and Switch Statements

For conditionals, for example an if/else statement, the formatting will be much the same as for functions but with the addition of the next section of the statement, for instance an else for a conditional statement. An example:

```
if (condition)
{
	…
}
else
{

	…

}
```

Variable Initialisation

A variable can be declared as per the developer's preference, using either =, ( ), or { } as all are valid within C++. Valid examples are as follows:

```
int x = y;
int x(y);
int x{y};
```

All of the above are valid ways of assigning a variable.

Namespaces

Namespaces will not have an indentation level, nor will anything within the namespace on the first level. This is because it is unnecessary to add a lot of white space and will generally make the code feel too sparse, rather than keeping it all neat and aligned.

Operators

Operators such as +, -, /, *, etc should always have spaces around them. This help to keep the code readable and easy to understand. Brackets do not require spaces. Examples include:

```
x + y = z
a / b = c
f * (t - s) = c
```

Contribution Guide

Code of Conduct

Contributors should be respectful towards their fellow group members and behave appropriately both inside and outside of the working environment.

Positive behaviour includes:

- Giving constructive feedback to other members' contributions and helping them to improve their work.
- Accepting criticism and asking for help when needed.
- Being inclusive and welcoming towards all members of the group.

Unacceptable behaviour includes:

- Use of insulting/derogatory language.
- Targeting and harassing individual members of the group.
- Discrimination based on ethnicity, disability, gender, etc.

The project manager is responsible for maintaining a positive working environment, listening to feedback from their peers and taking action against any members who demonstrate unacceptable behaviour.

Making contributions to the project

This project is coordinated using a GitHub repository on NTU's Olympuss server. Before making a pull from the repository, contributors should consult other team members, namely the lead software developer and/or the project manager for approval.

Commits should be appropriately labelled, describing the changes made and affected files. Files should be stored in the appropriate locations to keep the project organised and easy to access for other members of the group.

Group members should maintain close communication, especially in events such as a merge. Continuous integration helps prevent issues when more than one group member is programming, allowing for testing before a merge is made.

It is the project manager's responsibility to reject and remove any additions or changes they deem detrimental to the project. The project manager may take action against any members who repeatedly and deliberately make such changes.

Reporting bugs and issues

Bugs should be reported in the Issues section of the GitHub repository. Before submitting a bug report, users should make sure they are using the latest build of the project and check the Issues section to ensure their problem has not already been documented. Group members are encouraged to provide as much detail regarding the problem as possible, including their system specification, actions performed prior to the issue, the expected outcome and the actual result.

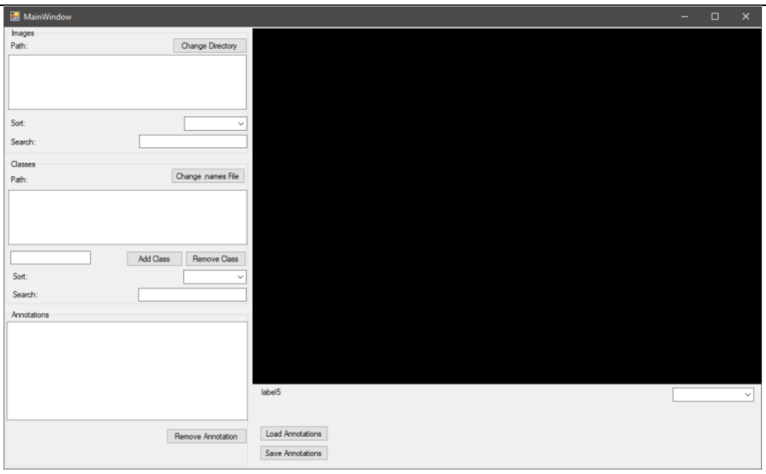
Appendix 2 – Test Report

Testing Conditions

All tests were performed on a 64-bit version of Windows 10 Professional using an 8th generation Intel i5 CPU with 16GB of physical memory (RAM) available. The tests were all run on an x86 version of the software – except for if a System.OutOfMemoryException occurred where the test was then run again on an x64 version of the software to see if the exception repeated. Most tests ran also feature in the final video, shown here: https://youtu.be/382K1KENv-c

Table 6 - Test Number 1

ID:	1a	Description:	The application should be opened to the correct window without any issues
Test Type:	Quality	Success Criteria:	The application opens successfully
Number of Attempts:	3	Comments:	The initial image area is black and looks off putting as well as the name of the window still being a default one. Unfortunately a black terminal window also opens up in the background.
List of Equipment/requirements	No extra requirements for this test, just a compatible OS		
Setup Instructions	The software needs to be built with an executable file available		
Failure Correction Procedure	Attempt to open it again, using administrator mode or equivalent as well. Otherwise confer with the software developer to attempt to fix it.		
Technician	Stephen Anderson		
Individual Results:	Pass – with concerns		

Screenshot	 MainWindow Images Path: Change Directory Sort: Search: Classes Path: Change .names File Add Class Remove Class Sort: Search: Annotations Remove Annotation label5 Load Annotations Save Annotations *Figure 10 Software Starts Successfully*

Test 1a is a manual black box acceptance test to ascertain whether or not the software can successfully open. As can be seen from Figure 10 the user is met with a blank workspace. This test is important as if the software cannot even start successfully then it is entirely useless to the end user.

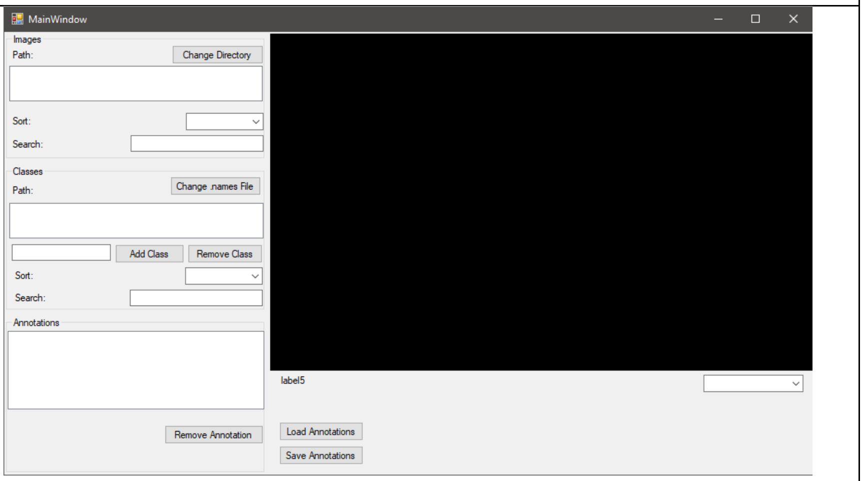
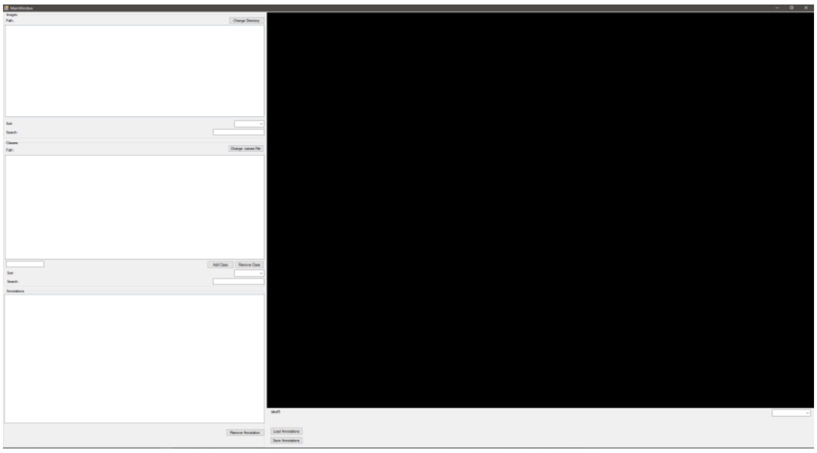
Table 7 - Test Number 2

ID:	2	Description:	The application should remain open for a long period of time without issues
Test Type:	Quality	Success Criteria:	The application does not suffer any problems after 60 minutes of being open but unused
Number of Attempts:	1	Comments:	No comments
List of Equipment/requirements	No extra requirements for this test, just a compatible OS		
Setup Instructions	The software needs to be built with an executable file available		
Failure Correction Procedure	Confer with the software developer to attempt to fix any errors		
Technician	Stephen Anderson		
Individual Results:	Pass		

Test 2 is a manual black box acceptance test that confirms the software does not have any issues when opened and left running for 60 minutes. This test is necessary as it shows that if a user loads up the software and then leaves their computer or is doing other tasks then the software will remain open without issue.

Table 8 - Test Number 3

ID:	3	Description:	The application window should be flexible in its sizing

Test Type:	Quality	Success Criteria:	The application can be resized using the OS built-in resizing tools – including maximise/minimise/border resize tools
Number of Attempts:	5	Comments:	No issues
List of Equipment/requirements	The OS that the software is running on needs to support window resizing and minimizing		
Setup Instructions	The software needs to be running from the executable file		
Failure Correction Procedure	Discuss with the software developer how to use the OS built-in window resizing		
Technician	Stephen Anderson		
Individual Results:	Pass		
Screenshots	 MainWindow Images Path: Change Directory Sort: Search: Classes Path: Change .names File Add Class Remove Class Sort: Search: Annotations Remove Annotation label5 Load Annotations Save Annotations *Figure 11 Window Becomes Smaller*  *Figure 12 Window Becomes Larger*		

Test 3 is a manual black box integration test all about the user being able to resize and manipulate the window itself. This includes using the minimise, maximise and pane resizing that is available via the OS. As can be seen from Figure 11 the user can resize the window to be smaller. Maximising the window works very well with the general layout not being hugely affected; as can be seen by Figure 12. Window manipulation is a very useful tool that is available to the end user as often software is either used in "fullscreen" or needs to be moved/minimized out of the way.

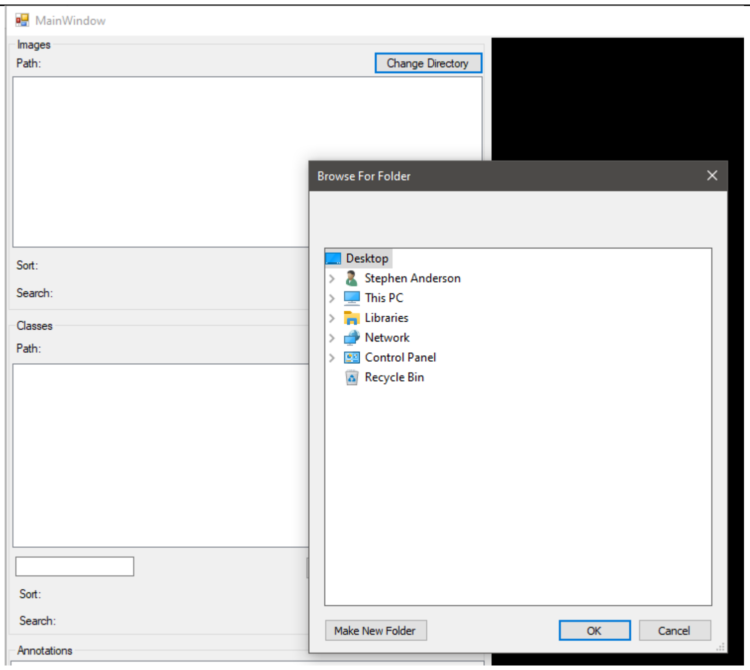
Table 9 - Test Number 4

ID:	4	Description:	The application should close without any issues
Test Type:	Quality	Success Criteria:	The application closes using the OS built-in button
Number of Attempts:	1	Comments:	Works without issues
List of Equipment/requirements	No extra requirements for this test		
Setup Instructions	The software needs to be running from the executable file		
Failure Correction Procedure	Discuss with the software developer potential fixes		
Technician	Stephen Anderson		
Individual Results:	Pass		

Test 4 is a manual black box acceptance test designed to check that the software actually closes on command and doesn't keep running in the background, potentially using up vital resources. This is effectively a very simple test to confirm that the OS built-in close window button functions correctly. The only concerns are what happens when a user has unsaved work as it did not have a popup prompting the user to save any work. If it has "autosave" features, then this is less of a concern but would be nice to have.

Table 10 - Test Number 5a

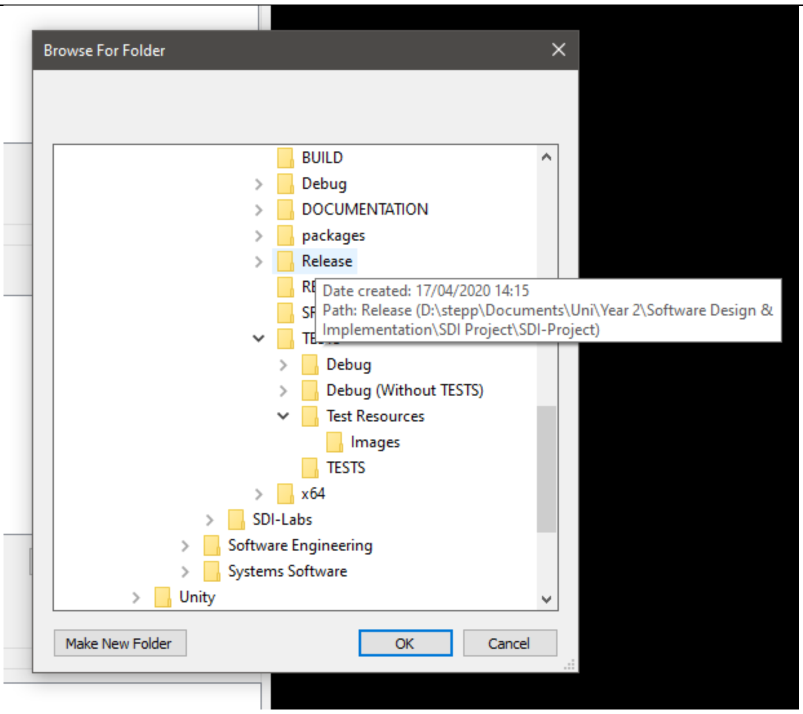
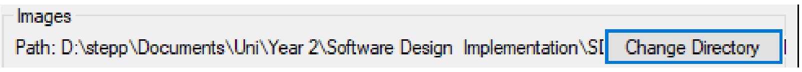
ID:	5a	Description:	The dialog needs to open so that a user can select an image directory
Test Type:	Quality	Success Criteria:	The dialog box successfully opens when the user presses the "Change Directory" button
Number of Attempts:	3	Comments:	Works without issues but maybe would be helpful to have the default directory to be the user's pictures or the currently loaded directory
List of Equipment/requirements	No extra requirements for this test		
Setup Instructions	The application needs to be running from the executable file		

Failure Correction Procedure	Discuss with developer what any issues might be
Technician	Stephen Anderson
Individual Results:	Pass
Screenshot	 MainWindow Images Path: Change Directory Sort: Search: Classes Path: Sort: Search: Annotations Browse For Folder Desktop Stephen Anderson This PC Libraries Network Control Panel Recycle Bin Make New Folder OK Cancel *Figure 13 Opening Directory Dialog Box*

Test 5a is a manual black box test that checks to see if the "Change Directory" button works and opens up the correct dialog for the user to then be able to select the required directory. As can be seen in Figure 13, the dialog has opened that lets the user browse for a folder.

Table 11 - Test Number 5b

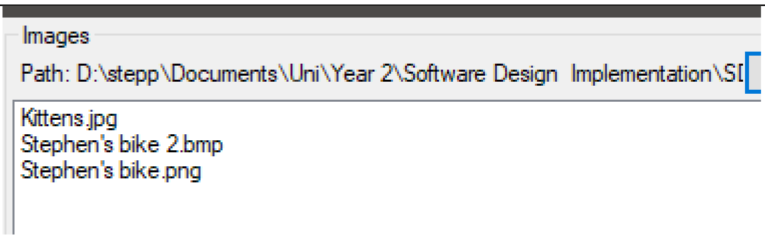
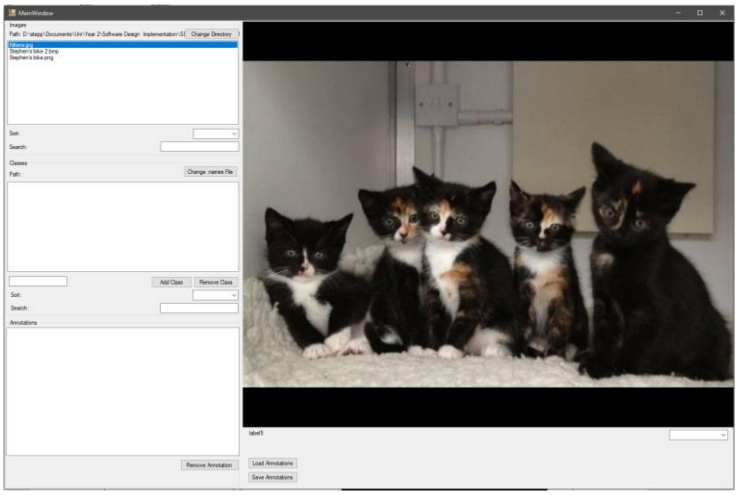
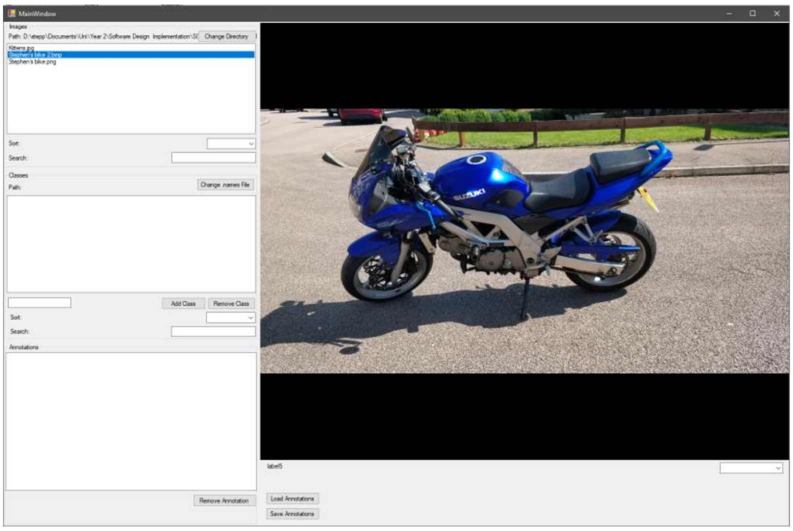
ID:	5b	Description:	The user needs to be able to select the desired directory
Test Type:	Quality	Success Criteria:	The user can select and change the directory from the dialog box
Number of Attempts:	2	Comments:	No Comments
List of Equipment/requirements	There needs to be a valid directory with at least one image file in it		
Setup Instructions	An image directory needs to be selected that contains at least one valid image		
Failure Correction Procedure	Discuss with software developer and look into how to select a specific item from within the list		
Technician	Stephen Anderson		
Individual Results:	Pass		

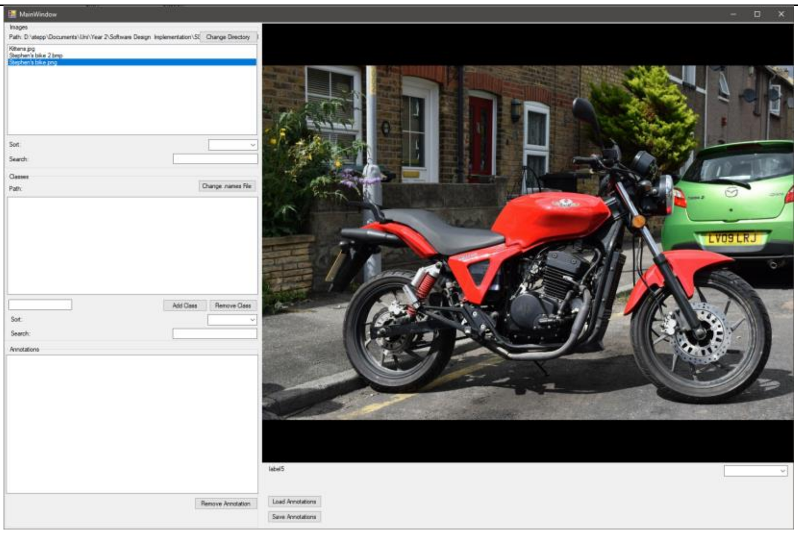
Screenshots	 Browse For Folder BUILD Debug DOCUMENTATION packages Release RE SF TE Debug Debug (Without TESTS) Test Resources Images TESTS x64 SDI-Labs Software Engineering Systems Software Unity Make New Folder OK Cancel Date created: 17/04/2020 14:15 Path: Release (D:\stepp\Documents\Uni\Year 2\Software Design & Implementation\SDI Project\SDI-Project) *Figure 14 Selecting the Directory*  Images Path: D:\stepp\Documents\Uni\Year 2\Software Design Implementation\SI *Figure 15 The Directory has Changed*

Test 5b is a manual black box integration test to confirm that the user can open the dialog box, select and then load the desired directory. Figure 14 and Figure 15 demonstrate that this is the case, first by selecting the folder from the dialog box and then updating the displayed path on the software window to reflect those changes.

Table 12 - Test Number 5c

ID:	5c	Description:	The user needs an image file to be correctly loaded
Test Type:	Quantity	Success Criteria:	A file with each file type for an image that is supported are can be loaded and displayed (.PNG, .JPG, .BMP)
Number of Attempts:	3 (3 images)	Comments:	No comments
List of Equipment/requirements	A number of image files with varying correct file types are needed		
Setup Instructions	A directory with the required images needs to be selected		
Failure Correction Procedure	Discuss with software developer how the file types are different and how they’re stored		
Technician	Stephen Anderson		

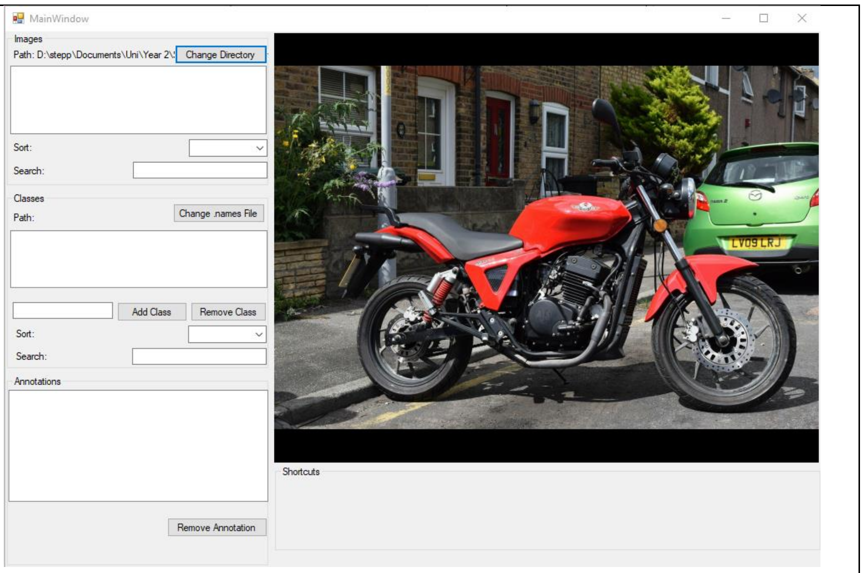
Individual Results:	Pass
Screenshot	 Images Path: D:\stepp\Documents\Uni\Year 2\Software Design Implementation\S[Kittens.jpg Stephen's bike 2.bmp Stephen's bike.png *Figure 16 Images Load Into the List*  MainWindow Images Path: D:\stepp\Documents\Uni\Year 2\Software Design Implementation\S[Change Directory Kittens.jpg Stephen's bike 2.bmp Stephen's bike.png Sort: Search: Classes Path: Change .names File Add Class Remove Class Sort: Search: Annotations Remove Annotation label5 Load Annotations Save Annotations *Figure 17 Kittens (.JPG)*  MainWindow Images Path: D:\stepp\Documents\Uni\Year 2\Software Design Implementation\S[Change Directory Kittens.jpg Stephen's bike 2.bmp Stephen's bike.png Sort: Search: Classes Path: Change .names File Add Class Remove Class Sort: Search: Annotations Remove Annotation label5 Load Annotations Save Annotations *Figure 18 SV650S (.BMP)*

	 MainWindow Images Path: D:\stepp\Documents\Uni\Year 2\Software Design Implementation\S Change Directory Kittens.jpg Stephen's bike 2.bmp Stephen's bike.png Sort: Search: Classes Path: Change .names File Add Class Remove Class Sort: Search: Annotations Remove Annotation label5 Load Annotations Save Annotations *Figure 19 NAC12 (.PNG)*

Test 5c is a black box acceptance test that checks that the software loads all available images from the selected directory and that the user can then select and load those images. The only required image types were JPEG/JPG, PNG and BMP so those are the three types checked in this test. It is important that these three image types as a minimum are loaded as without them the user cannot use the software to its full effect. As can be seen from Figure 17, Figure 18 and Figure 19 these pictures are loaded and can be displayed correctly.

Table 13 - Test Number 5d

ID:	5d	Description:	The software can handle other types of images
Test Type:	Quantity	Success Criteria:	Other types of image file should be displayed or handled (not causing exceptions)
Number of Attempts:	2 (3 images)	Comments:	Doesn't display the images. Maybe tell the user that there are images that couldn't be loaded?
List of Equipment/requirements	The images contained within Images. The types included are .GIF, .TIFF and .WEBM (can be obtained from https://file-examples.com/index.php/sample-images-download/)		
Setup Instructions	A directory with the required images needs to be selected		
Failure Correction Procedure	Look at how the program handles different file types and attempting to fix it so that they can be displayed properly		
Technician	Stephen Anderson		
Individual Results:	Pass		

Screenshots	 MainWindow Images Change Directory Sort: Search: Classes Path: Change .names File Add Class Remove Class Sort: Search: Annotations Remove Annotation Shortcuts Path: D:\stepp\Documents\Uni\Year 2\ *Figure 20 Invalid Images*

Test No. 5d shows what happens when loading slightly less common image file types: GIF, TIFF and WEBP. These images were taken from https://file-examples.com/index.php/sample-images-download/ as they offer a few different types of images with convenient downloads. The images were not loaded, however the previous image stayed on the screen. It may also be worth having a popup telling the user that there were some unsupported file types inside their chosen directory.

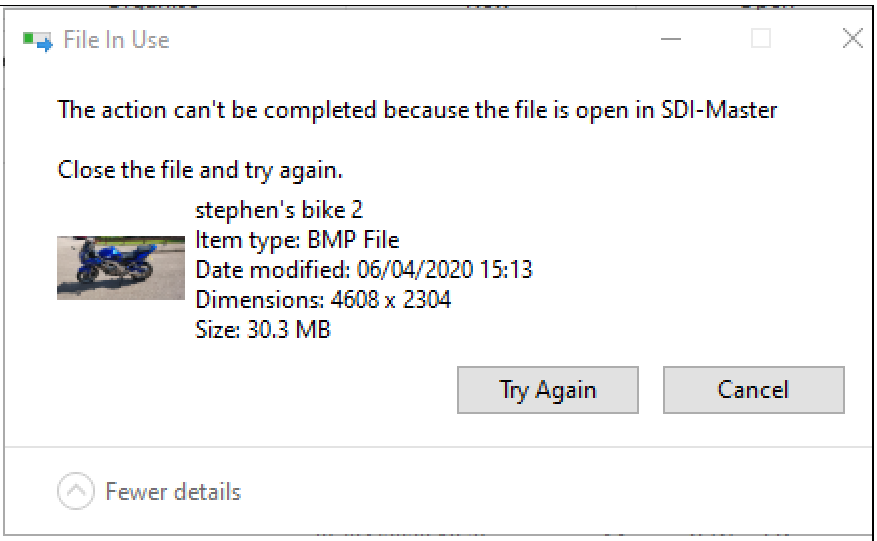
Table 14 - Test Number 5e

ID:	5e	Description:	Adding an image to the currently selected directory adds it to the opened images
Test Type:	Quantity	Success Criteria:	The software should automatically open any newly added images to the directory that the user has selected
Number of Attempts:	1	Comments:	Doesn’t update automatically. Maybe make it a separate update thread?
List of Equipment/requirements	A valid directory and a valid image not currently in it		
Setup Instructions	Open the directory using the software and a file browser, then add the new file to the required directory		
Failure Correction Procedure	Discuss with the software developer the best course of action for refreshing the list of images without making it too resource intensive		
Engineer/Technician	Samuel Harrison/Stephen Anderson		
Individual Results:	Fail		

Test 5e is a manual black box test made to see what happens when a user adds an image to the working directory while it is open in the software. It should automatically refresh and display the image in the list however it currently does not. A form of multithreading can be used to fix this by having a form of refresh on a timer to see if there are any changes to the directory and if so then it can load the new images into the list. This is useful as it would allow the user to add new files to the directory and the software automatically realise this without the user needing to manually reselect the working directory.

Table 15 - Test Number 5f

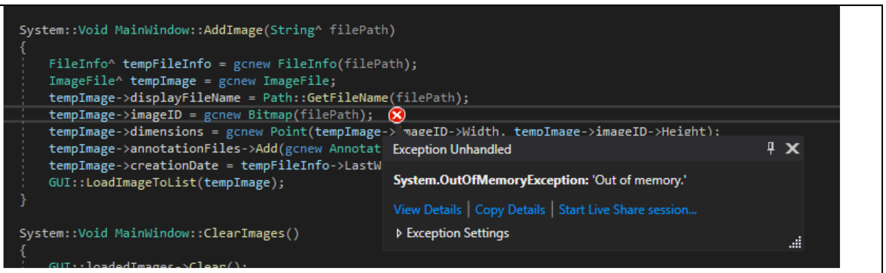
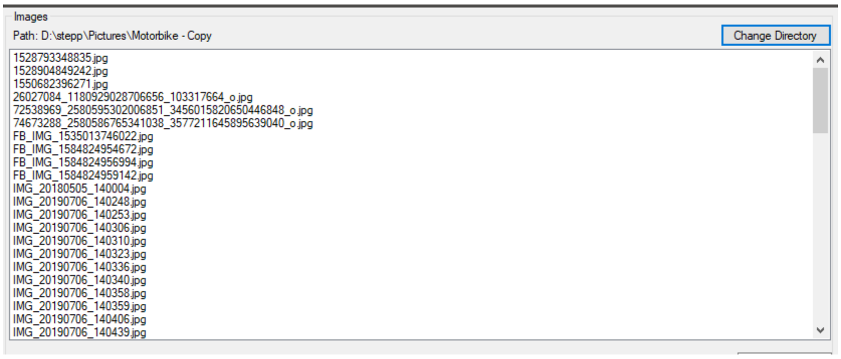
ID:	5f	Description:	It shouldn't be possible to remove an image from the working directory to help prevent issues with inconsistency due to some changes being made but not saved
Test Type:	Quantity	Success Criteria:	The software should prevent the user from moving/removing a file from the working directory
Number of Attempts:	1	Comments:	No comments
List of Equipment/requirements	A valid directory and a valid image currently in it		
Setup Instructions	Open the directory using the software and a file browser, then move the new file to a different directory		
Failure Correction Procedure	Discuss with the software developer the best course of action for locking the user from being able to move or delete a file when it is being used by the software		
Engineer/Technician	Samuel Harrison/Stephen Anderson		
Individual Results:	Pass		

Screenshot	 File In Use The action can't be completed because the file is open in SDI-Master Close the file and try again. stephen's bike 2 Item type: BMP File Date modified: 06/04/2020 15:13 Dimensions: 4608 x 2304 Size: 30.3 MB *Figure 21 Image in Use*

Test 5f a manual white box acceptance test; there are no inputs in this test and as the file is open in the software then the OS should not be able to change it even if a user attempts to remove an image from the directory. This is to help prevent issues that could occur when annotating it. Figure 21 demonstrates that this is the case as it would not move the image to a different folder.

Table 16 - Test Number 5g

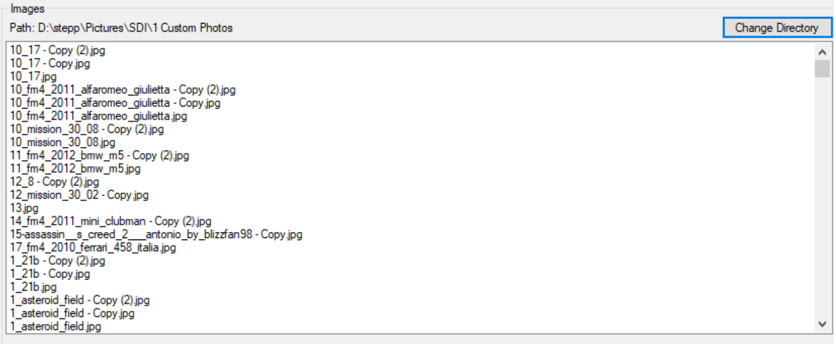
ID:	5g	Description:	Can open a directory with lots of images
Test Type:	Quantity	Success Criteria:	Can open a directory that contains 100 images
Number of Attempts:	2	Comments:	Runs out of memory when run as a 32-bit version.
List of Equipment/requirements	A directory that contains 100 images in it (the zipped folder can be obtained here: https://drive.google.com/open?id=1SdonAdFGIFg_wxaT7T2K3UzV1dGYDKpE)		
Setup Instructions	Find/make a directory that has 100 images in it then load that directory and its images using the required button in the software		
Failure Correction Procedure	Discuss with the software developer the best way to increase the allocated amount of memory available for the 32-bit version of the software.		
Engineer/Technician	Samuel Harrison/Stephen Anderson		
Individual Results:	Pass		

Screenshot	
	`System::Void MainWindow::AddImage(String^ filePath)`
	`{`
	`    FileInfo^ tempFileInfo = gcnew FileInfo(filePath);`
	`    ImageFile^ tempImage = gcnew ImageFile;`
	`    tempImage->displayFileName = Path::GetFileName(filePath);`
	`    tempImage->imageID = gcnew Bitmap(filePath);`
	`    tempImage->dimensions = gcnew Point(tempImage-> mageID->Width, tempImage->imageID->Height);`
	`    tempImage->annotationFiles->Add(gcnew Annotat`
	`    tempImage->creationDate = tempFileInfo->LastW`
	`    GUI::LoadImageToList(tempImage);`
	`}`
	
	`System::Void MainWindow::ClearImages()`
	`{`
	`    GUI::loadedImages->Clear();`
	Exception Unhandled
	System.OutOfMemoryException: 'Out of memory.'
	 View Details | Copy Details | Start Live Share session...
	Exception Settings
	Figure 22 Out of Memory Exception
	
	Images
	Path: D:\stepp\Pictures\Motorbike - Copy
	
	1528793348835.jpg
	1528904849242.jpg
	1550682396271.jpg
	26027084_1180929028706656_103317664_o.jpg
	72538969_2580595302006851_3456015820650446848_o.jpg
	74673288_2580586765341038_3577211645895639040_o.jpg
	FB_IMG_1535013746022.jpg
	FB_IMG_1584824954672.jpg
	FB_IMG_1584824956994.jpg
	FB_IMG_1584824959142.jpg
	IMG_20180505_140004.jpg
	IMG_20190706_140248.jpg
	IMG_20190706_140253.jpg
	IMG_20190706_140306.jpg
	IMG_20190706_140310.jpg
	IMG_20190706_140323.jpg
	IMG_20190706_140336.jpg
	IMG_20190706_140340.jpg
	IMG_20190706_140358.jpg
	IMG_20190706_140359.jpg
	IMG_20190706_140406.jpg
	IMG_20190706_140439.jpg
	Figure 23 100 Images Loaded

Test 5g is a black box acceptance test related to loading the images. This test is designed to show what happens if a user tries to use a working directory with more than just a few images. For this, a folder with 100 images was used. Unfortunately, when running the 32-bit version of the software it gets a “System.OutOfMemoryException” which is likely to be due to the limited memory available in that configuration. When running the 64-bit version it suffers no issues except for taking a while to load the images.

Table 17 - Test Number 5h

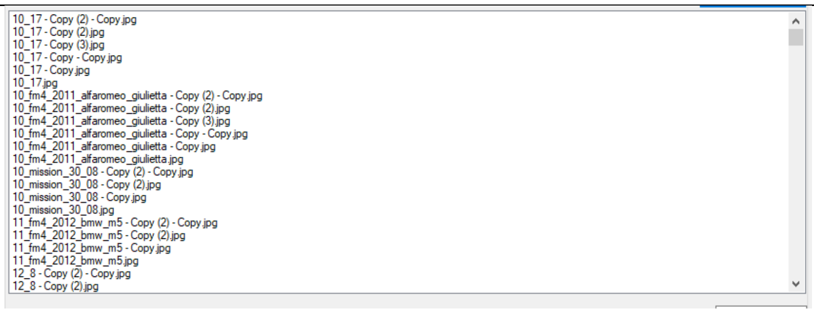
ID:	5h	Description:	Can open a directory with lots of images
Test Type:	Quantity	Success Criteria:	Can open a directory that contains 1000 images
Number of Attempts:	1	Comments:	The automated garbage collector as part of CLI is run, frequently. Even with that the program started using 8.6GB of RAM. There appears to be a memory leak somewhere as the folder only uses up 0.99GB of disk storage.
List of Equipment/requirements	A directory that contains 1000 images in it		
Setup Instructions	Find/make a directory that has 1000 images in it then load that directory and its images using the required button in the software		

Failure Correction Procedure	Discuss with the software developer the best way to increase the allocated amount of memory available.
Engineer/Technician	Samuel Harrison/Stephen Anderson
Individual Results:	Pass
Screenshot	 Path: D:\stepp\Pictures\SDI\1 Custom Photos 10_17 - Copy (2).jpg 10_17 - Copy.jpg 10_17.jpg 10_fm4_2011_alfaromeo_giulietta - Copy (2).jpg 10_fm4_2011_alfaromeo_giulietta - Copy.jpg 10_fm4_2011_alfaromeo_giulietta.jpg 10_mission_30_08 - Copy (2).jpg 10_mission_30_08.jpg 11_fm4_2012_bmw_m5 - Copy (2).jpg 11_fm4_2012_bmw_m5.jpg 12_8 - Copy (2).jpg 12_mission_30_02 - Copy.jpg 13.jpg 14_fm4_2011_mini_clubman - Copy (2).jpg 15-assassin__s_creed_2___antonio_by_blizzfan98 - Copy.jpg 17_fm4_2010_ferrari_458_italia.jpg 1_21b - Copy (2).jpg 1_21b - Copy.jpg 1_21b.jpg 1_asteroid_field - Copy (2).jpg 1_asteroid_field - Copy.jpg 1_asteroid_field.jpg *Figure 24 1000 Images Loaded*

Test 5h is an acceptance test to see if there is a limit to how many images the software can load at once. The limit was not found on this test, although it did highlight that there is a likely memory leak as the software started to use a significant amount of RAM – 9x that of the size of the directory on the storage disk.

Table 18 - Test Number 5i

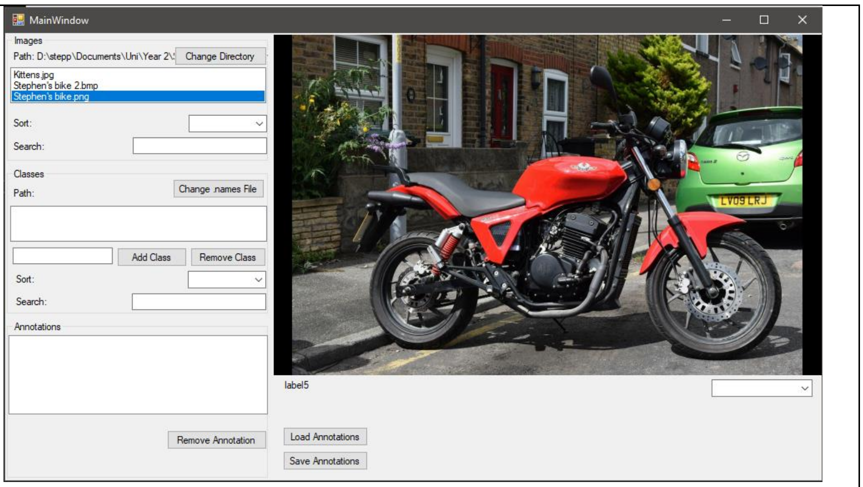
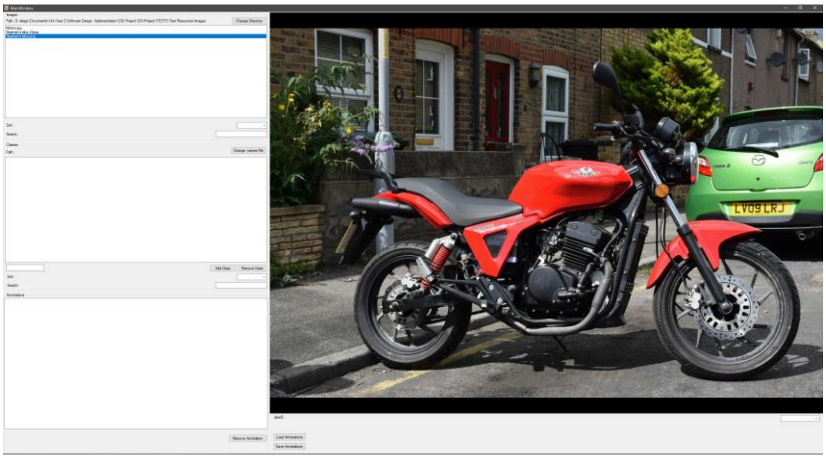
ID:	5i	Description:	Can open a directory with lots of images
Test Type:	Quantity	Success Criteria:	Can open a directory that contains 2000 images
Number of Attempts:	1	Comments:	The automated garbage collector as part of CLI is run, frequently. Even with that the program started using 17.1GB of RAM – more than is available as physical hardware.
List of Equipment/requirements	A directory that contains 2000 images in it		
Setup Instructions	Find/make a directory that has 2000 images in it then load that directory and its images using the required button in the software		
Failure Correction Procedure	Discuss with the software developer the best way to increase the allocated amount of memory available.		
Engineer/Technician	Samuel Harrison/Stephen Anderson		
Individual Results:	Pass		

Screenshot	 10_17 - Copy (2) - Copy.jpg 10_17 - Copy (2).jpg 10_17 - Copy (3).jpg 10_17 - Copy - Copy.jpg 10_17 - Copy.jpg 10_17.jpg 10_fm4_2011_alfaromeo_giulietta - Copy (2) - Copy.jpg 10_fm4_2011_alfaromeo_giulietta - Copy (2).jpg 10_fm4_2011_alfaromeo_giulietta - Copy (3).jpg 10_fm4_2011_alfaromeo_giulietta - Copy - Copy.jpg 10_fm4_2011_alfaromeo_giulietta - Copy.jpg 10_fm4_2011_alfaromeo_giulietta.jpg 10_mission_30_08 - Copy (2) - Copy.jpg 10_mission_30_08 - Copy (2).jpg 10_mission_30_08 - Copy.jpg 10_mission_30_08.jpg 11_fm4_2012_bmw_m5 - Copy (2) - Copy.jpg 11_fm4_2012_bmw_m5 - Copy (2).jpg 11_fm4_2012_bmw_m5 - Copy.jpg 11_fm4_2012_bmw_m5.jpg 12_8 - Copy (2) - Copy.jpg 12_8 - Copy (2).jpg *Figure 25 2000 Images Loaded*

Test 5i further highlights that there is likely a memory leak of some kind; the Visual Studio Debugger started reporting that 17.1GB of memory was in use. That is more than is available in the test environment as physical RAM. It should not be using that amount to load less than 2GB of images. Unfortunately, the limit was not found with this test but it does demonstrate that there likely is no limit as long as there is memory – physical or virtual - available. This indicates that the software is not the limit when storing images, but rather the amount of system memory available.

Table 19 - Test Number 6

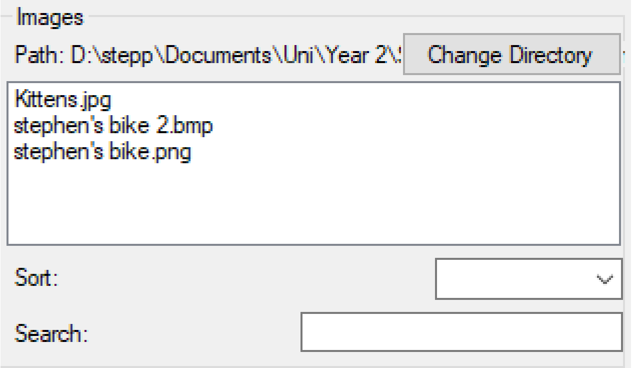
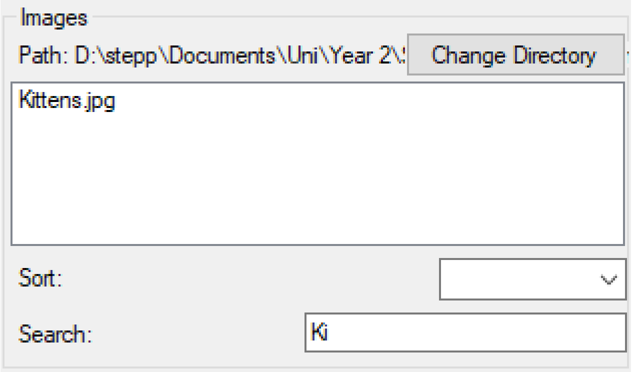
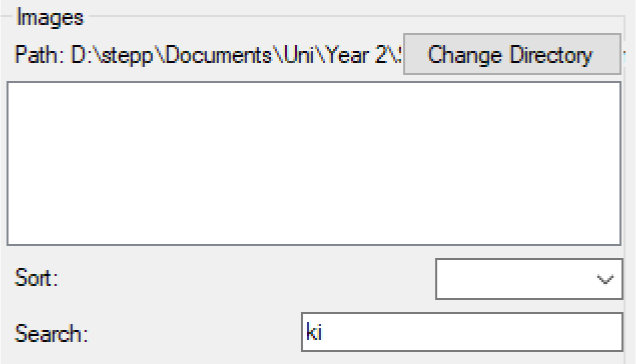
ID:	6	Description:	The user should be able to resize the window and still have the whole image displayed
Test Type:	Quality	Success Criteria:	When the window is resized, the loaded image needs to stay wholly in view in the centre of the drawing area
Number of Attempts:	2	Comments:	Works with minimal issues, however resizing the window can feel quite unresponsive
List of Equipment/requirements	A valid image with a supported file type is required		
Setup Instructions	An image needs to be loaded into the drawing area		
Failure Correction Procedure	Look into which way the image needs to be loaded to improve responsiveness and flexibility		
Technician	Stephen Anderson		
Individual Results:	Pass		

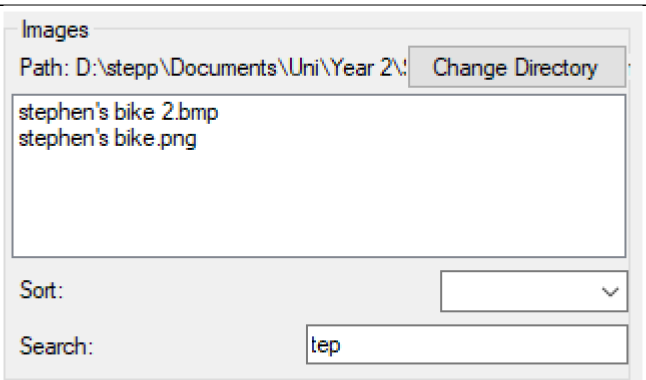
Screenshot	 MainWindow Images Change Directory Sort: Search: Classes Path: Change .names File Add Class Remove Class Sort: Search: Annotations Remove Annotation label5 Load Annotations Save Annotations Path: D:\stepp\Documents\Uni\Year 2\: Kittens.jpg Stephen's bike 2.bmp Stephen's bike.png *Figure 26 NAC 12 Small*  *Figure 27 NAC 12 Large*

Test 6 is a manual black box integration test that demonstrates that the window can be resized without detriment to the image – it remains fully visible. The test passes without any issues related to the image as can be seen from Figure 26 and Figure 27.

Table 20 - Test Number 7

ID:	7	Description:	The user should be able to search for a single image based on the file name from the list of loaded images
Test Type:	Quality	Success Criteria:	The search results in an image with a matching name to the search and any partial matches if appropriate
Number of Attempts:	2	Comments:	Works but is case sensitive
List of Equipment/requirements	A few valid images with supported types		

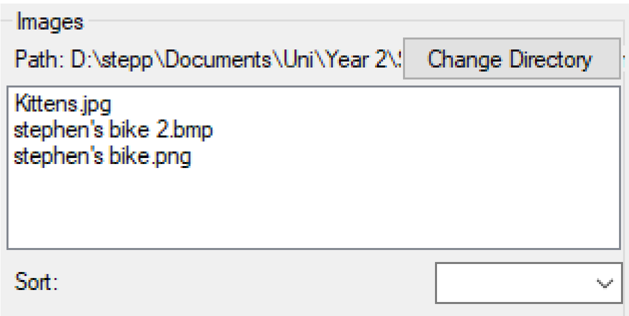
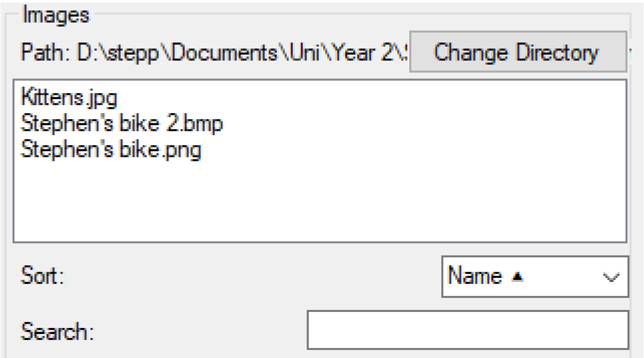
Setup Instructions	Multiple images with unique/partially unique names should be loaded into the list of images
Failure Correction Procedure	Discuss with the software developer and potentially group manager the best filtering/searching algorithms and the best way to implement it
Engineer/Technician	Samuel Harrison/Stephen Anderson
Individual Results:	Pass – with an area for concern
Screenshots	 Images Path: D:\stepp\Documents\Uni\Year 2\ Change Directory Kittens.jpg stephen's bike 2.bmp stephen's bike.png Sort: Search: *Figure 28 Unsearched Images*  Images Path: D:\stepp\Documents\Uni\Year 2\ Change Directory Kittens.jpg Sort: Search: Ki *Figure 29 Kitten Search*  Images Path: D:\stepp\Documents\Uni\Year 2\ Change Directory Sort: Search: ki *Figure 30 Case Sensitive Kitten Search*

	 Images Path: D:\stepp\Documents\Uni\Year 2\ Change Directory stephen's bike 2.bmp stephen's bike.png Sort: Search: tep *Figure 31 Multiple Image Matches*

Test 7 is a black box integration test that highlights any potential issues when the user tries to search for an image's name. The function performs well when done manually (unit tests are elsewhere in this document) however it matches exact cases – it is case sensitive. Functionality can be easily seen in Figure 28, Figure 29, Figure 30 and Figure 31.

Table 21 - Test Number 8a

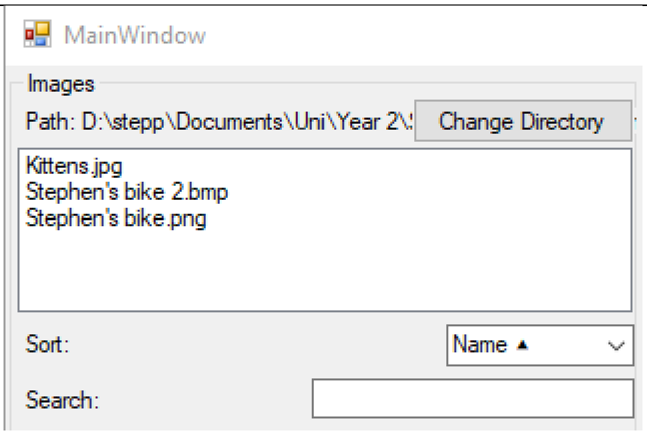
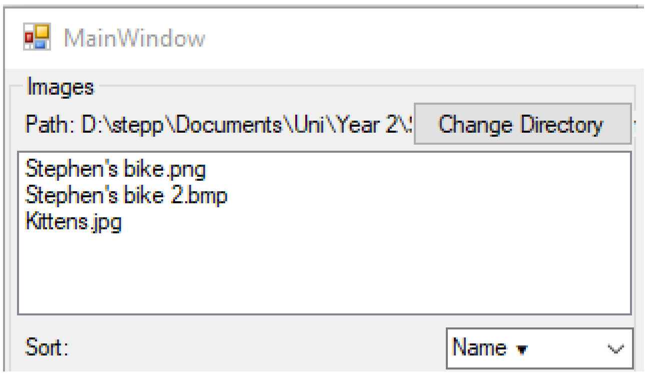
ID:	8a	Description:	The user should be able to sort the loaded images by ascending name
Test Type:	Quality	Success Criteria:	The images should move their entries in the list based on the sorting method
Number of Attempts:	2	Comments:	The images load in automatically sorted already but the function does also appear to work based on other features and tests
List of Equipment/requirements	A directory with at least two differently named valid images		
Setup Instructions	Load the directory that contains the required images		
Failure Correction Procedure	Discuss the best sorting algorithms to implement		
Engineer/Technician	Samuel Harrison/Stephen Anderson		
Individual Results:	Pass		

Screenshots	 Path: D:\stepp\Documents\Uni\Year 2\ Kittens.jpg stephen's bike 2.bmp stephen's bike.png Sort: *Figure 32 Unsorted Image List 1*  Path: D:\stepp\Documents\Uni\Year 2\ Kittens.jpg Stephen's bike 2.bmp Stephen's bike.png Sort: Name ▲ Search: *Figure 33 Image List After Sorting A-Z*

Test 8a is a black box acceptance test that shows if the sorting methods available for the images work correctly. This is a fairly important test as it is likely a user will want to re-order the images that are loaded to assist them in using the software if there are lots of images loaded. As can be seen in Figure 33 the sorting method has changed to reflect that it is now in ascending alphabetical order.

Table 22 - Test Number 8b

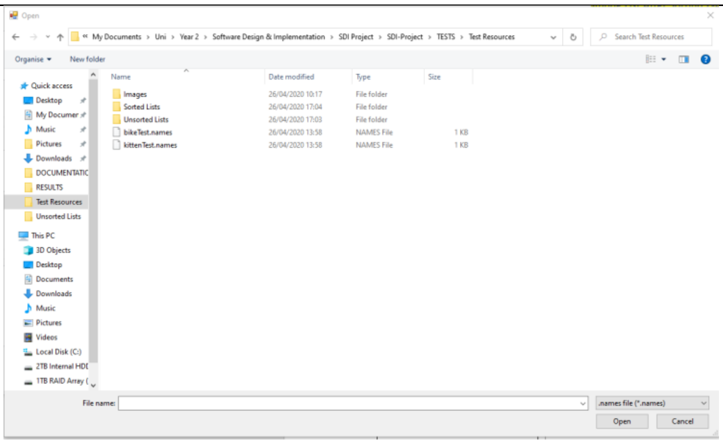
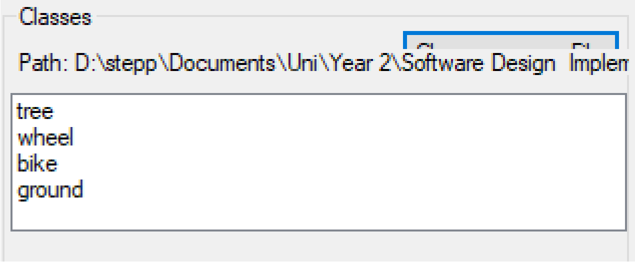
ID:	8b	Description:	The user should be able to sort the loaded images by descending name
Test Type:	Quality	Success Criteria:	The images should move their entries in the list based on the sorting method
Number of Attempts:	2	Comments:	Works but it may be worth looking at ignoring spaces as the two similarly named images are the other way around to what would be logical
List of Equipment/requirements	A directory with at least two differently named valid images		
Setup Instructions	Load the directory that contains the required images		
Failure Correction Procedure	Discuss the best sorting algorithms to implement		
Engineer/Technician	Samuel Harrison/Stephen Anderson		

Individual Results:	Pass
Screenshots	 MainWindow Images Path: D:\stepp\Documents\Uni\Year 2\ Kittens.jpg Stephen's bike 2.bmp Stephen's bike.png Sort: Name ▲ Search: *Figure 34 Unsorted Image List 2*  MainWindow Images Path: D:\stepp\Documents\Uni\Year 2\ Stephen's bike.png Stephen's bike 2.bmp Kittens.jpg Sort: Name ▼ *Figure 35 Image List After Sorting Z-A*

Test 8b is a black box acceptance test that shows if the sorting methods available for the images work correctly. This is a fairly important test as it is likely a user will want to re-order the images that are loaded to assist them in using the software if there are lots of images loaded. As can be seen in Figure 35 the sorting method has affected the list and changed the sorting method displayed to reflect that it is now in descending alphabetical order.

Table 23 - Test Number 9

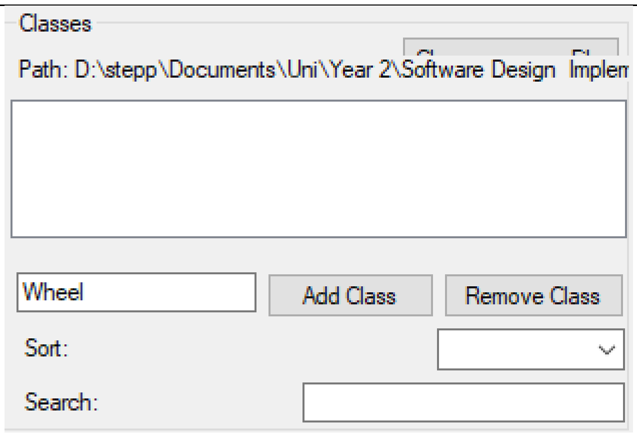
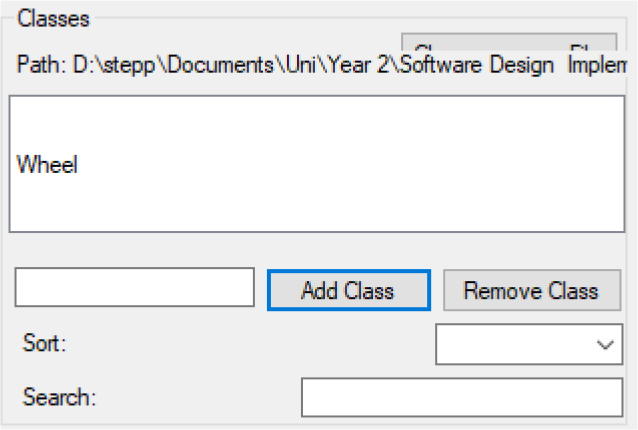
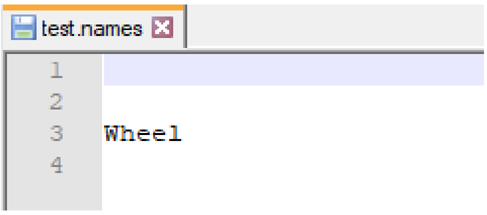
ID:	9	Description:	Add/change the names file
Test Type:	Quality	Success Criteria:	Allows the user to select the required file from the dialog box
Number of Attempts:	2	Comments:	The path shown covers up the change file button
List of Equipment/requirements	A valid .names file (can be empty)		
Setup Instructions	Use the "Change .names File" button to open the dialog box and then select the required file		
Failure Correction Procedure	Discuss with the software developer how to prevent the path from covering the button		

Engineer/Technician	Samuel Harrison/Stephen Anderson
Individual Results:	Pass – with concerns
Screenshot	 Open « My Documents › Uni › Year 2 › Software Design & Implementation › SDI Project › SDI-Project › TESTS › Test Resources Search Test Resources Organise ▾ New folder Quick access Desktop My Documer Music Pictures Downloads DOCUMENTATIC RESULTS Test Resources Unsorted Lists This PC 3D Objects Desktop Documents Downloads Music Pictures Videos Local Disk (C:) 2TB Internal HDI 1TB RAID Array (Open Cancel Name | Date modified | Type | Size Images | 26/04/2020 10:17 | File folder | Sorted Lists | 26/04/2020 17:04 | File folder | Unsorted Lists | 26/04/2020 17:03 | File folder | bikeTest.names | 26/04/2020 13:58 | NAMES File | 1 KB kittenTest.names | 26/04/2020 13:58 | NAMES File | 1 KB File name: .names file (*.names) *Figure 36 Selecting the .names File*  Path: D:\stepp\Documents\Uni\Year 2\Software Design Implem tree wheel bike ground *Figure 37 .names File Path*

Test 9 is a black box integration test to demonstrate that the user can open the dialog box designed for opening a specific file, can then be filtered to the correct file type and add the desired file to the software. This works exceptionally well with the filter actually being restricted to just .names files which helps to prevent invalid files from loading.

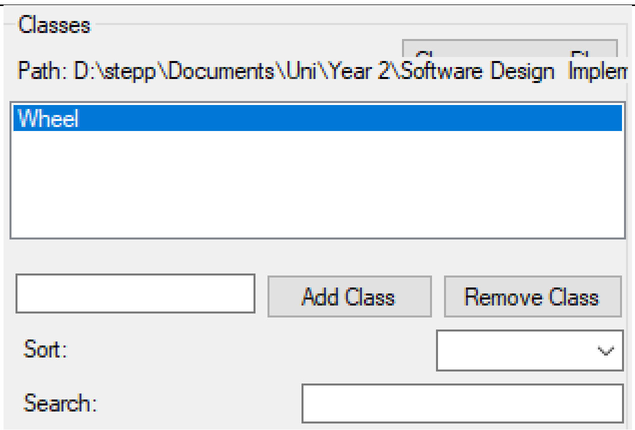
Table 24 - Test Number 10a

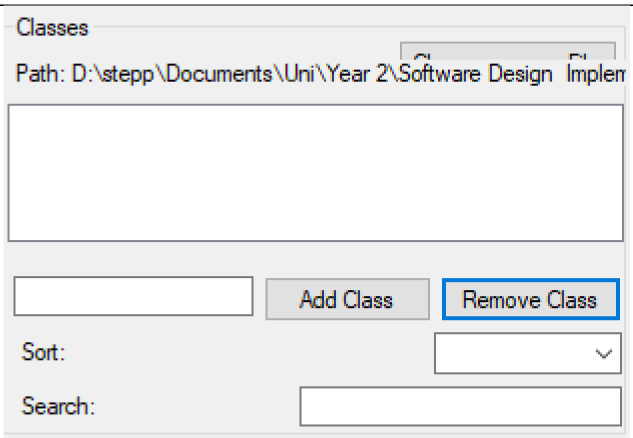
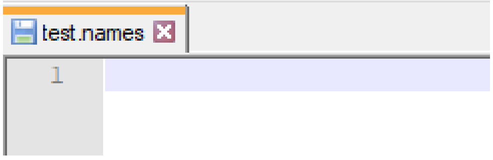
ID:	10a	Description:	Can add a new class to the .names file
Test Type:	Quantity	Success Criteria:	Considered a success if the typed in class is added to the .names list and file
Number of Attempts:	1	Comments:	Works fine but the .names file has extra whitespace added if I click “Add Class” without typing anything so may be worth adding in a form of validation for that. It also implies that there can be multiple classes with the same name

List of Equipment/requirements	Have a valid empty .names file
Setup Instructions	Make sure that the .names file is loaded before typing into the text box next to the “Add Class” button
Failure Correction Procedure	Discuss with the software developer how to output to the required file
Engineer/Technician	Samuel Harrison/Stephen Anderson
Individual Results:	Pass
Screenshots	 Path: D:\stepp\Documents\Uni\Year 2\Software Design Implem Wheel *Figure 38 New Class Name*  Path: D:\stepp\Documents\Uni\Year 2\Software Design Implem Wheel *Figure 39 New Class Made*  Wheel *Figure 40 New Class Made (In the File)*

Test 10a is a black box integration test to see if a user can add a new class type to their loaded .names file which will then allow them to label annotation types later. Currently there appears to be no form of validation to the class names as during testing, three blank classes were made; as can be seen most clearly in Figure 40.

Table 25 - Test Number 10b

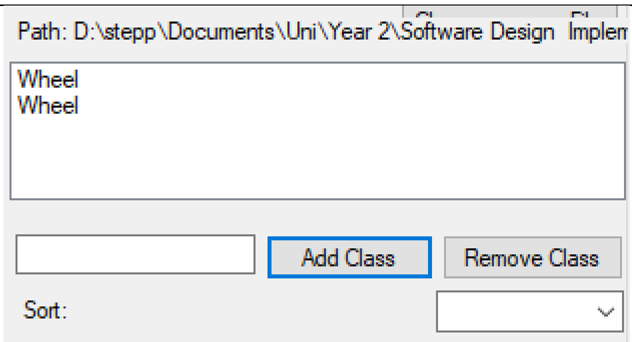
ID:	10b	Description:	Can successfully remove a class from the .names file
Test Type:	Quantity	Success Criteria:	Considered a success if the selected class is removed from the displayed list and the file itself
Number of Attempts:	1	Comments:	Appears to be functioning without issue
List of Equipment/requirements	A valid .names file with at least one named class		
Setup Instructions	Load the required .names file		
Failure Correction Procedure	Have a look at how the classes are loaded and stored as well as how to alter the file		
Engineer/Technician	Samuel Harrison/Stephen Anderson		
Individual Results:	Pass		
Screenshots	 Path: D:\stepp\Documents\Uni\Year 2\Software Design Implem Wheel *Figure 41 Existing Class*		

	 Classes Path: D:\stepp\Documents\Uni\Year 2\Software Design Implem Add Class Remove Class Sort: Search: *Figure 42 Class Removed From List*  *Figure 43 Class Removed From File*

Test 10b is another black box integration test to determine whether or not the user can remove classes that have been created and added to the .names file. As can be seen from Figure 41, Figure 42 and Figure 43 this is in fact the case and the file itself updates as well.

Table 26 - Test Number 10c

ID:	10c	Description:	Should not be able to add a class with the name of an existing class
Test Type:	Quality	Success Criteria:	The software does not let two classes exist with the same name in the list
Number of Attempts:	1	Comments:	No form of validation on class names
List of Equipment/requirements	A valid .names file with an existing class in it		
Setup Instructions	Load the required .names file		
Failure Correction Procedure	Look at how the classes are added and look at putting in a form of validation that prevents invalid class names		
Engineer/Technician	Samuel Harrison/Stephen Anderson		
Individual Results:	Fail		

Screenshots	 Path: D:\stepp\Documents\Uni\Year 2\Software Design Implem Wheel Wheel *Figure 44 Multiple Classes With the Same Name*

Test 10c is a black box unit test that confirms there is no validation for creating class names within the same file. As can be seen from Figure 44 there are two classes with exactly the same name. If there was a longer list of unsorted class names, then it would be possible for a user to accidentally create two classes which could then cause issues if they try to use them at the same time.

Table 27 - Test Number 10d

ID:	10d	Description:	Changing the .names file directly should reload the list in the software
Test Type:	Quality	Success Criteria:	Any changes made directly to the file update the loaded list
Number of Attempts:	1	Comments:	Does not update the loaded list if the file is changed directly
List of Equipment/requirements	A loaded .names file		
Setup Instructions	Load the .names file in the software and a text editor		
Failure Correction Procedure	Look at making a thread that checks for updates to the file		
Engineer/Technician	Samuel Harrison/Stephen Anderson		
Individual Results:	Fail		

Test 10d is a black box acceptance test to confirm that there is no form of refresh available for the class names list when the file on disk is changed. This would be a nice feature to have as it would allow a user to create a lot of class names in a slightly more efficient way as they could use a text editor of their choice.

Table 28 - Test Number 10e

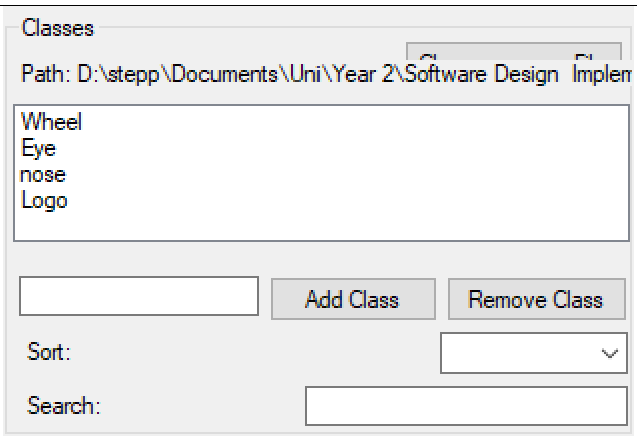
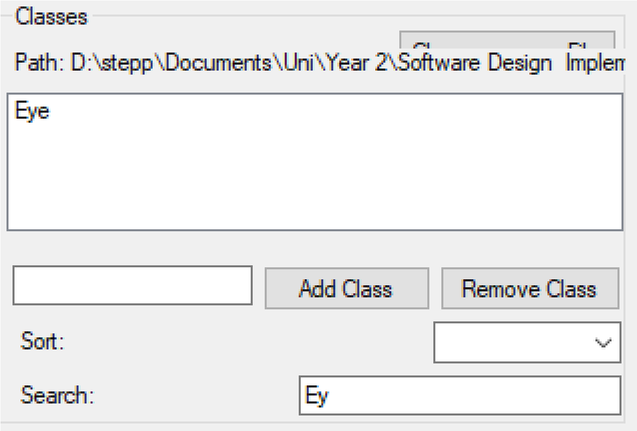
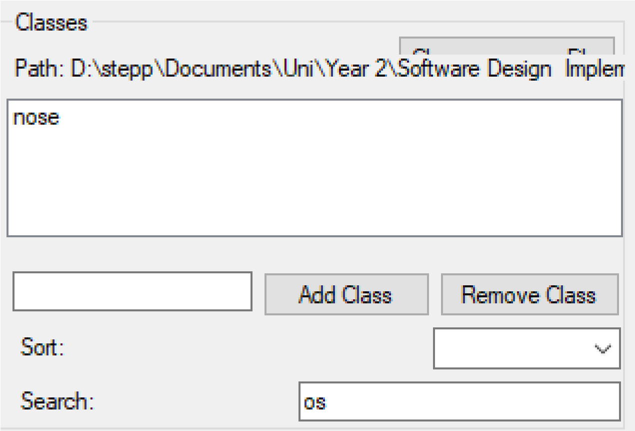
ID:	10e	Description:	Rename class
Test Type:	Quality	Success Criteria:	A way of selecting and then choosing to rename a class leads to the class having a different name

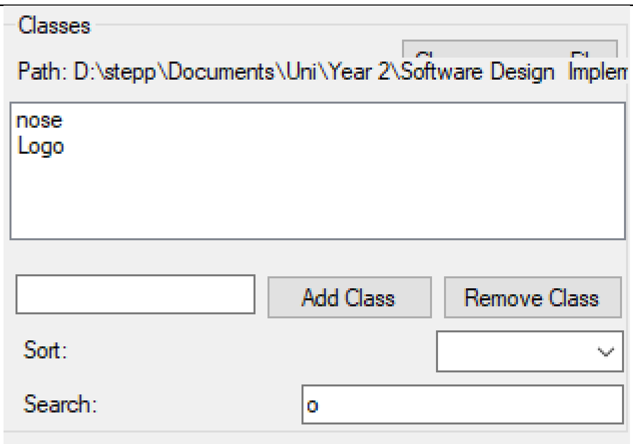
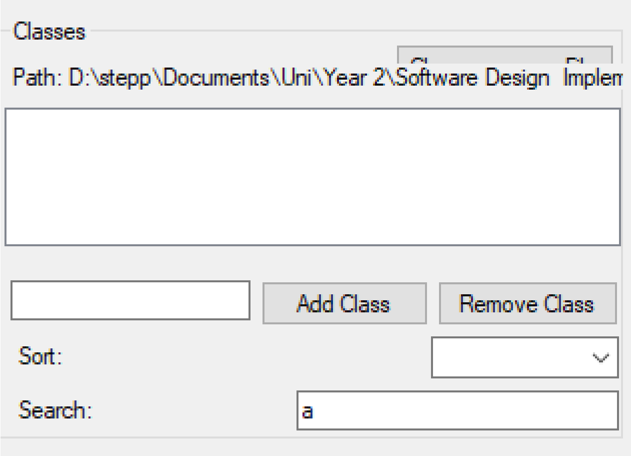
Number of Attempts:	1	Comments:	No apparent way – either through software button or hardware input – to start the renaming process. The only apparent way is to delete the existing class and create a new one
List of Equipment/requirements	A valid .names file with at least one class		
Setup Instructions	Load the required .names file		
Failure Correction Procedure	Add a button or hardware input that is linked to a function to change the name. This may require a more complex system to show the current name of the label as it is being changed		
Engineer/Technician	Samuel Harrison/Stephen Anderson		
Individual Results:	Fail		

Test 10e is a black box test that demonstrates a common use case that a user might have; they may want to pick a better suited name or a mistake was made that needs to be corrected. It would be beneficial if there was a proper way to edit the existing names in the class file instead of deleting and making the class again.

Table 29 - Test Number 11

ID:	11	Description:	Search for a desired class name
Test Type:	Quality	Success Criteria:	The search results in a class with a matching name to the search and any partial matches if appropriate
Number of Attempts:	2	Comments:	Works the same as the other search function; still case sensitive but is functional
List of Equipment/requirements	A valid .names file with a few differently named classes		
Setup Instructions	Load the required .names file		
Failure Correction Procedure	I believe that the same function is being used as to search for an image so follow the same correction procedure as that function		
Engineer/Technician	Samuel Harrison/Stephen Anderson		
Individual Results:	Pass		

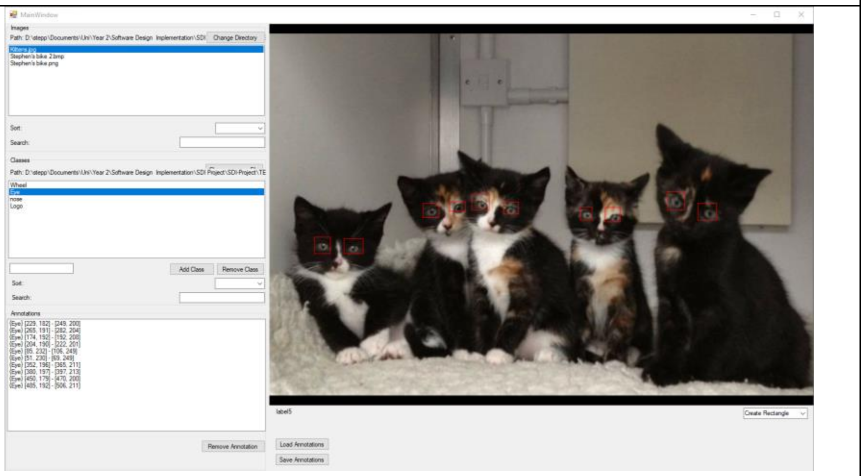
Screenshots	 Path: D:\stepp\Documents\Uni\Year 2\Software Design Implem Wheel Eye nose Logo *Figure 45 Unsearched Class List*  Path: D:\stepp\Documents\Uni\Year 2\Software Design Implem Eye Search: Ey *Figure 46 Searched Class List 1*  Path: D:\stepp\Documents\Uni\Year 2\Software Design Implem nose Search: os *Figure 47 Searched Class List 2*

	 Path: D:\stepp\Documents\Uni\Year 2\Software Design Implem nose Logo Search: o *Figure 48 Searched Class List 3*  Path: D:\stepp\Documents\Uni\Year 2\Software Design Implem Search: a *Figure 49 Searched Class List 4*

Test 11 is a black box acceptance test that demonstrates the use case of a user wanting to filter down the list of available class names. This is a very real and likely use case that needs to work as it is quite possible that a user will have many different class names. As can be seen from Figure 46, Figure 47, Figure 48 and Figure 49 the search function works well but is currently case sensitive which is not a major problem but it would be better if it wasn't.

Table 30 - Test Number 12a

ID:	12a	Description:	Add annotation
Test Type:	Quality	Success Criteria:	Adds a new annotation to the image
Number of Attempts:	3	Comments:	Can draw rectangles but it would be nice if it showed the drawing in real time
List of Equipment/requirements	A valid image as well as a .names file with a class in it		

Setup Instructions	Load the image and .names file before selecting the class from the list. Then select “Create Rectangle” from the dropdown just under the bottom right corner of the image workspace
Failure Correction Procedure	Discuss with the software developer the best way to implement drawing to a workspace – probably using a type of canvas bitmap
Engineer/Technician	Samuel Harrison/Stephen Anderson
Individual Results:	Pass
Screenshots	 MainWindow Images Path: D:\stepp\Documents\Uni\Year 2\Software Design Implementation\SDI Kittens.jpg Stephen's bike 2.bmp Stephen's bike.png Sort: Search: Classes Path: D:\stepp\Documents\Uni\Year 2\Software Design Implementation\SDI Project\SDI-Project\TE Wheel Eye nose Logo Sort: Search: Annotations (Eye) [229, 182] - [249, 200] (Eye) [265, 191] - [282, 204] (Eye) [174, 192] - [192, 208] (Eye) [204, 190] - [222, 201] (Eye) [85, 232] - [106, 249] (Eye) [51, 230] - [69, 249] (Eye) [352, 196] - [365, 211] (Eye) [380, 197] - [397, 213] (Eye) [450, 179] - [470, 200] (Eye) [485, 192] - [506, 211] label5 Create Rectangle Change Directory · Add Class · Remove Class · Remove Annotation · Load Annotations · Save Annotations *Figure 50 Kittens Eyes Annotated*

Test 12a is a black box integration test designed to show that different aspects such as selecting an image, class name and annotation shape work correctly which then allows the user to draw annotations on their chosen image. These annotations can be any size within the workspace which is very good as it gives the user a lot of freedom.

Table 31 - Test Number 12b

ID:	12b	Description:	Save annotations
Test Type:	Quality	Success Criteria:	Opens up a dialog box that allows the user to select a folder and file name for the annotation file
Number of Attempts:	1	Comments:	Saves the file but doesn’t let the user select a file name or where to save the file to
List of Equipment/requirements	A valid image, .names file with class name and at least one annotation on the image		
Setup Instructions	Load the required files, draw an annotation and then press the “Save Annotation” button		
Failure Correction Procedure	Discuss with the software developer the best format to save the data and then how to implement the conversion		
Engineer/Technician	Samuel Harrison/Stephen Anderson		

Individual Results:	Fail

Test 12b is a black box integration test designed to demonstrate whether a user can successfully save a file that stores the annotations made on a particular image, including whether they can name and save it in a particular location. This is an important use case and is likely to be one of the most common that the software undergoes.

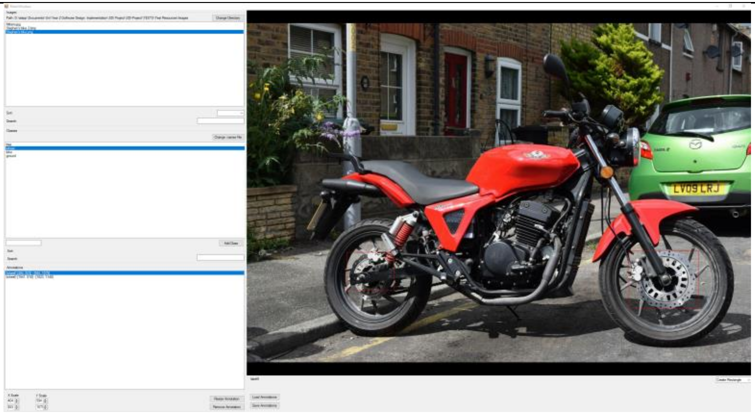
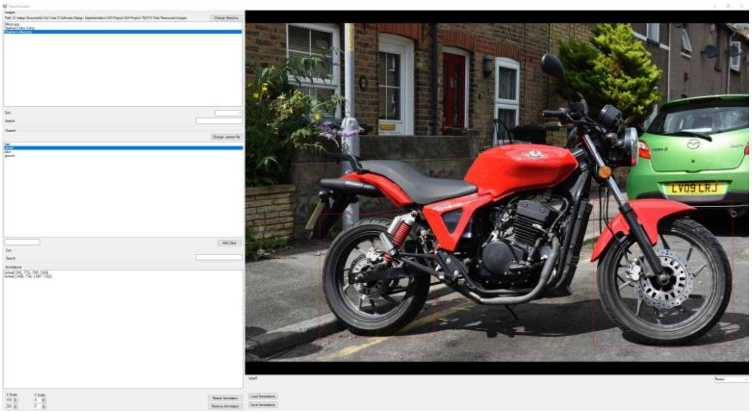
Table 32 - Test Number 12c

ID:	12c	Description:	Load annotations
Test Type:	Quality	Success Criteria:	Opens up a select file dialog box to allow the user to select and load the desired annotations file
Number of Attempts:	1	Comments:	Button loads the default file
List of Equipment/requirements	A valid annotation file, and related image and .names files		
Setup Instructions	Load the image and .names file		
Failure Correction Procedure	Discuss with the software developer the best way to let the user choose the desired file and then convert that to something the software can use		
Engineer/Technician	Samuel Harrison/Stephen Anderson		
Individual Results:	Fail		

Test 12c is a black box integration test to see if a dialog box opens that lets the user pick a particular file which will then load the annotations about an image (if it is already open) and draw them to the workspace.

Table 33 - Test Number 12d

ID:	12d	Description:	Resize annotations
Test Type:	Quality	Success Criteria:	A created annotation should be able to be selected and the size of it can be manually changed
Number of Attempts:	2	Comments:	Works well except the screen flickers in-between resizes
List of Equipment/requirements	A valid image, .names file and annotation		
Setup Instructions	Load the image, the .names file and the annotation		
Failure Correction Procedure	Work with the software developer to work out the best implementation		
Engineer/Technician	Samuel Harrison/Stephen Anderson		

Individual Results:	Pass
	 Images Change Directory Sort Search Classes Change .names file Add Class Annotations X Scale Y Scale Resize Annotation Remove Annotation Load Annotations Save Annotations *Figure 51 NAC 12 Small Annotations on Wheels*  Images Change Directory Sort Search Classes Change .names file Add Class Annotations X Scale Y Scale Resize Annotation Remove Annotation Load Annotations Save Annotations *Figure 52 NAC 12 Large Annotations On Wheels*

Test 12d is a black box integration test that demonstrates how it is possible to resize existing annotations to encompass different areas of the image that is loaded. As can be seen on Figure 51 – there are red rectangles around the centres of the wheels – and Figure 52 has large red rectangles encompassing the whole of each wheel. They are hard to see but they are there and they are the same annotation.

Table 34 - Test Number 12e

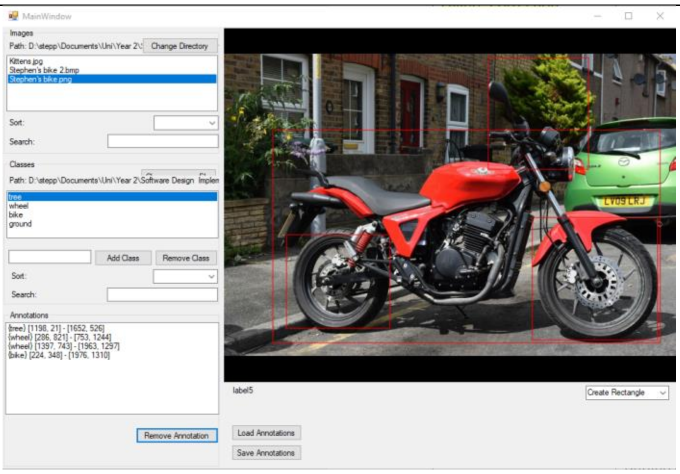
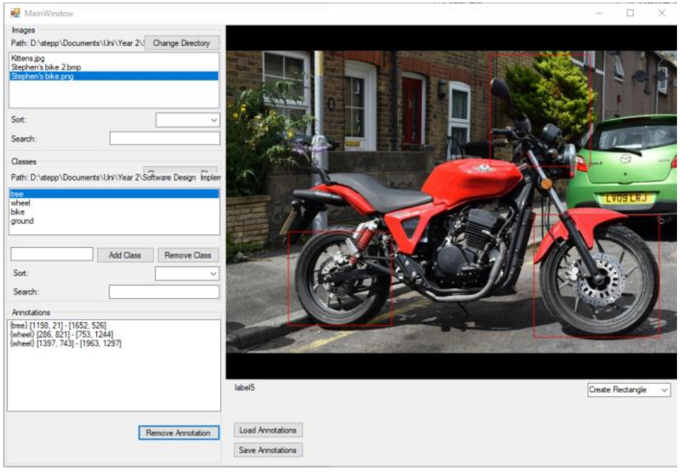
ID:	12e	Description:	Move annotations
Test Type:	Quality	Success Criteria:	Can drag and drop a selected annotation to another location on the image
Number of Attempts:	2	Comments:	It is hard to move the annotations as it is necessary to effectively resize them to the new location
List of Equipment/requirements	A valid image, .names file and annotation		
Setup Instructions	Load the image, the .names file and the annotation		

Failure Correction Procedure	Work with the software developer to work out the best implementation
Engineer/Technician	Samuel Harrison/Stephen Anderson
Individual Results:	Pass with concerns

Test 12e is a black box integration test that is designed to show how a user can take existing annotations and manipulate them. This is very useful as the original annotation may have been slightly off to one side and need to be move for the user's work.

Table 35 - Test Number 13

ID:	13	Description:	Remove annotation
Test Type:	Quality	Success Criteria:	A selected annotation will be removed from the list and also the drawn workspace
Number of Attempts:	3	Comments:	Works and the only area of concern is that it is possible to accidentally select the wrong annotation as they are not named very clearly and do not become obviously selected on the workspace when selected in the menu
List of Equipment/requirements	A valid image and an annotation related to it		
Setup Instructions	Load the image and make a new annotation for it		
Failure Correction Procedure	Discuss with the software developer the best way to select and remove a specific annotation from the list, then removing the drawn outline on the workspace		
Engineer/Technician	Samuel Harrison/Stephen Anderson		
Individual Results:	Pass		

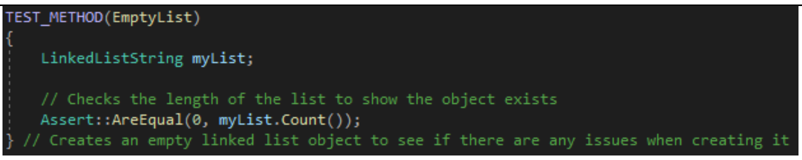
Screenshots	 MainWindow Images Change Directory Sort: Search: Classes Add Class Remove Class Sort: Search: Annotations Remove Annotation label5 Create Rectangle Load Annotations Save Annotations Path: D:\stepp\Documents\Uni\Year 2\ Kittens.jpg Stephen's bike 2.bmp Stephen's bike.png Path: D:\stepp\Documents\Uni\Year 2\Software Design Implen tree wheel bike ground {tree} [1198, 21] - [1652, 526] {wheel} [286, 821] - [753, 1244] {wheel} [1397, 743] - [1963, 1297] {bike} [224, 348] - [1976, 1310] *Figure 53 NAC12 Annotated*  MainWindow Images Change Directory Sort: Search: Classes Add Class Remove Class Sort: Search: Annotations Remove Annotation label5 Create Rectangle Load Annotations Save Annotations Path: D:\stepp\Documents\Uni\Year 2\ Kittens.jpg Stephen's bike 2.bmp Stephen's bike.png Path: D:\stepp\Documents\Uni\Year 2\Software Design Implen tree wheel bike ground {tree} [1198, 21] - [1652, 526] {wheel} [286, 821] - [753, 1244] {wheel} [1397, 743] - [1963, 1297] *Figure 54 NAC 12 With an Annotation Removed*

Test 13 is a black box integration test that checks to see if a user can select and remove a desired annotation from the workspace. This is an important test as removing an annotation is likely to be a very common use case that the user absolutely needs.

Table 36 - Test Number 14a

ID:	14a	Description:	Empty linked list
Test Type:	Quality	Success Criteria:	A linked list object can be created
Number of Attempts:	1	Comments:	Appears to work without issue
List of Equipment/requirements	A linked list class object		
Setup Instructions	Create an empty linked list		
Failure Correction Procedure	Discuss with the software developer on how to implement the class		
Engineer/Technician	Samuel Harrison/Stephen Anderson		

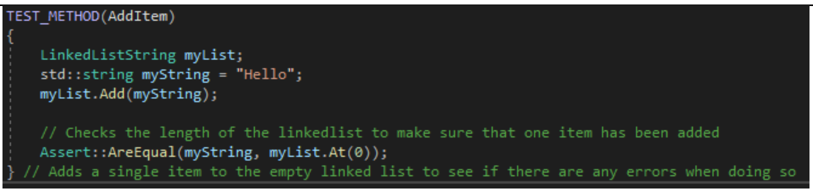
Individual Results:	Pass

Screenshots	
	`TEST_METHOD(EmptyList)`
	`{`
	`    LinkedListString myList;`
	
	`    // Checks the length of the list to show the object exists`
	`    Assert::AreEqual(0, myList.Count());`
	`} // Creates an empty linked list object to see if there are any issues when creating it`
	
	Figure 55 Empty Linked List Test Code

Test 14a is a black box unit test that determines if the linked list class is implemented at all and an object can be created without any errors occurring, as can be seen from Figure 55. This appears to work fine, with aspects of the object (such as how many items are contained within it) being accessible.

Table 37 - Test Number 14b

ID:	14b	Description:	A linked list can have an item added to it
Test Type:	Quality	Success Criteria:	An empty linked list can have an item added to it after creation
Number of Attempts:	1	Comments:	Appears to work without issue
List of Equipment/requirements	An empty linked list class object and a valid compatible string with only alphabetical characters in it		
Setup Instructions	Create the empty linked list and the item separately		
Failure Correction Procedure	Discuss with the software developer on how to implement the class and the best course of action for the potential errors		
Engineer/Technician	Samuel Harrison/Stephen Anderson		
Individual Results:	Pass		

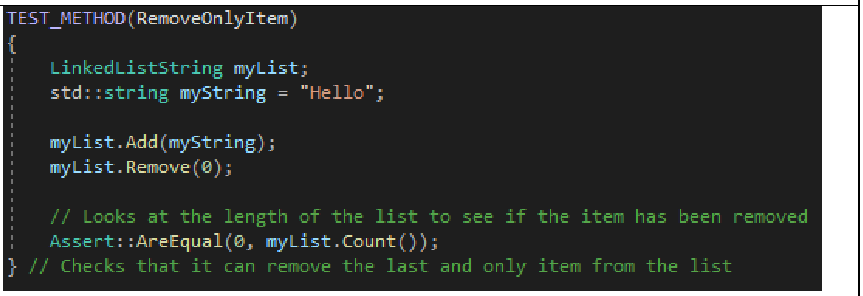
Screenshots	
	`TEST_METHOD(AddItem)`
	`{`
	`    LinkedListString myList;`
	`    std::string myString = "Hello";`
	`    myList.Add(myString);`
	
	`    // Checks the length of the linkedlist to make sure that one item has been added`
	`    Assert::AreEqual(myString, myList.At(0));`
	`} // Adds a single item to the empty linked list to see if there are any errors when doing so`
	
	Figure 56 Add Item Linked List Test Code		

Test 14b is a black box unit test designed to ensure that an item can be added to the linked list and then have the value of that item accessed correctly, as shown in Figure 56. This is important as if the class doesn't pass this test then it is not going to work as a data structure that can be used elsewhere in the code.

Table 38 - Test Number 14c

ID:	14c	Description:	A linked list can have the only item removed from it

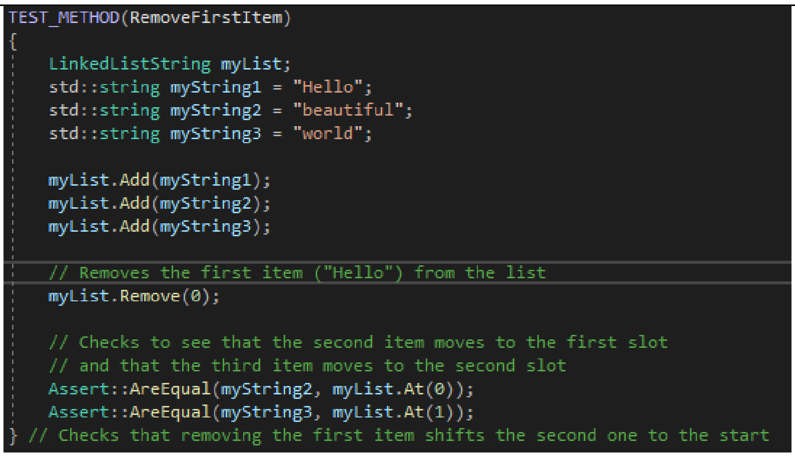
Test Type:	Quality	Success Criteria:	An empty linked list can have an item removed from it after creation
Number of Attempts:	1	Comments:	No apparent issues
List of Equipment/requirements	An empty linked list class object and a valid compatible string with only alphabetical characters in it		
Setup Instructions	Create the empty linked list and add the item to it		
Failure Correction Procedure	Discuss with the software developer on how to implement the class, particularly about how to remove a reference to a particular item		
Engineer/Technician	Samuel Harrison/Stephen Anderson		
Individual Results:	Pass		

Screenshots	
	`TEST_METHOD(RemoveOnlyItem)`
	`{`
	`    LinkedListString myList;`
	`    std::string myString = "Hello";`
	
	`    myList.Add(myString);`
	`    myList.Remove(0);`
	
	`    // Looks at the length of the list to see if the item has been removed`
	`    Assert::AreEqual(0, myList.Count());`
	`} // Checks that it can remove the last and only item from the list`
	Figure 57 Remove Only Item Linked List Test Code		

Test 14c, from Figure 57, is a black box unit test that is designed to see if a single item can be removed from a linked list without throwing errors. This is necessary as the data structure needs to be able to handle having items removed from it, especially when it then becomes empty. As part of the test it also checks that the number of items in the linked list is updated correctly.

Table 39 - Test Number 14d

ID:	14d	Description:	A linked list can have the first item removed from it
Test Type:	Quality	Success Criteria:	The item can be removed and the remaining item can then be accessed without issues as the first item
Number of Attempts:	1	Comments:	Appears to work without issues
List of Equipment/requirements	A linked list and a valid compatible string with only alphabetical characters in it		
Setup Instructions	Create a linked list object and add the item to it		
Failure Correction Procedure	Discuss with the software developer on how to implement the class, particularly about how to remove a reference to a particular item and move the remaining items		
Engineer/Technician	Samuel Harrison/Stephen Anderson		

Individual Results:	Pass
Screenshots	

```
TEST_METHOD(RemoveFirstItem)
{
    LinkedListString myList;
    std::string myString1 = "Hello";
    std::string myString2 = "beautiful";
    std::string myString3 = "world";

    myList.Add(myString1);
    myList.Add(myString2);
    myList.Add(myString3);

    // Removes the first item ("Hello") from the list
    myList.Remove(0);

    // Checks to see that the second item moves to the first slot
    // and that the third item moves to the second slot
    Assert::AreEqual(myString2, myList.At(0));
    Assert::AreEqual(myString3, myList.At(1));
} // Checks that removing the first item shifts the second one to the start
```

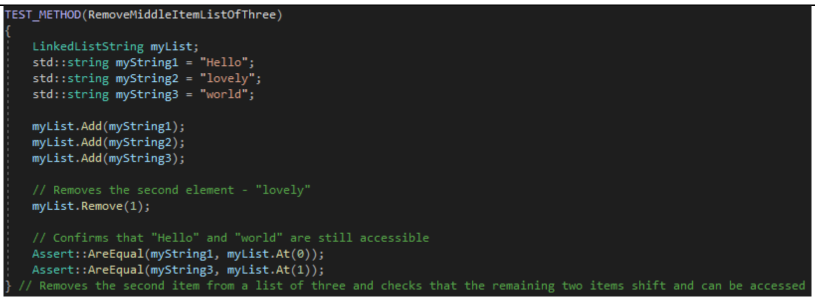
Figure 58 Remove First Item Linked List Test Code

Test 14d is a white box unit test that is designed to make sure that the first item in a list can be removed and then the other items in a list are properly shifted along so that the second item becomes the first, the third becomes the second etc. as can be seen from Figure 58.

Table 40 - Test Number 14e

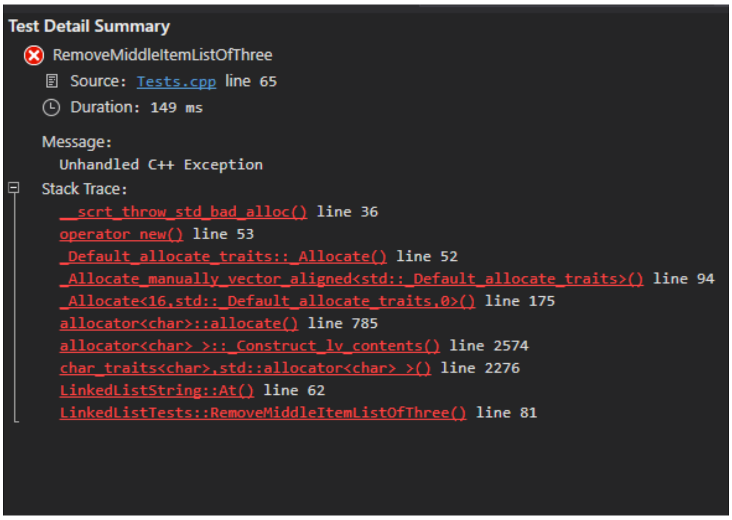
ID:	14e	Description:	A linked list of three can have the middle item removed from it
Test Type:	Quality	Success Criteria:	The item can be removed and the remaining items can then be accessed without issues
Number of Attempts:	5	Comments:	A c++ exception occurs, it appears as though the items are not accessed and adjusted correctly
List of Equipment/requirements	A linked list and three valid, compatible strings with only alphabetical characters in them		
Setup Instructions	Create a linked list object and add the items to it		
Failure Correction Procedure	Discuss with the software developer on how to implement the class, particularly about how to remove a reference to a particular item and move the remaining items		
Engineer/Technician	Samuel Harrison/Stephen Anderson		
Individual Results:	Fail		

Screenshots



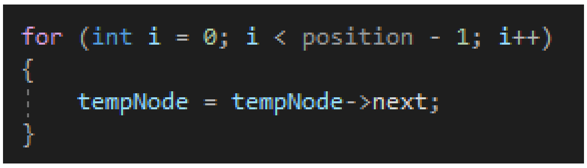
```
TEST_METHOD(RemoveMiddleItemListOfThree)
{
    LinkedListString myList;
    std::string myString1 = "Hello";
    std::string myString2 = "lovely";
    std::string myString3 = "world";

    myList.Add(myString1);
    myList.Add(myString2);
    myList.Add(myString3);

    // Removes the second element - "lovely"
    myList.Remove(1);

    // Confirms that "Hello" and "world" are still accessible
    Assert::AreEqual(myString1, myList.At(0));
    Assert::AreEqual(myString3, myList.At(1));
} // Removes the second item from a list of three and checks that the remaining two items shift and can be accessed
```

Figure 59 Remove Middle Item Linked List of Three Test Code



```
Test Detail Summary
  RemoveMiddleItemListOfThree
    Source: Tests.cpp line 65
    Duration: 149 ms

    Message:
      Unhandled C++ Exception
    Stack Trace:
      __scrt_throw_std_bad_alloc() line 36
      operator new() line 53
      _Default_allocate_traits::_Allocate() line 52
      _Allocate_manually_vector_aligned<std::_Default_allocate_traits>() line 94
      _Allocate<16,std::_Default_allocate_traits,0>() line 175
      allocator<char>::allocate() line 785
      allocator<char> >::_Construct_lv_contents() line 2574
      char_traits<char>,std::allocator<char> >() line 2276
      LinkedListString::At() line 62
      LinkedListTests::RemoveMiddleItemListOfThree() line 81
```

Figure 60 Remove Middle Item Linked List of Three Exception



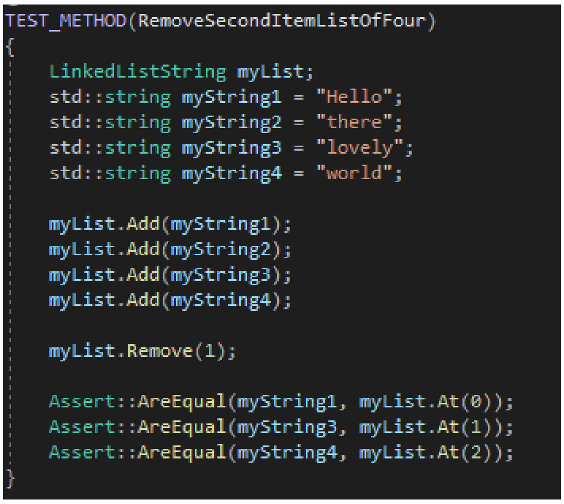
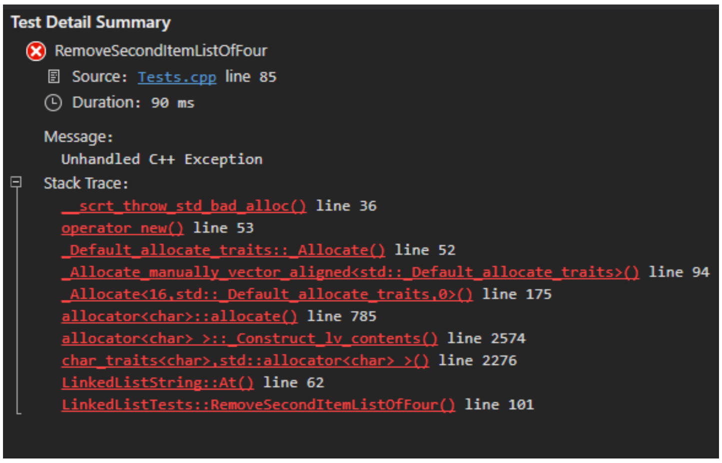
```
for (int i = 0; i < position - 1; i++)
{
    tempNode = tempNode->next;
}
```

Figure 61 Linked List Potentially Problematic Code

Test 14e is a white box test that is designed to see that the second item from a list of three can successfully be removed and the third item can be moved. This is done as it requires only moving one item but looking at the existing implementation for the data structure there is a logic issue that could occur as part of a for loop, shown in Figure 61, as if the position given is 1 (as the second object in the list) then the loop is never executed due to 'i' not being less than 0.

Table 41 - Test Number 14e (i)

ID:	14e (i)	Description:	A linked list of four can have the second item removed from it
Test Type:	Quality	Success Criteria:	The item can be removed and the remaining items can then be accessed without issues
Number of Attempts:	1	Comments:	Confirms my suspicions that it doesn't like removing that

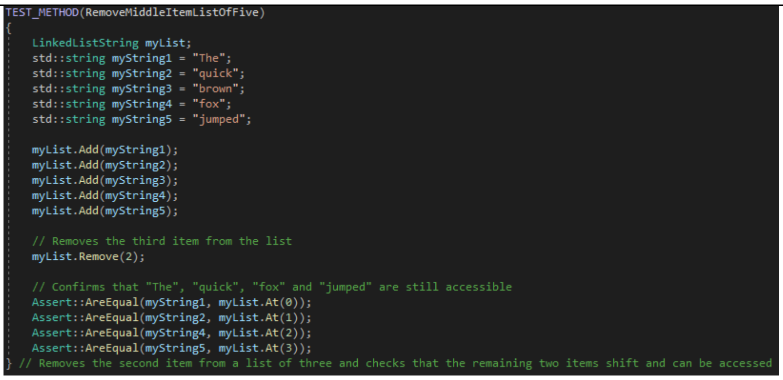
List of Equipment/requirements	A linked list and four valid, compatible strings with only alphabetical characters in them
Setup Instructions	Create a linked list object and add the items to it
Failure Correction Procedure	Discuss with the software developer on what the cause is and how to solve the code not being executed.
Engineer/Technician	Samuel Harrison/Stephen Anderson
Individual Results:	Fail
Screenshots	 *Figure 62 Remove Second Item Linked List of Four Test Code*  *Figure 63 Remove Second Item Linked List of Four Test Exception*

```
TEST_METHOD(RemoveSecondItemListOfFour)
{
    LinkedListString myList;
    std::string myString1 = "Hello";
    std::string myString2 = "there";
    std::string myString3 = "lovely";
    std::string myString4 = "world";

    myList.Add(myString1);
    myList.Add(myString2);
    myList.Add(myString3);
    myList.Add(myString4);

    myList.Remove(1);

    Assert::AreEqual(myString1, myList.At(0));
    Assert::AreEqual(myString3, myList.At(1));
    Assert::AreEqual(myString4, myList.At(2));
}
```

```
Test Detail Summary
  ☒ RemoveSecondItemListOfFour
     Source: Tests.cpp line 85
     Duration: 90 ms

     Message:
       Unhandled C++ Exception
     Stack Trace:
       __scrt_throw_std_bad_alloc() line 36
       operator new() line 53
       _Default_allocate_traits::_Allocate() line 52
       _Allocate_manually_vector_aligned<std::_Default_allocate_traits>() line 94
       _Allocate<16,std::_Default_allocate_traits,0>() line 175
       allocator<char>::allocate() line 785
       allocator<char> >::_Construct_lv_contents() line 2574
       char_traits<char>,std::allocator<char> >() line 2276
       LinkedListString::At() line 62
       LinkedListTests::RemoveSecondItemListOfFour() line 101
```

Table 42 - Test Number 14f

ID:	14f	Description:	A linked list of five can have the middle item removed from it

<table>
<tr><td>Test Type:</td><td>Quality</td><td>Success Criteria:</td><td>The item can be removed and the remaining items can then be accessed without issues</td></tr>
<tr><td>Number of Attempts:</td><td>2</td><td>Comments:</td><td>Appears to work without issues</td></tr>
<tr><td>List of Equipment/requirements</td><td colspan="3">A linked list and five valid, compatible strings with only alphabetical characters in them</td></tr>
<tr><td>Setup Instructions</td><td colspan="3">Create a linked list object and add the items to it</td></tr>
<tr><td>Failure Correction Procedure</td><td colspan="3">Discuss with the software developer on how to implement the class, particularly about how to remove a reference to a particular item and move the remaining items</td></tr>
<tr><td>Engineer/Technician</td><td colspan="3">Samuel Harrison/Stephen Anderson</td></tr>
<tr><td>Individual Results:</td><td colspan="3">Pass</td></tr>
<tr><td>Screenshots</td><td colspan="3">


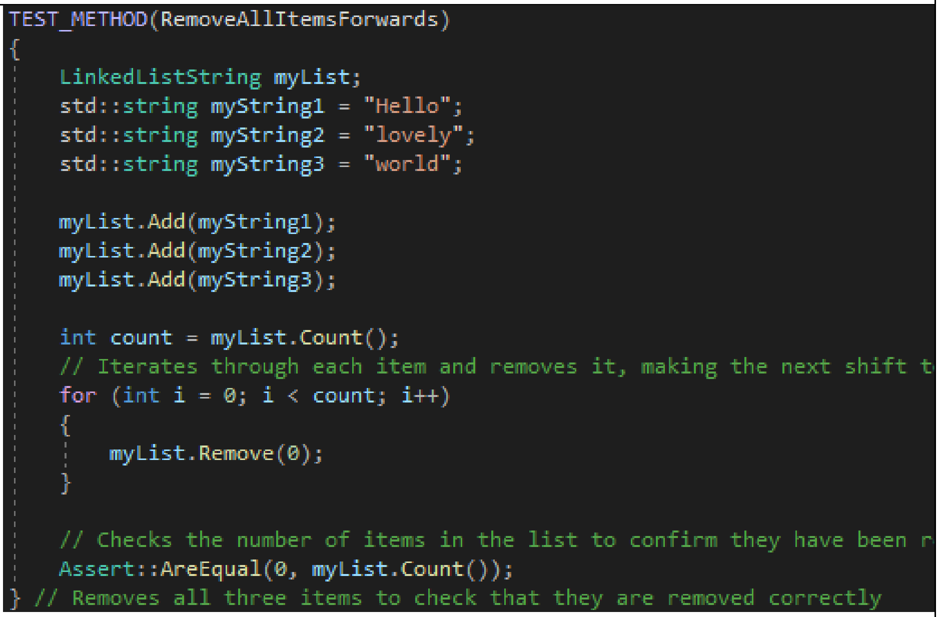
```
TEST_METHOD(RemoveMiddleItemListOfFive)
{
    LinkedListString myList;
    std::string myString1 = "The";
    std::string myString2 = "quick";
    std::string myString3 = "brown";
    std::string myString4 = "fox";
    std::string myString5 = "jumped";

    myList.Add(myString1);
    myList.Add(myString2);
    myList.Add(myString3);
    myList.Add(myString4);
    myList.Add(myString5);

    // Removes the third item from the list
    myList.Remove(2);

    // Confirms that "The", "quick", "fox" and "jumped" are still accessible
    Assert::AreEqual(myString1, myList.At(0));
    Assert::AreEqual(myString2, myList.At(1));
    Assert::AreEqual(myString4, myList.At(2));
    Assert::AreEqual(myString5, myList.At(3));
} // Removes the second item from a list of three and checks that the remaining two items shift and can be accessed
```

Figure 64 Remove Middle Item List Of Five Test Code
</td></tr>
</table>

Test 14f is a white box unit test designed to show that the data structure can have an item removed from the middle of it without losing data, order or causing exceptions. The test itself can be seen in Figure 64 and is chosen to show that although when the second item is removed from a list, it causes issues, if an item is further along the list than that then it works properly.

Table 43 - Test Number 14g

<table>
<tr><td>ID:</td><td>14g</td><td>Description:</td><td>A linked list can have all items removed from it going from front to back</td></tr>
<tr><td>Test Type:</td><td>Quality</td><td>Success Criteria:</td><td>The items can be removed without issues</td></tr>
<tr><td>Number of Attempts:</td><td>2</td><td>Comments:</td><td>No apparent issues</td></tr>
<tr><td>List of Equipment/requirements</td><td colspan="3">A linked list and three valid, compatible strings with only alphabetical characters in them</td></tr>
<tr><td>Setup Instructions</td><td colspan="3">Create a linked list object and add the items to it</td></tr>
</table>

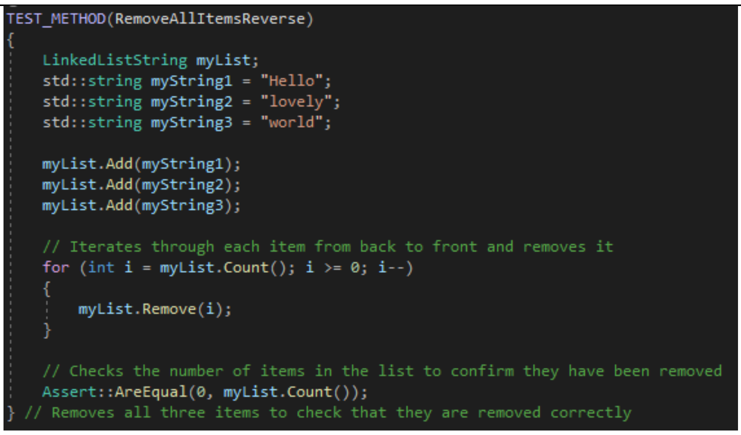
Failure Correction Procedure	Discuss with the software developer on how to implement the class, particularly about how to remove a reference to a particular item and move the remaining items
Engineer/Technician	Samuel Harrison/Stephen Anderson
Individual Results:	Pass
Screenshots	 *Figure 65 Remove All Items Forwards Test Code*

```
TEST_METHOD(RemoveAllItemsForwards)
{
    LinkedListString myList;
    std::string myString1 = "Hello";
    std::string myString2 = "lovely";
    std::string myString3 = "world";

    myList.Add(myString1);
    myList.Add(myString2);
    myList.Add(myString3);

    int count = myList.Count();
    // Iterates through each item and removes it, making the next shift t
    for (int i = 0; i < count; i++)
    {
        myList.Remove(0);
    }

    // Checks the number of items in the list to confirm they have been r
    Assert::AreEqual(0, myList.Count());
} // Removes all three items to check that they are removed correctly
```

Test 14g is a white box unit test that tests to make sure that the items are shifted along when the head is deleted, which when done repeatedly should result in the list becoming empty – as shown in Figure 65.

Table 44 - Test Number 14h

ID:	14h	Description:	A linked list can have all items removed from it going from back to front
Test Type:	Quality	Success Criteria:	The items can be removed without issues
Number of Attempts:	2	Comments:	No apparent issues
List of Equipment/requirements	A linked list and three valid, compatible strings with only alphabetical characters in them		
Setup Instructions	Create a linked list object and add the items to it		
Failure Correction Procedure	Discuss with the software developer on how to implement the class, particularly about how to remove a reference to a particular item and move the remaining items		
Engineer/Technician	Samuel Harrison/Stephen Anderson		

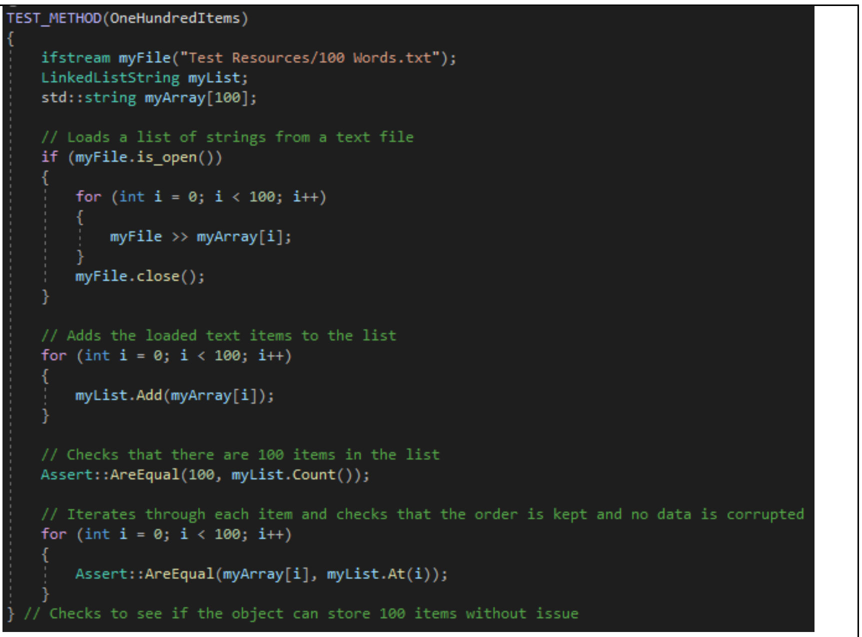
Individual Results:	Pass
Screenshots	 *Figure 66 Remove All Items Reverse Test Code*

```
TEST_METHOD(RemoveAllItemsReverse)
{
    LinkedListString myList;
    std::string myString1 = "Hello";
    std::string myString2 = "lovely";
    std::string myString3 = "world";

    myList.Add(myString1);
    myList.Add(myString2);
    myList.Add(myString3);

    // Iterates through each item from back to front and removes it
    for (int i = myList.Count(); i >= 0; i--)
    {
        myList.Remove(i);
    }

    // Checks the number of items in the list to confirm they have been removed
    Assert::AreEqual(0, myList.Count());
} // Removes all three items to check that they are removed correctly
```

Test 14h is another white box unit test and is the opposite of test 14g in that the structure should be able to handle the last object pointer, or the tail, being moved until it reaches the head.

Table 45 - Test Number 14i

ID:	14i	Description:	A linked list can contain at least one hundred items in it
Test Type:	Quantity	Success Criteria:	The linked list can be created and store one hundred items within it without any issues
Number of Attempts:	1	Comments:	Works flawlessly
List of Equipment/requirements	A linked list and one hundred valid compatible strings stored in a .txt file with only alphabetical characters them		
Setup Instructions	Create the linked list object with the one hundred items loaded into an array to make it easier to add to the list		
Failure Correction Procedure	Discuss with the software developer on what the issues might be when dealing with larger numbers of items		
Engineer/Technician	Samuel Harrison/Stephen Anderson		
Individual Results:	Pass		

Screenshots	 *Figure 67 Linked List 100 Items Test Code*

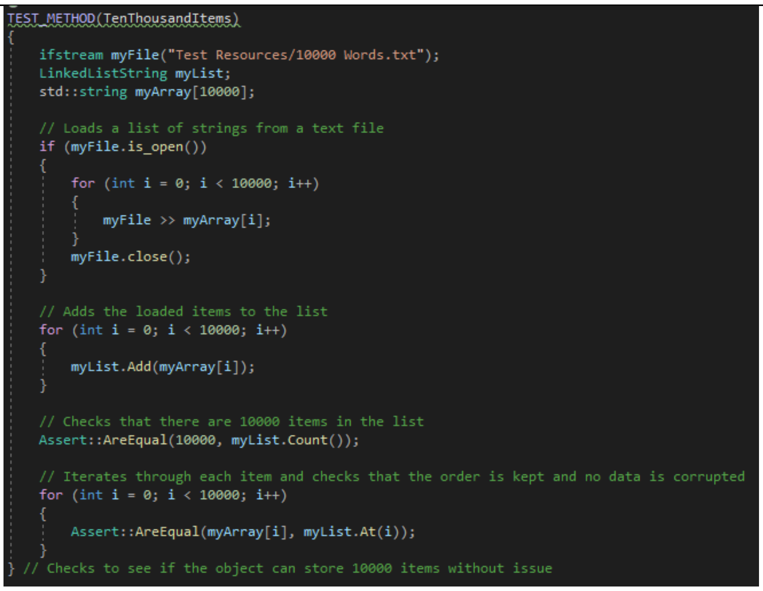
```
TEST_METHOD(OneHundredItems)
{
    ifstream myFile("Test Resources/100 Words.txt");
    LinkedListString myList;
    std::string myArray[100];

    // Loads a list of strings from a text file
    if (myFile.is_open())
    {
        for (int i = 0; i < 100; i++)
        {
            myFile >> myArray[i];
        }
        myFile.close();
    }

    // Adds the loaded text items to the list
    for (int i = 0; i < 100; i++)
    {
        myList.Add(myArray[i]);
    }

    // Checks that there are 100 items in the list
    Assert::AreEqual(100, myList.Count());

    // Iterates through each item and checks that the order is kept and no data is corrupted
    for (int i = 0; i < 100; i++)
    {
        Assert::AreEqual(myArray[i], myList.At(i));
    }
} // Checks to see if the object can store 100 items without issue
```

Test 14i is a black box acceptance test that is designed to show a likely scenario for when the software, and the data structure, are in use. The data structure would not be very good if it could not store even just 100 items in it. As can be seen from Figure 67, the string items are loaded from a file into an array which is then incrementally added to the linked list object and then checked at the end to confirm that no data or order is lost.

Table 46 - Test Number 14j

ID:	14j	Description:	A linked list can contain at least ten thousand items in it
Test Type:	Quantity	Success Criteria:	The linked list can be created and store one thousand items within it without any issues
Number of Attempts:	1	Comments:	Works without issues
List of Equipment/requirements	A linked list and ten thousand valid compatible strings stored in a .txt file with only alphabetical characters them		
Setup Instructions	Create the linked list object with the ten thousand items loaded into an array to make it easier to add to the list		
Failure Correction Procedure	Discuss with the software developer on what the issues might be when dealing with larger numbers of items		
Engineer/Technician	Samuel Harrison/Stephen Anderson		
Individual Results:	Pass		

<table>
<tr>
<td>Screenshots</td>
<td>



```
TEST_METHOD(TenThousandItems)
{
    ifstream myFile("Test Resources/10000 Words.txt");
    LinkedListString myList;
    std::string myArray[10000];

    // Loads a list of strings from a text file
    if (myFile.is_open())
    {
        for (int i = 0; i < 10000; i++)
        {
            myFile >> myArray[i];
        }
        myFile.close();
    }

    // Adds the loaded items to the list
    for (int i = 0; i < 10000; i++)
    {
        myList.Add(myArray[i]);
    }

    // Checks that there are 10000 items in the list
    Assert::AreEqual(10000, myList.Count());

    // Iterates through each item and checks that the order is kept and no data is corrupted
    for (int i = 0; i < 10000; i++)
    {
        Assert::AreEqual(myArray[i], myList.At(i));
    }
} // Checks to see if the object can store 10000 items without issue
```

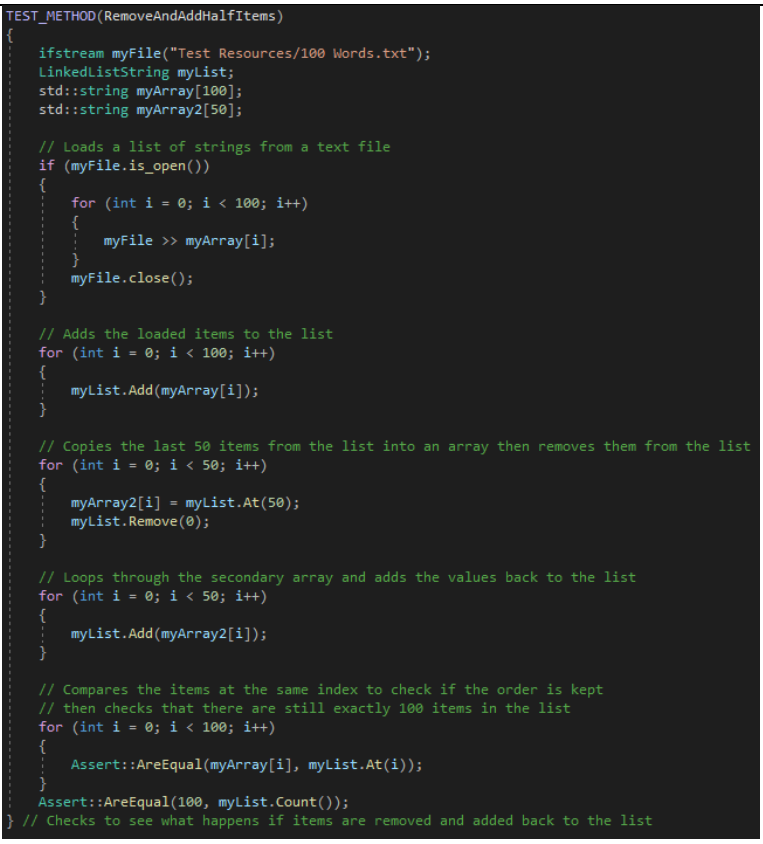
Figure 68 Linked List 10000 Items Test Code

</td>
</tr>
</table>

Test 14j is similar to test 14i but takes it a bit further to see if the data structure can handle the more extreme end of usage with a number of items in it that is significantly larger than is expected from general use of the software. As can be seen from Figure 68 it is a similar test but has many more data points.

Table 47 - Test Number 14k

ID:	14k	Description:	A linked list can have half of its items removed from the middle and added back again
Test Type:	Quality	Success Criteria:	The linked list can the last fifty of its one hundred items removed and added back again without losing the order
Number of Attempts:	1	Comments:	Appears to work without losing any data or the order that data is in
List of Equipment/requirements	A linked list and one hundred valid compatible strings with only alphabetical characters them		
Setup Instructions	Create the linked list object with the one hundred items – it does not matter how the items are added to the list at this stage		
Failure Correction Procedure	Discuss with the software developer on how to implement the class, particularly removing and adding items		
Engineer/Technician	Samuel Harrison/Stephen Anderson		
Individual Results:	Pass		

<table>
<tr><td>Screenshots</td><td>
<pre>
TEST_METHOD(RemoveAndAddHalfItems)
{
    ifstream myFile("Test Resources/100 Words.txt");
    LinkedListString myList;
    std::string myArray[100];
    std::string myArray2[50];

    // Loads a list of strings from a text file
    if (myFile.is_open())
    {
        for (int i = 0; i &lt; 100; i++)
        {
            myFile &gt;&gt; myArray[i];
        }
        myFile.close();
    }

    // Adds the loaded items to the list
    for (int i = 0; i &lt; 100; i++)
    {
        myList.Add(myArray[i]);
    }

    // Copies the last 50 items from the list into an array then removes them from the list
    for (int i = 0; i &lt; 50; i++)
    {
        myArray2[i] = myList.At(50);
        myList.Remove(0);
    }

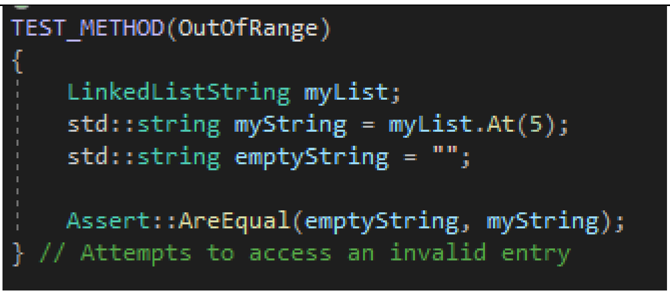
    // Loops through the secondary array and adds the values back to the list
    for (int i = 0; i &lt; 50; i++)
    {
        myList.Add(myArray2[i]);
    }

    // Compares the items at the same index to check if the order is kept
    // then checks that there are still exactly 100 items in the list
    for (int i = 0; i &lt; 100; i++)
    {
        Assert::AreEqual(myArray[i], myList.At(i));
    }
    Assert::AreEqual(100, myList.Count());
} // Checks to see what happens if items are removed and added back to the list
</pre>
<i>Figure 69 Remove and Add Half Items Linked List Test Code</i></td></tr>
</table>

Test 14k is a black box integration test designed to show that it is possible to add and remove a reasonable number of items from the data structure without losing data integrity. This is important as if lots of changes are made to the linked list then it should be able to handle that without causing errors or potential problems when reading from it again.

Table 48 - Test Number 14l

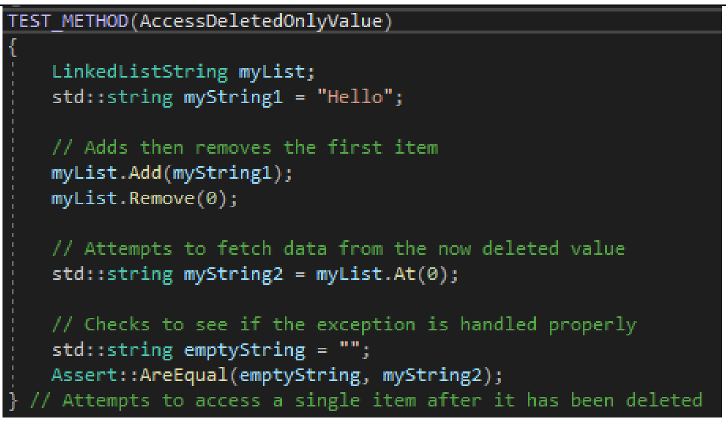
ID:	14l	Description:	Attempt to access a non-existent element
Test Type:	Quality	Success Criteria:	The linked list should handle the out of range exception
Number of Attempts:	1	Comments:	Handles without issue
List of Equipment/requirements	An empty linked list		
Setup Instructions	Create the linked list object		
Failure Correction Procedure	Discuss with the software developer on how to implement the class		
Engineer/Technician	Samuel Harrison/Stephen Anderson		
Individual Results:	Pass		

Screenshots	 *Figure 70 Linked List Non-Existent Element Test Code*

```
TEST_METHOD(OutOfRange)
{
    LinkedListString myList;
    std::string myString = myList.At(5);
    std::string emptyString = "";

    Assert::AreEqual(emptyString, myString);
} // Attempts to access an invalid entry
```

Test 14l is a black box acceptance test that demonstrates the linked list structure can handle attempts to access a non-existent index location in the list. It is a simple but necessary test as it means that instead of throwing an exception, it just returns an empty value that prevents the software from breaking as seen in Figure 70.

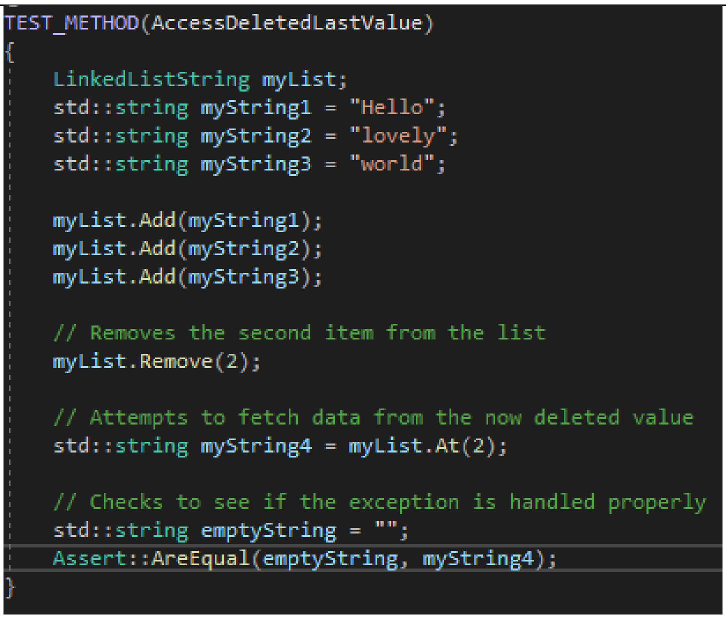
Table 49 - Test Number 14m

ID:	14m	Description:	Attempt to access the only element from a list after it is removed
Test Type:	Quality	Success Criteria:	The linked list should handle the out of range exception
Number of Attempts:	1	Comments:	Works without issue
List of Equipment/requirements	An empty linked list and an item that can be added to it		
Setup Instructions	Create the linked list object		
Failure Correction Procedure	Discuss with the software developer on how to implement the class, especially how to deal with how to handle attempts to access indexes out of range		
Engineer/Technician	Samuel Harrison/Stephen Anderson		
Individual Results:	Pass		
Screenshots	 *Figure 71 Access Deleted Value From Linked List Test Code*		

```
TEST_METHOD(AccessDeletedOnlyValue)
{
    LinkedListString myList;
    std::string myString1 = "Hello";

    // Adds then removes the first item
    myList.Add(myString1);
    myList.Remove(0);

    // Attempts to fetch data from the now deleted value
    std::string myString2 = myList.At(0);

    // Checks to see if the exception is handled properly
    std::string emptyString = "";
    Assert::AreEqual(emptyString, myString2);
} // Attempts to access a single item after it has been deleted
```

Test 14m is a white box integration test designed to show if an item can still be accessed even after the reference to it should have been removed from the linked list structure, instead returning either a null value or an empty string as seen by the test in Figure 71

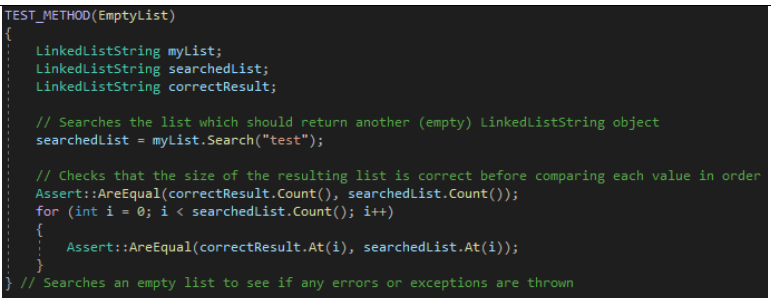
Table 50 - Test Number 14n

ID:	14n	Description:	Attempt to access the last element from a list after it is removed
Test Type:	Quality	Success Criteria:	The linked list should handle the out of range exception
Number of Attempts:	1	Comments:	Works without issue
List of Equipment/requirements	An empty linked list with three items that can be added to it		
Setup Instructions	Create the linked list object		
Failure Correction Procedure	Discuss with the software developer on how to implement the class, especially how to deal with how to handle attempts to access indexes out of range		
Engineer/Technician	Samuel Harrison/Stephen Anderson		
Individual Results:	Pass		
Screenshots	 *Figure 72 Access Deleted Last Value From Linked List Test Code*		

```
TEST_METHOD(AccessDeletedLastValue)
{
    LinkedListString myList;
    std::string myString1 = "Hello";
    std::string myString2 = "lovely";
    std::string myString3 = "world";

    myList.Add(myString1);
    myList.Add(myString2);
    myList.Add(myString3);

    // Removes the second item from the list
    myList.Remove(2);

    // Attempts to fetch data from the now deleted value
    std::string myString4 = myList.At(2);

    // Checks to see if the exception is handled properly
    std::string emptyString = "";
    Assert::AreEqual(emptyString, myString4);
}
```

Test 14n is another version of test 14m that is designed to test what happens when there are other items in the list and it is the last item that is deleted, shown in Figure 72. These tests are important is it proves that the data structure can handle the removal of items without compromising data integrity.

Table 51 - Test Number 15a

ID:	15a	Description:	Searching an empty list
Test Type:	Quality	Success Criteria:	Searching for a valid item type in an empty list causes no issues
Number of Attempts:	2	Comments:	
List of Equipment/requirements	An empty list and a valid compatible search term with only alphabetical characters		
Setup Instructions	Create the list and the item		
Failure Correction Procedure	Discuss with the software developer how best to implement a linear search		
Engineer/Technician	Samuel Harrison/Stephen Anderson		
Individual Results:	Pass		
Screenshots	 *Figure 73 Empty List Search Test Code*		

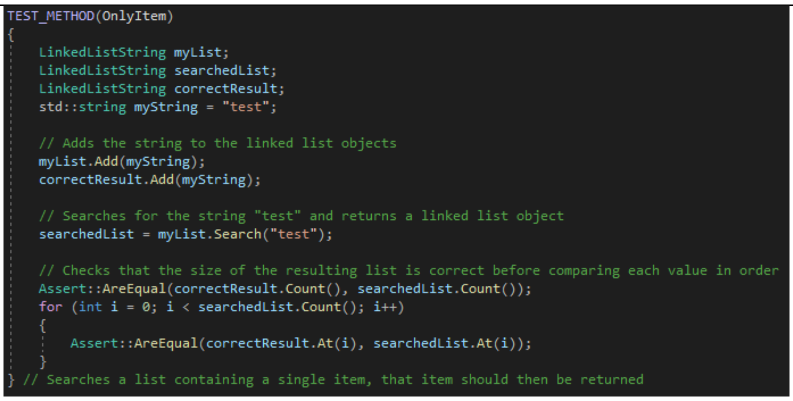
```
TEST_METHOD(EmptyList)
{
    LinkedListString myList;
    LinkedListString searchedList;
    LinkedListString correctResult;

    // Searches the list which should return another (empty) LinkedListString object
    searchedList = myList.Search("test");

    // Checks that the size of the resulting list is correct before comparing each value in order
    Assert::AreEqual(correctResult.Count(), searchedList.Count());
    for (int i = 0; i < searchedList.Count(); i++)
    {
        Assert::AreEqual(correctResult.At(i), searchedList.At(i));
    }
} // Searches an empty list to see if any errors or exceptions are thrown
```

Test 15a is a black box unit test that is designed to check that an empty linked list can be searched without any errors being created. This is important because if there is ever a time that the function is run in the background before a user has had a chance to even load something into one of the available lists.

Table 52 - Test Number 15b

ID:	15b	Description:	Searching a list that contains a single object
Test Type:	Quality	Success Criteria:	Searching as list for the only item returns the correct value
Number of Attempts:	1	Comments:	
List of Equipment/requirements	A list and a valid compatible item and search term with only alphabetical characters in them		
Setup Instructions	Create the list with the item in it		
Failure Correction Procedure	Discuss with the software developer how best to implement a linear search		
Engineer/Technician	Samuel Harrison/Stephen Anderson		

<table>
<tr><td>Individual Results:</td><td>Pass</td></tr>
<tr><td>Screenshots</td><td>



```
TEST_METHOD(OnlyItem)
{
    LinkedListString myList;
    LinkedListString searchedList;
    LinkedListString correctResult;
    std::string myString = "test";

    // Adds the string to the linked list objects
    myList.Add(myString);
    correctResult.Add(myString);

    // Searches for the string "test" and returns a linked list object
    searchedList = myList.Search("test");

    // Checks that the size of the resulting list is correct before comparing each value in order
    Assert::AreEqual(correctResult.Count(), searchedList.Count());
    for (int i = 0; i < searchedList.Count(); i++)
    {
        Assert::AreEqual(correctResult.At(i), searchedList.At(i));
    }
} // Searches a list containing a single item, that item should then be returned
```

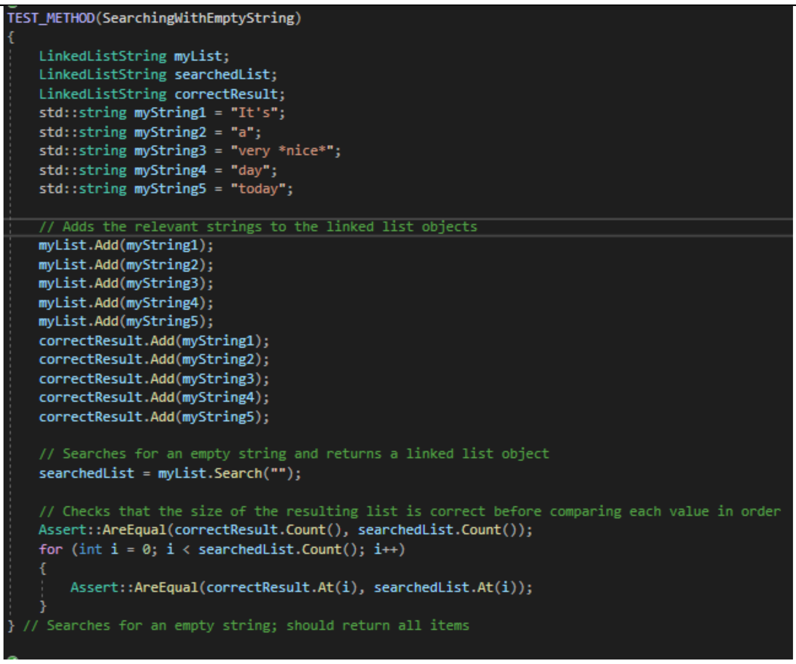
Figure 74 Search a Single Item List Test Code

</td></tr>
</table>

Test 15b is a black box unit test that makes sure that searching for a single item can return a correct list with the item inside of it. This is a basic but important test as it shows that there is at least basic functionality.

Table 53 - Test Number 15c

ID:	15c	Description:	Searching a list of five items with a complete match
Test Type:	Quality	Success Criteria:	The search returns only the correct item from the list
Number of Attempts:	1	Comments:	
List of Equipment/requirements	A list and five unique, valid and compatible items with only alphabetical characters in their names		
Setup Instructions	Create the list with the five items in it		
Failure Correction Procedure	Discuss with the software developer how best to implement a linear search		
Engineer/Technician	Samuel Harrison/Stephen Anderson		
Individual Results:	Pass		

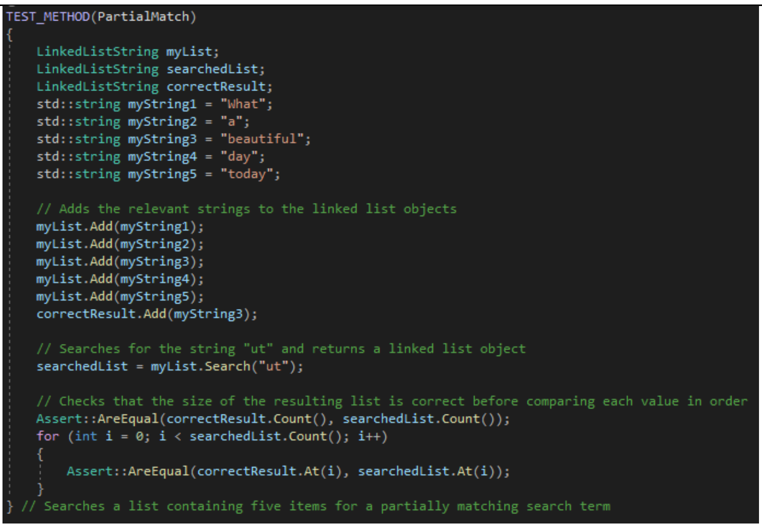
Screenshots	 *Figure 75 Search Complete Match Test Code*

```
TEST_METHOD(SearchingWithEmptyString)
{
    LinkedListString myList;
    LinkedListString searchedList;
    LinkedListString correctResult;
    std::string myString1 = "It's";
    std::string myString2 = "a";
    std::string myString3 = "very *nice*";
    std::string myString4 = "day";
    std::string myString5 = "today";

    // Adds the relevant strings to the linked list objects
    myList.Add(myString1);
    myList.Add(myString2);
    myList.Add(myString3);
    myList.Add(myString4);
    myList.Add(myString5);
    correctResult.Add(myString1);
    correctResult.Add(myString2);
    correctResult.Add(myString3);
    correctResult.Add(myString4);
    correctResult.Add(myString5);

    // Searches for an empty string and returns a linked list object
    searchedList = myList.Search("");

    // Checks that the size of the resulting list is correct before comparing each value in order
    Assert::AreEqual(correctResult.Count(), searchedList.Count());
    for (int i = 0; i < searchedList.Count(); i++)
    {
        Assert::AreEqual(correctResult.At(i), searchedList.At(i));
    }
} // Searches for an empty string; should return all items
```

Test 15c is a black box unit test designed to show that the search algorithm actually works properly and can return a limited list based on an input.

Table 54 - Test Number 15d

ID:	15d	Description:	Searching a list of five items with a partial match
Test Type:	Quality	Success Criteria:	The search returns only the correct item from the list
Number of Attempts:	1	Comments:	
List of Equipment/requirements	A list of five unique, valid and compatible items with only alphabetical characters in their names		
Setup Instructions	Create the list with the five items in it		
Failure Correction Procedure	Discuss with the software developer how best to implement a linear search		
Engineer/Technician	Samuel Harrison/Stephen Anderson		
Individual Results:	Pass		

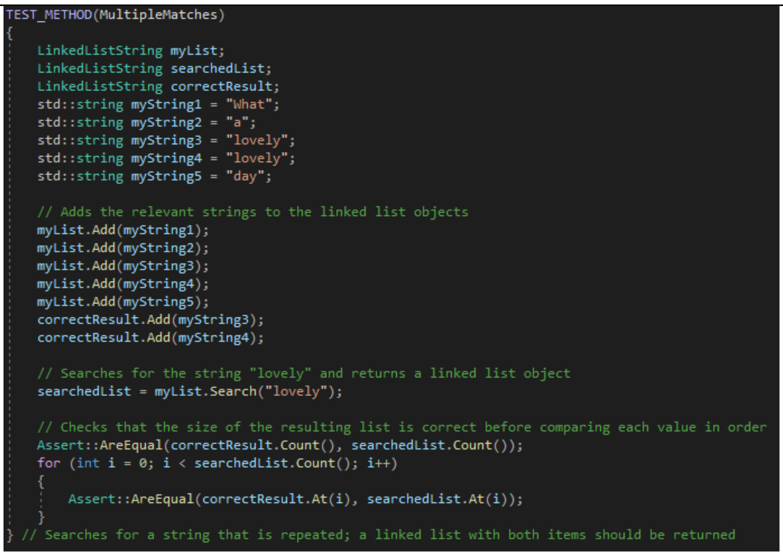
Screenshots	 *Figure 76 Search Partial Match Test Code*

```
TEST_METHOD(PartialMatch)
{
    LinkedListString myList;
    LinkedListString searchedList;
    LinkedListString correctResult;
    std::string myString1 = "What";
    std::string myString2 = "a";
    std::string myString3 = "beautiful";
    std::string myString4 = "day";
    std::string myString5 = "today";

    // Adds the relevant strings to the linked list objects
    myList.Add(myString1);
    myList.Add(myString2);
    myList.Add(myString3);
    myList.Add(myString4);
    myList.Add(myString5);
    correctResult.Add(myString3);

    // Searches for the string "ut" and returns a linked list object
    searchedList = myList.Search("ut");

    // Checks that the size of the resulting list is correct before comparing each value in order
    Assert::AreEqual(correctResult.Count(), searchedList.Count());
    for (int i = 0; i < searchedList.Count(); i++)
    {
        Assert::AreEqual(correctResult.At(i), searchedList.At(i));
    }
} // Searches a list containing five items for a partially matching search term
```

Test 15d is a black box unit test that shows the other main part of the search function – partial matches. They are important as they allow the user to only enter part of the search criteria which is very useful when there are lots of items in the list.

Table 55 - Test Number 15e

ID:	15e	Description:	Searching a list of five items that are similar
Test Type:	Quality	Success Criteria:	Returns multiple partially matched items from the search criteria
Number of Attempts:	1	Comments:	
List of Equipment/requirements	A list of three unique and two identical valid and compatible items with only alphabetical characters in their names		
Setup Instructions	Create the list with the five items in it		
Failure Correction Procedure	Discuss with the software developer how best to implement a linear search		
Engineer/Technician	Samuel Harrison/Stephen Anderson		
Individual Results:	Pass		

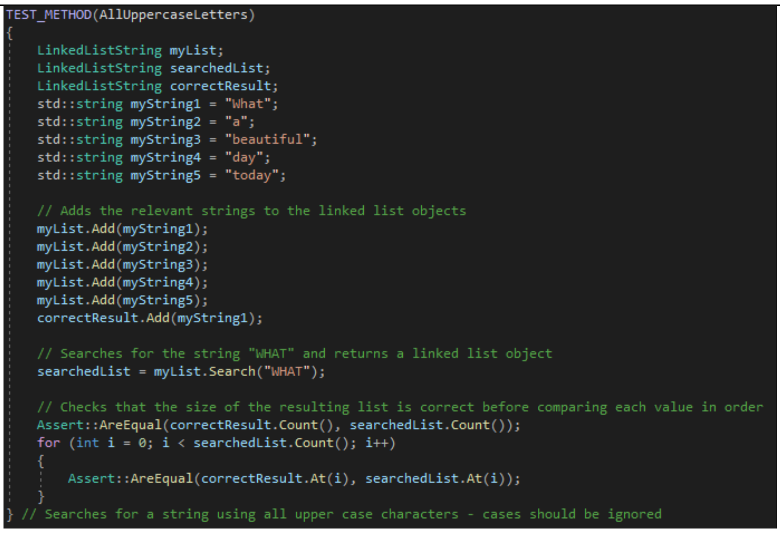
Screenshots	 *Figure 77 Search Multiple Matches Test Code*

```
TEST_METHOD(MultipleMatches)
{
    LinkedListString myList;
    LinkedListString searchedList;
    LinkedListString correctResult;
    std::string myString1 = "What";
    std::string myString2 = "a";
    std::string myString3 = "lovely";
    std::string myString4 = "lovely";
    std::string myString5 = "day";

    // Adds the relevant strings to the linked list objects
    myList.Add(myString1);
    myList.Add(myString2);
    myList.Add(myString3);
    myList.Add(myString4);
    myList.Add(myString5);
    correctResult.Add(myString3);
    correctResult.Add(myString4);

    // Searches for the string "lovely" and returns a linked list object
    searchedList = myList.Search("lovely");

    // Checks that the size of the resulting list is correct before comparing each value in order
    Assert::AreEqual(correctResult.Count(), searchedList.Count());
    for (int i = 0; i < searchedList.Count(); i++)
    {
        Assert::AreEqual(correctResult.At(i), searchedList.At(i));
    }
} // Searches for a string that is repeated; a linked list with both items should be returned
```

Test 15e is a black box unit test that specifically shows that multiple items can be returned from a search. This is important as there may be similarly named items and the user needs to select the correct one from the available list.

Table 56 - Test Number 15f

ID:	15f	Description:	Searching a list of five items with all uppercase characters
Test Type:	Quality	Success Criteria:	Should return the correct item from the list even though the cases are different
Number of Attempts:	1	Comments:	
List of Equipment/requirements	A list of five unique, valid and compatible items with only alphabetical characters in their names – names should alternate between upper and lower case		
Setup Instructions	Create the list with the five items in it		
Failure Correction Procedure	Discuss with the software developer how best to implement a linear search		
Engineer/Technician	Samuel Harrison/Stephen Anderson		
Individual Results:	Fail		

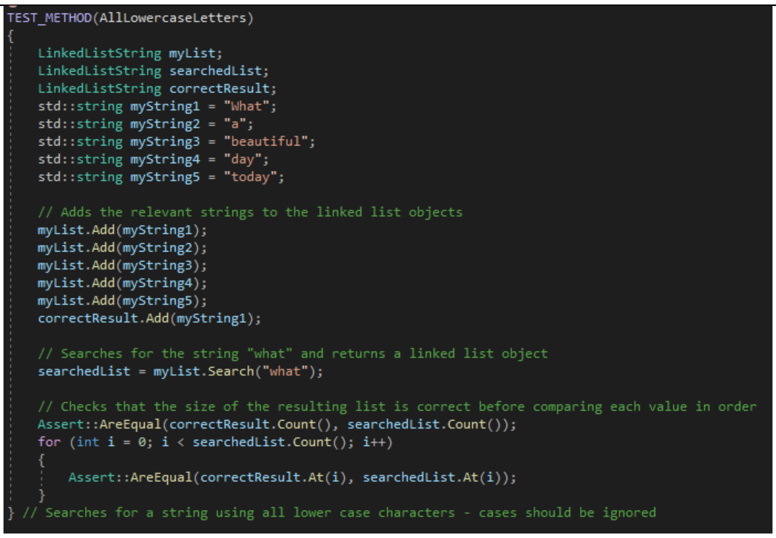
Screenshots	 *Figure 78 Search With an Upper Case Criterion Test Code*

```
TEST_METHOD(AllUppercaseLetters)
{
    LinkedListString myList;
    LinkedListString searchedList;
    LinkedListString correctResult;
    std::string myString1 = "What";
    std::string myString2 = "a";
    std::string myString3 = "beautiful";
    std::string myString4 = "day";
    std::string myString5 = "today";

    // Adds the relevant strings to the linked list objects
    myList.Add(myString1);
    myList.Add(myString2);
    myList.Add(myString3);
    myList.Add(myString4);
    myList.Add(myString5);
    correctResult.Add(myString1);

    // Searches for the string "WHAT" and returns a linked list object
    searchedList = myList.Search("WHAT");

    // Checks that the size of the resulting list is correct before comparing each value in order
    Assert::AreEqual(correctResult.Count(), searchedList.Count());
    for (int i = 0; i < searchedList.Count(); i++)
    {
        Assert::AreEqual(correctResult.At(i), searchedList.At(i));
    }
} // Searches for a string using all upper case characters - cases should be ignored
```

Test 15f is a black box unit test that is designed to show if the function is case sensitive by searching for a term that is in all upper case letters when the item that should be found only begins with an upper case letter.

Table 57 - Test Number 15g

ID:	15g	Description:	Searching a list of five items with all lowercase characters
Test Type:	Quality	Success Criteria:	Should return the correct item from the list even though the cases are different
Number of Attempts:	1	Comments:	
List of Equipment/requirements	A list of five unique, valid and compatible items with only alphabetical characters in their names – names should alternate between upper and lower case		
Setup Instructions	Create the list with the five items in it		
Failure Correction Procedure	Discuss with the software developer how best to implement a linear search		
Engineer/Technician	Samuel Harrison/Stephen Anderson		
Individual Results:	Fail		

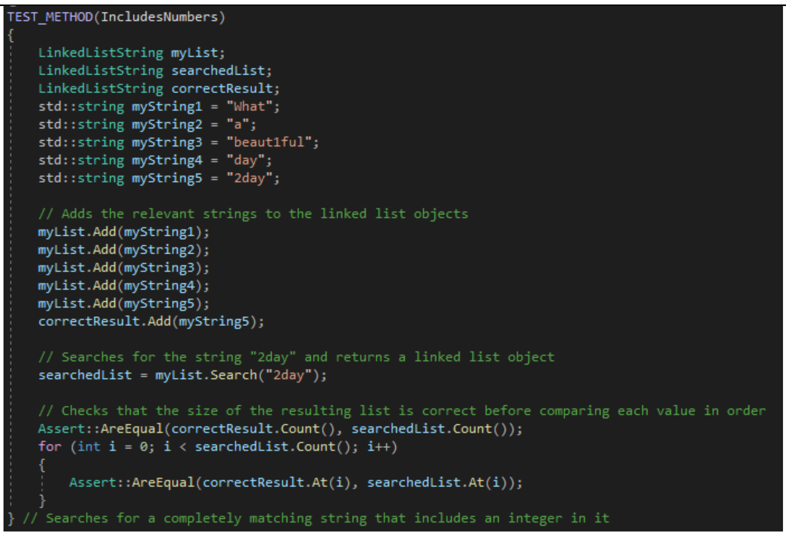
Screenshots	 *Figure 79 Search With a Lower Case Criterion Test Code*

```
TEST_METHOD(AllLowercaseLetters)
{
    LinkedListString myList;
    LinkedListString searchedList;
    LinkedListString correctResult;
    std::string myString1 = "What";
    std::string myString2 = "a";
    std::string myString3 = "beautiful";
    std::string myString4 = "day";
    std::string myString5 = "today";

    // Adds the relevant strings to the linked list objects
    myList.Add(myString1);
    myList.Add(myString2);
    myList.Add(myString3);
    myList.Add(myString4);
    myList.Add(myString5);
    correctResult.Add(myString1);

    // Searches for the string "what" and returns a linked list object
    searchedList = myList.Search("what");

    // Checks that the size of the resulting list is correct before comparing each value in order
    Assert::AreEqual(correctResult.Count(), searchedList.Count());
    for (int i = 0; i < searchedList.Count(); i++)
    {
        Assert::AreEqual(correctResult.At(i), searchedList.At(i));
    }
} // Searches for a string using all lower case characters - cases should be ignored
```

Test 15g is a black box unit test that is designed to show if the function is case sensitive by searching for a term that is in all lower case letters when the item that should be found actually begins with an upper case letter.

Table 58 - Test Number 15h

ID:	15h	Description:	Searching a list of five items when some of the names contain numbers
Test Type:	Quality	Success Criteria:	Should return the correct item without any issues
Number of Attempts:	1	Comments:	
List of Equipment/requirements	A list of five unique, valid and compatible items. Two of the items must have at least one numerical character in their names		
Setup Instructions	Create the list with the five items in it		
Failure Correction Procedure	Discuss with the software developer how best to implement a linear search		
Engineer/Technician	Samuel Harrison/Stephen Anderson		
Individual Results:	Pass		

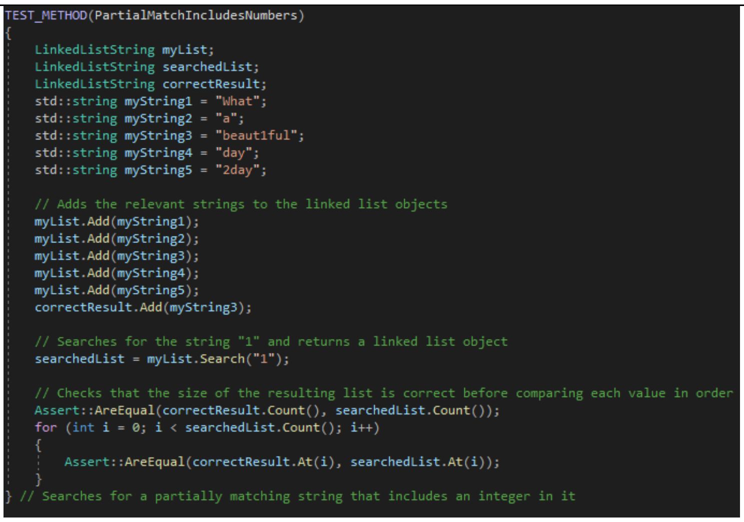
Screenshots	 *Figure 80 Search For a Complete Match With a Number Test Code*

```
TEST_METHOD(IncludesNumbers)
{
    LinkedListString myList;
    LinkedListString searchedList;
    LinkedListString correctResult;
    std::string myString1 = "What";
    std::string myString2 = "a";
    std::string myString3 = "beaut1ful";
    std::string myString4 = "day";
    std::string myString5 = "2day";

    // Adds the relevant strings to the linked list objects
    myList.Add(myString1);
    myList.Add(myString2);
    myList.Add(myString3);
    myList.Add(myString4);
    myList.Add(myString5);
    correctResult.Add(myString5);

    // Searches for the string "2day" and returns a linked list object
    searchedList = myList.Search("2day");

    // Checks that the size of the resulting list is correct before comparing each value in order
    Assert::AreEqual(correctResult.Count(), searchedList.Count());
    for (int i = 0; i < searchedList.Count(); i++)
    {
        Assert::AreEqual(correctResult.At(i), searchedList.At(i));
    }
} // Searches for a completely matching string that includes an integer in it
```

Test 15h is a black box unit test that targets how the search copes with non-alphabetical characters. This is useful as quite often file names (such as the images that are loaded in this piece of software) will have a number or a few numbers in them and so the function should be able to cope with this without throwing errors.

Table 59 - Test Number 15i

ID:	15i	Description:	Searching a list of five items when some of the names contain numbers with a partially matching search term
Test Type:	Quality	Success Criteria:	Should return the correct item without any issues
Number of Attempts:	1	Comments:	
List of Equipment/requirements	A list of five unique, valid and compatible items. Two of the items must have at least one numerical character in their names		
Setup Instructions	Create the list with the five items in it		
Failure Correction Procedure	Discuss with the software developer how best to implement a linear search		
Engineer/Technician	Samuel Harrison/Stephen Anderson		
Individual Results:	Pass		

<table>
<tr><td>Screenshots</td><td>
<pre>
TEST_METHOD(PartialMatchIncludesNumbers)
{
    LinkedListString myList;
    LinkedListString searchedList;
    LinkedListString correctResult;
    std::string myString1 = "What";
    std::string myString2 = "a";
    std::string myString3 = "beaut1ful";
    std::string myString4 = "day";
    std::string myString5 = "2day";

    // Adds the relevant strings to the linked list objects
    myList.Add(myString1);
    myList.Add(myString2);
    myList.Add(myString3);
    myList.Add(myString4);
    myList.Add(myString5);
    correctResult.Add(myString3);

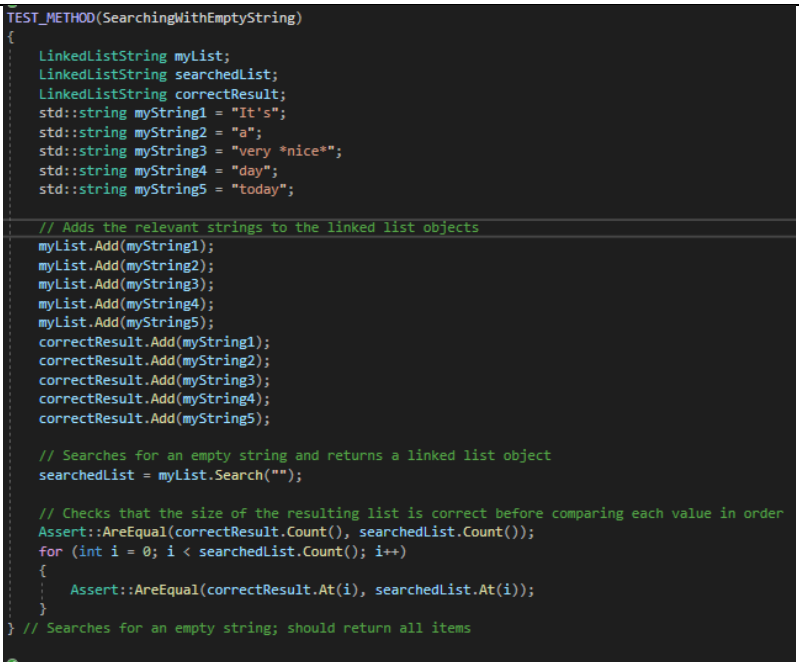
    // Searches for the string "1" and returns a linked list object
    searchedList = myList.Search("1");

    // Checks that the size of the resulting list is correct before comparing each value in order
    Assert::AreEqual(correctResult.Count(), searchedList.Count());
    for (int i = 0; i &lt; searchedList.Count(); i++)
    {
        Assert::AreEqual(correctResult.At(i), searchedList.At(i));
    }
} // Searches for a partially matching string that includes an integer in it
</pre>
<i>Figure 81 Search For a Partial Match With a Number Test Code</i></td></tr>
</table>

Test 15i is a black box unit test that is designed to see if the function can handle only being given a partial match that is a number. This is useful as if a file has been copied or there are multiple versions then often people (such as the user) will number them to help differentiate between them.

Table 60 - Test Number 15j

ID:	15j	Description:	Searching a list of five items using an empty string as a search
Test Type:	Quality	Success Criteria:	Should return every item in the list
Number of Attempts:	1	Comments:	
List of Equipment/requirements	A list of ten unique, valid and compatible items.		
Setup Instructions	Create the list with the five items in it		
Failure Correction Procedure	Discuss with the software developer how best to implement a linear search		
Engineer/Technician	Samuel Harrison/Stephen Anderson		
Individual Results:	Pass		

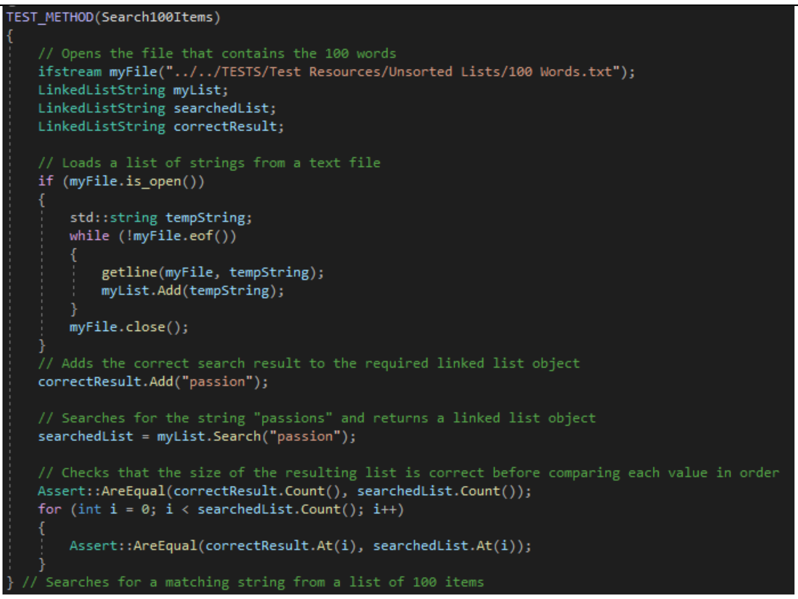
Screenshots	 *Figure 82 Search Using an Empty String Test Code*

```
TEST_METHOD(SearchingWithEmptyString)
{
    LinkedListString myList;
    LinkedListString searchedList;
    LinkedListString correctResult;
    std::string myString1 = "It's";
    std::string myString2 = "a";
    std::string myString3 = "very *nice*";
    std::string myString4 = "day";
    std::string myString5 = "today";

    // Adds the relevant strings to the linked list objects
    myList.Add(myString1);
    myList.Add(myString2);
    myList.Add(myString3);
    myList.Add(myString4);
    myList.Add(myString5);
    correctResult.Add(myString1);
    correctResult.Add(myString2);
    correctResult.Add(myString3);
    correctResult.Add(myString4);
    correctResult.Add(myString5);

    // Searches for an empty string and returns a linked list object
    searchedList = myList.Search("");

    // Checks that the size of the resulting list is correct before comparing each value in order
    Assert::AreEqual(correctResult.Count(), searchedList.Count());
    for (int i = 0; i < searchedList.Count(); i++)
    {
        Assert::AreEqual(correctResult.At(i), searchedList.At(i));
    }
} // Searches for an empty string; should return all items
```

Test 15j is a black box unit test that is designed because it is possible that the user will start searching for an item and then not enter a value. This could then be a problem if the function does not return every single item in the list as the user may then want to stop searching but the displayed list of items is still empty.

Table 61 - Test Number 15k

ID:	15k	Description:	Searching a list of one hundred items
Test Type:	Quality	Success Criteria:	Should return the correct item from the list
Number of Attempts:	1	Comments:	
List of Equipment/requirements	A list of one hundred unique, valid and compatible items.		
Setup Instructions	Create the list with the one hundred items in it		
Failure Correction Procedure	Discuss with the software developer how best to implement a linear search		
Engineer/Technician	Samuel Harrison/Stephen Anderson		
Individual Results:	Pass		

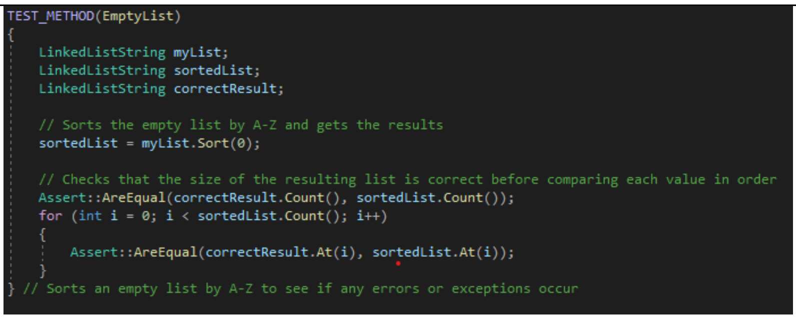
Screenshots	 *Figure 83 Search One Hundred Items Test Code*

```
TEST_METHOD(Search100Items)
{
    // Opens the file that contains the 100 words
    ifstream myFile("../../TESTS/Test Resources/Unsorted Lists/100 Words.txt");
    LinkedListString myList;
    LinkedListString searchedList;
    LinkedListString correctResult;

    // Loads a list of strings from a text file
    if (myFile.is_open())
    {
        std::string tempString;
        while (!myFile.eof())
        {
            getline(myFile, tempString);
            myList.Add(tempString);
        }
        myFile.close();
    }
    // Adds the correct search result to the required linked list object
    correctResult.Add("passion");

    // Searches for the string "passions" and returns a linked list object
    searchedList = myList.Search("passion");

    // Checks that the size of the resulting list is correct before comparing each value in order
    Assert::AreEqual(correctResult.Count(), searchedList.Count());
    for (int i = 0; i < searchedList.Count(); i++)
    {
        Assert::AreEqual(correctResult.At(i), searchedList.At(i));
    }
} // Searches for a matching string from a list of 100 items
```

Test 15k is a black box unit test designed to show that the function can cope even if there is a relatively large number of items, in this case one hundred of them.

Table 62 - Test Number 16a

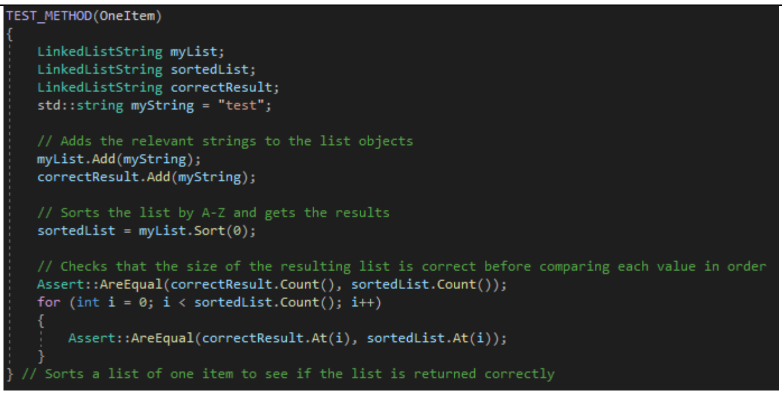
ID:	16a	Description:	Sorting an empty list A-Z
Test Type:	Quality	Success Criteria:	Should not produce any errors
Number of Attempts:	1	Comments:	Currently causes a c++ exception with exception code C0000005
List of Equipment/requirements	An empty list		
Setup Instructions	Create the list		
Failure Correction Procedure	Discuss with the software developer the best way to implement sorting of a linked list – most likely requiring a second list and iterating through them		
Engineer/Technician	Samuel Harrison/Stephen Anderson		
Individual Results:	Fail		

Screenshots	 *Figure 84 Sort Empty List Test Code*

```
TEST_METHOD(EmptyList)
{
    LinkedListString myList;
    LinkedListString sortedList;
    LinkedListString correctResult;

    // Sorts the empty list by A-Z and gets the results
    sortedList = myList.Sort(0);

    // Checks that the size of the resulting list is correct before comparing each value in order
    Assert::AreEqual(correctResult.Count(), sortedList.Count());
    for (int i = 0; i < sortedList.Count(); i++)
    {
        Assert::AreEqual(correctResult.At(i), sortedList.At(i));
    }
} // Sorts an empty list by A-Z to see if any errors or exceptions occur
```

Test 16a is black box unit test that is designed to see if sorting an empty list causes any errors to be created. This is important because if there is ever a time that the function is run in the background before a user has had a chance to even load something into one of the available lists.

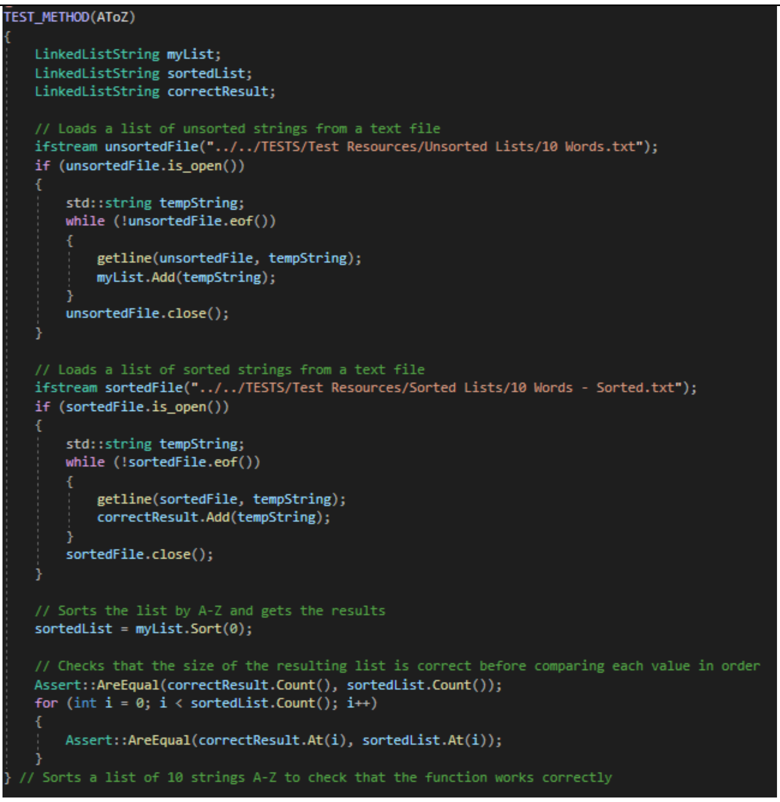
Table 63 - Test Number 16b

ID:	16b	Description:	Sorting a list with one item A-Z
Test Type:	Quality	Success Criteria:	Should just return the list as it was given
Number of Attempts:	1	Comments:	Currently causes a c++ exception with exception code C0000005
List of Equipment/requirements	A list and a single valid and compatible item that is named only with an alphabetical character		
Setup Instructions	Create the list with the item in it		
Failure Correction Procedure	Look at what is causing the exception and discuss with the software developer how to fix it		
Engineer/Technician	Samuel Harrison/Stephen Anderson		
Individual Results:	Fail		
Screenshots	 *Figure 85 Sort Single Item List Test Code*		

```
TEST_METHOD(OneItem)
{
    LinkedListString myList;
    LinkedListString sortedList;
    LinkedListString correctResult;
    std::string myString = "test";

    // Adds the relevant strings to the list objects
    myList.Add(myString);
    correctResult.Add(myString);

    // Sorts the list by A-Z and gets the results
    sortedList = myList.Sort(0);

    // Checks that the size of the resulting list is correct before comparing each value in order
    Assert::AreEqual(correctResult.Count(), sortedList.Count());
    for (int i = 0; i < sortedList.Count(); i++)
    {
        Assert::AreEqual(correctResult.At(i), sortedList.At(i));
    }
} // Sorts a list of one item to see if the list is returned correctly
```

Test 16b is a black box unit test that is designed to see that there is at least basic functionality and that the function returns a value.

Table 64 - Test Number 16c

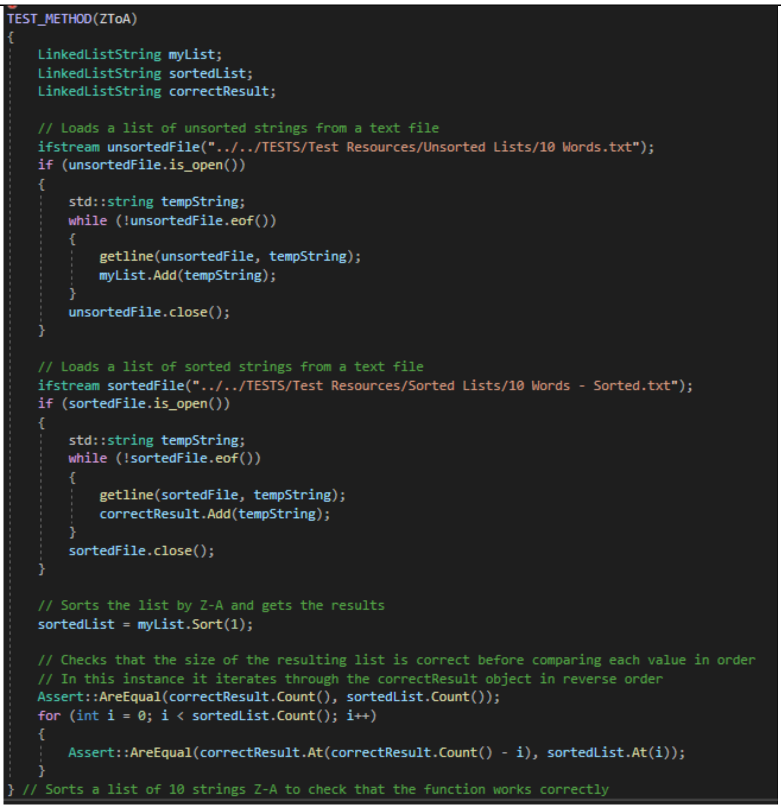
ID:	16c	Description:	Sorting a list of ten items A-Z
Test Type:	Quality	Success Criteria:	Should return the list in the correct order
Number of Attempts:	1	Comments:	
List of Equipment/requirements	A list and ten unique, valid and compatible items that are named only using alphabetical characters		
Setup Instructions	Create the list with the items – unordered – in it		
Failure Correction Procedure	Discuss with the software developer the best way to implement the sorting algorithm		
Engineer/Technician	Samuel Harrison/Stephen Anderson		
Individual Results:	Fail		
Screenshots	 *Figure 86 Sort Ten Items A-Z Test Code*		

```
TEST_METHOD(AToZ)
{
    LinkedListString myList;
    LinkedListString sortedList;
    LinkedListString correctResult;

    // Loads a list of unsorted strings from a text file
    ifstream unsortedFile("../../TESTS/Test Resources/Unsorted Lists/10 Words.txt");
    if (unsortedFile.is_open())
    {
        std::string tempString;
        while (!unsortedFile.eof())
        {
            getline(unsortedFile, tempString);
            myList.Add(tempString);
        }
        unsortedFile.close();
    }

    // Loads a list of sorted strings from a text file
    ifstream sortedFile("../../TESTS/Test Resources/Sorted Lists/10 Words - Sorted.txt");
    if (sortedFile.is_open())
    {
        std::string tempString;
        while (!sortedFile.eof())
        {
            getline(sortedFile, tempString);
            correctResult.Add(tempString);
        }
        sortedFile.close();
    }

    // Sorts the list by A-Z and gets the results
    sortedList = myList.Sort(0);

    // Checks that the size of the resulting list is correct before comparing each value in order
    Assert::AreEqual(correctResult.Count(), sortedList.Count());
    for (int i = 0; i < sortedList.Count(); i++)
    {
        Assert::AreEqual(correctResult.At(i), sortedList.At(i));
    }
} // Sorts a list of 10 strings A-Z to check that the function works correctly
```

Test 16c is a black box unit test that is designed to test whether or not the function works at all for sorting a list into alphabetical order. If it does not do this, then there is little functionality and therefore it is almost useless to use to return a value.

Table 65 - Test Number 16d

ID:	16d	Description:	Sorting a list of ten items Z-A

Test Type:	Quality	Success Criteria:	Should return the list in the correct order
Number of Attempts:	1	Comments:	
List of Equipment/requirements	A list and ten unique, valid and compatible items that are named only using alphabetical characters		
Setup Instructions	Create the list with the items – unordered – in it		
Failure Correction Procedure	Discuss with the software developer the best way to implement the sorting algorithm		
Engineer/Technician	Samuel Harrison/Stephen Anderson		
Individual Results:	Fail		
Screenshots			

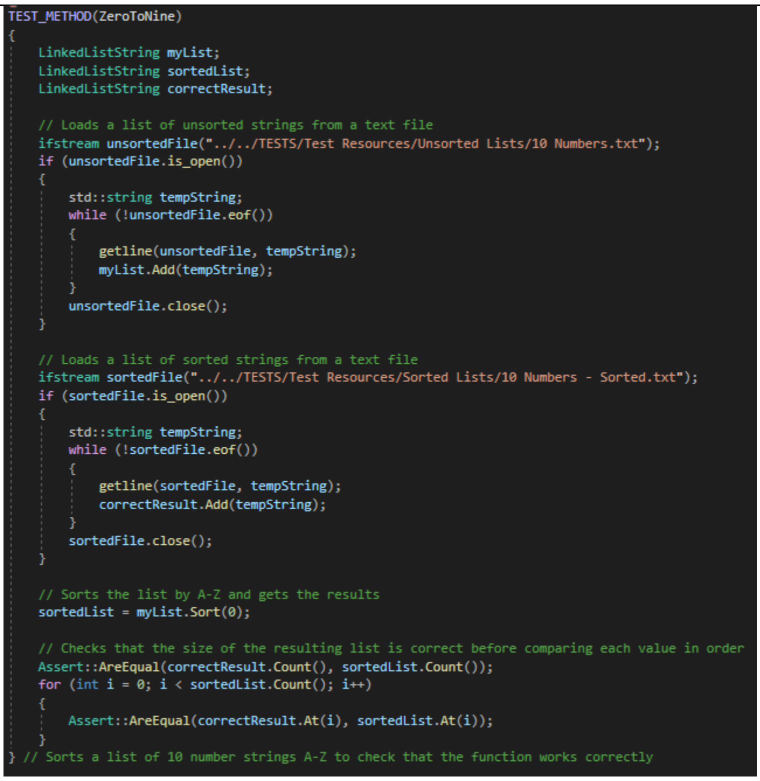
```
TEST_METHOD(ZToA)
{
    LinkedListString myList;
    LinkedListString sortedList;
    LinkedListString correctResult;

    // Loads a list of unsorted strings from a text file
    ifstream unsortedFile("../../TESTS/Test Resources/Unsorted Lists/10 Words.txt");
    if (unsortedFile.is_open())
    {
        std::string tempString;
        while (!unsortedFile.eof())
        {
            getline(unsortedFile, tempString);
            myList.Add(tempString);
        }
        unsortedFile.close();
    }

    // Loads a list of sorted strings from a text file
    ifstream sortedFile("../../TESTS/Test Resources/Sorted Lists/10 Words - Sorted.txt");
    if (sortedFile.is_open())
    {
        std::string tempString;
        while (!sortedFile.eof())
        {
            getline(sortedFile, tempString);
            correctResult.Add(tempString);
        }
        sortedFile.close();
    }

    // Sorts the list by Z-A and gets the results
    sortedList = myList.Sort(1);

    // Checks that the size of the resulting list is correct before comparing each value in order
    // In this instance it iterates through the correctResult object in reverse order
    Assert::AreEqual(correctResult.Count(), sortedList.Count());
    for (int i = 0; i < sortedList.Count(); i++)
    {
        Assert::AreEqual(correctResult.At(correctResult.Count() - i), sortedList.At(i));
    }
} // Sorts a list of 10 strings Z-A to check that the function works correctly
```

Figure 87 Sort Ten Items Z-A Test Code

Test 16d is a black box unit test that is designed to test whether or not the function works at all for sorting a list into reverse alphabetical order. If it does not do this, then there is little functionality and therefore it is almost useless to use to return a value.

Table 66 - Test Number 16e

ID:	16e	Description:	Sorting a list of ten numerically named items A-Z
Test Type:	Quality	Success Criteria:	Should return the list in the correct order

Number of Attempts:	1	Comments:	
List of Equipment/requirements	A list of ten unique, valid and compatible items that are named only using numeric values – the range does not matter		
Setup Instructions	Create the list with the items – unordered – in it		
Failure Correction Procedure	Discuss with the software developer the best way to implement the sorting algorithm		
Engineer/Technician	Samuel Harrison/Stephen Anderson		
Individual Results:	Fail		
Screenshots			

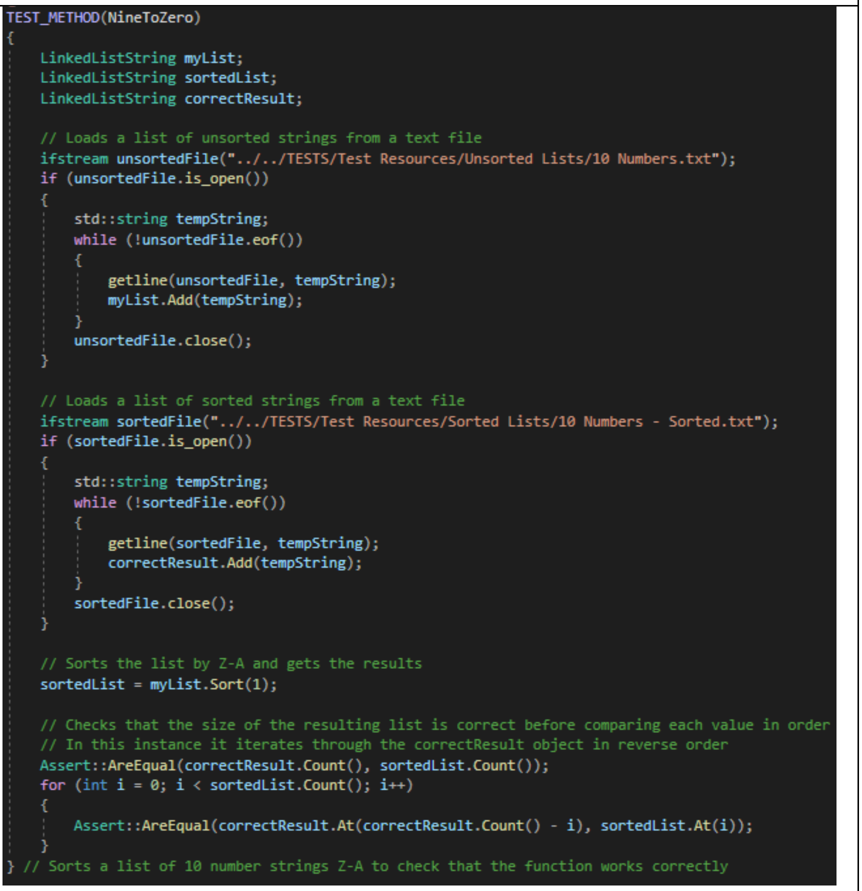
```
TEST_METHOD(ZeroToNine)
{
    LinkedListString myList;
    LinkedListString sortedList;
    LinkedListString correctResult;

    // Loads a list of unsorted strings from a text file
    ifstream unsortedFile("../../TESTS/Test Resources/Unsorted Lists/10 Numbers.txt");
    if (unsortedFile.is_open())
    {
        std::string tempString;
        while (!unsortedFile.eof())
        {
            getline(unsortedFile, tempString);
            myList.Add(tempString);
        }
        unsortedFile.close();
    }

    // Loads a list of sorted strings from a text file
    ifstream sortedFile("../../TESTS/Test Resources/Sorted Lists/10 Numbers - Sorted.txt");
    if (sortedFile.is_open())
    {
        std::string tempString;
        while (!sortedFile.eof())
        {
            getline(sortedFile, tempString);
            correctResult.Add(tempString);
        }
        sortedFile.close();
    }

    // Sorts the list by A-Z and gets the results
    sortedList = myList.Sort(0);

    // Checks that the size of the resulting list is correct before comparing each value in order
    Assert::AreEqual(correctResult.Count(), sortedList.Count());
    for (int i = 0; i < sortedList.Count(); i++)
    {
        Assert::AreEqual(correctResult.At(i), sortedList.At(i));
    }
} // Sorts a list of 10 number strings A-Z to check that the function works correctly
```

Figure 88 Sort 10 Numerical Items A-Z

Test 16e is a black box unit test that is designed to test whether or not the function properly for sorting numerical values. This is very useful as often names will be entirely made of numbers and it should still be able to sort them – in this case from smallest to largest.

Table 67 - Test Number 16f

ID:	16f	Description:	Sorting a list of ten numerically named items Z-A
Test Type:	Quality	Success Criteria:	Should return the list in the correct order
Number of Attempts:	1	Comments:	
List of Equipment/requirements	A list of ten unique, valid and compatible items that are named only using numeric values – the range does not matter		

<table>
<tr><td>Setup Instructions</td><td>Create the list with the items – unordered – in it</td></tr>
<tr><td>Failure Correction Procedure</td><td>Discuss with the software developer the best way to implement the sorting algorithm</td></tr>
<tr><td>Engineer/Technician</td><td>Samuel Harrison/Stephen Anderson</td></tr>
<tr><td>Individual Results:</td><td>Fail</td></tr>
<tr><td>Screenshots</td><td>
<pre>
TEST_METHOD(NineToZero)
{
    LinkedListString myList;
    LinkedListString sortedList;
    LinkedListString correctResult;

    // Loads a list of unsorted strings from a text file
    ifstream unsortedFile("../../TESTS/Test Resources/Unsorted Lists/10 Numbers.txt");
    if (unsortedFile.is_open())
    {
        std::string tempString;
        while (!unsortedFile.eof())
        {
            getline(unsortedFile, tempString);
            myList.Add(tempString);
        }
        unsortedFile.close();
    }

    // Loads a list of sorted strings from a text file
    ifstream sortedFile("../../TESTS/Test Resources/Sorted Lists/10 Numbers - Sorted.txt");
    if (sortedFile.is_open())
    {
        std::string tempString;
        while (!sortedFile.eof())
        {
            getline(sortedFile, tempString);
            correctResult.Add(tempString);
        }
        sortedFile.close();
    }

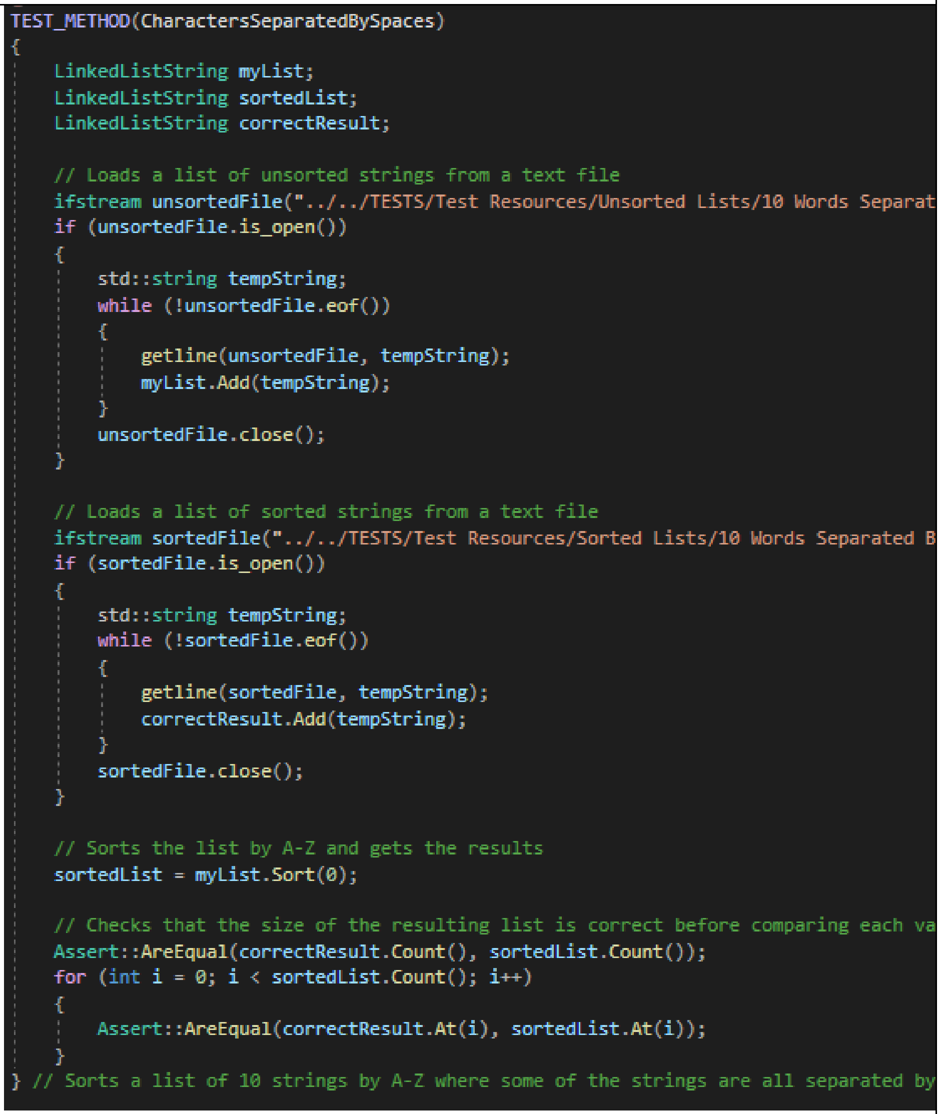
    // Sorts the list by Z-A and gets the results
    sortedList = myList.Sort(1);

    // Checks that the size of the resulting list is correct before comparing each value in order
    // In this instance it iterates through the correctResult object in reverse order
    Assert::AreEqual(correctResult.Count(), sortedList.Count());
    for (int i = 0; i &lt; sortedList.Count(); i++)
    {
        Assert::AreEqual(correctResult.At(correctResult.Count() - i), sortedList.At(i));
    }
} // Sorts a list of 10 number strings Z-A to check that the function works correctly
</pre>
<i>Figure 89 Sort Ten Numerical Items Z-A</i></td></tr>
</table>

Test 16f is a black box unit test that is designed to test whether or not the function properly for sorting numerical values. This is very useful as often names will be entirely made of numbers and it should still be able to sort them – in this case largest to smallest.

Table 68 - Test Number 16g

ID:	16g	Description:	Sorting a list of items with spaces in their names A-Z
Test Type:	Quality	Success Criteria:	Should return the list in the correct order, effectively ignoring the spaces
Number of Attempts:	1	Comments:	

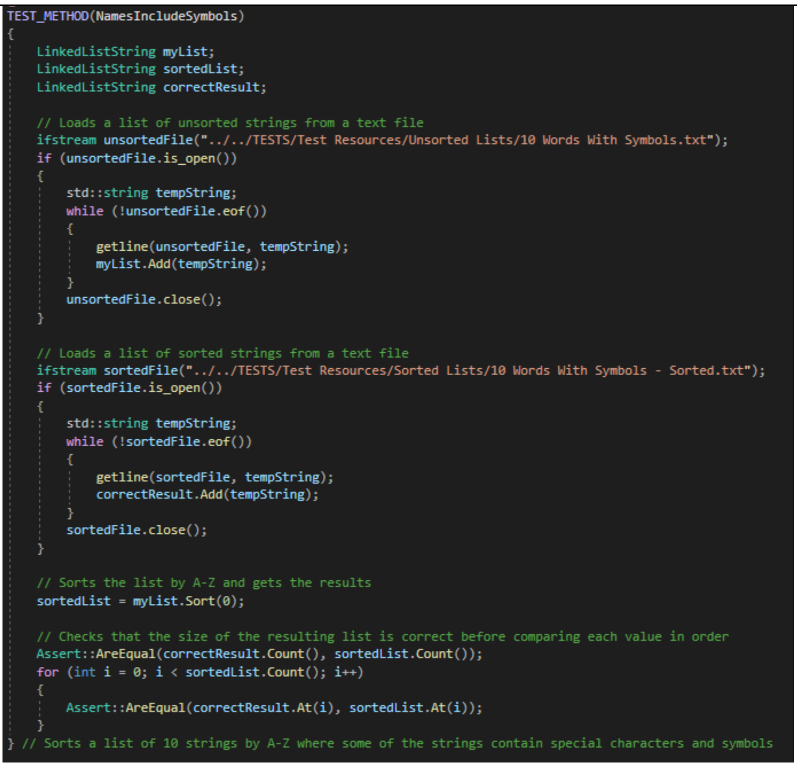
List of Equipment/requirements	A list of five item names that have every character separated by a space, with another five item names containing only alphabetical characters
Setup Instructions	Create the list with the items – unordered – in it
Failure Correction Procedure	Discuss with the software developer the best way to implement the sorting algorithm
Engineer/Technician	Samuel Harrison/Stephen Anderson
Individual Results:	Fail
Screenshots	

```
TEST_METHOD(CharactersSeparatedBySpaces)
{
    LinkedListString myList;
    LinkedListString sortedList;
    LinkedListString correctResult;

    // Loads a list of unsorted strings from a text file
    ifstream unsortedFile("../../TESTS/Test Resources/Unsorted Lists/10 Words Separat
    if (unsortedFile.is_open())
    {
        std::string tempString;
        while (!unsortedFile.eof())
        {
            getline(unsortedFile, tempString);
            myList.Add(tempString);
        }
        unsortedFile.close();
    }

    // Loads a list of sorted strings from a text file
    ifstream sortedFile("../../TESTS/Test Resources/Sorted Lists/10 Words Separated B
    if (sortedFile.is_open())
    {
        std::string tempString;
        while (!sortedFile.eof())
        {
            getline(sortedFile, tempString);
            correctResult.Add(tempString);
        }
        sortedFile.close();
    }

    // Sorts the list by A-Z and gets the results
    sortedList = myList.Sort(0);

    // Checks that the size of the resulting list is correct before comparing each va
    Assert::AreEqual(correctResult.Count(), sortedList.Count());
    for (int i = 0; i < sortedList.Count(); i++)
    {
        Assert::AreEqual(correctResult.At(i), sortedList.At(i));
    }
} // Sorts a list of 10 strings by A-Z where some of the strings are all separated by
```

Figure 90 Sort Ten Items Separate By Spaces

Test 16g is a black box unit test that is designed to see if the function can handle an item having spaces in it – especially useful as again a lot of file names which is where this will see the most use in the software are going to have spaces in them.

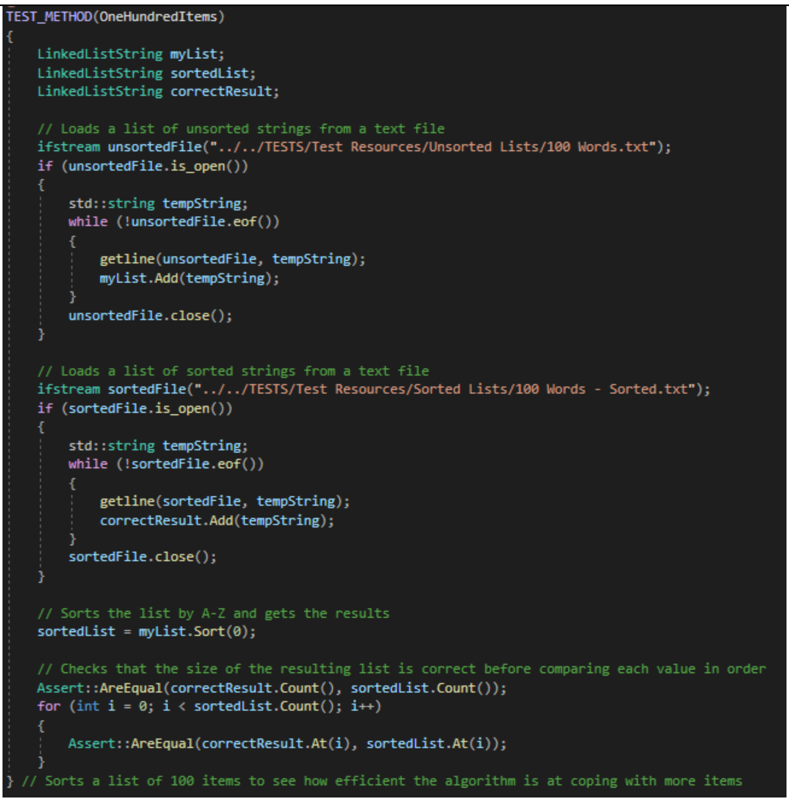
Table 69 - Test Number 16h

ID:	16h	Description:	Sorting a list of items with special characters in their names A-Z
Test Type:	Quality	Success Criteria:	Should return the list in the correct alphabetical order, ignoring the special characters
Number of Attempts:	1	Comments:	
List of Equipment/requirements	A list of five item names that have at least one non-alphanumeric character, with another five item names containing only alphabetical characters		
Setup Instructions	Create the list with the items – unordered – in it		
Failure Correction Procedure	Discuss with the software developer the best way to implement the sorting algorithm		
Engineer/Technician	Samuel Harrison/Stephen Anderson		
Individual Results:	Fail		
Screenshots	 *Figure 91 Sort Ten Items That Include Special Characters Test Code*		

```cpp
TEST_METHOD(NamesIncludeSymbols)
{
    LinkedListString myList;
    LinkedListString sortedList;
    LinkedListString correctResult;

    // Loads a list of unsorted strings from a text file
    ifstream unsortedFile("../../TESTS/Test Resources/Unsorted Lists/10 Words With Symbols.txt");
    if (unsortedFile.is_open())
    {
        std::string tempString;
        while (!unsortedFile.eof())
        {
            getline(unsortedFile, tempString);
            myList.Add(tempString);
        }
        unsortedFile.close();
    }

    // Loads a list of sorted strings from a text file
    ifstream sortedFile("../../TESTS/Test Resources/Sorted Lists/10 Words With Symbols - Sorted.txt");
    if (sortedFile.is_open())
    {
        std::string tempString;
        while (!sortedFile.eof())
        {
            getline(sortedFile, tempString);
            correctResult.Add(tempString);
        }
        sortedFile.close();
    }

    // Sorts the list by A-Z and gets the results
    sortedList = myList.Sort(0);

    // Checks that the size of the resulting list is correct before comparing each value in order
    Assert::AreEqual(correctResult.Count(), sortedList.Count());
    for (int i = 0; i < sortedList.Count(); i++)
    {
        Assert::AreEqual(correctResult.At(i), sortedList.At(i));
    }
} // Sorts a list of 10 strings by A-Z where some of the strings contain special characters and symbols
```

Test 16h is a black box unit test that is designed to see if the function can handle an item having special characters in it – this is primarily because a lot of file names will often have characters other than the standard alphanumeric ones.

Table 70 - Test Number 16i

ID:	16i	Description:	Sorting a list of one hundred items A-Z
Test Type:	Quality	Success Criteria:	Should return the list in the correct order within a reasonable amount of time (less than 2000ms is ideal, but less than 5000ms is acceptable)
Number of Attempts:	1	Comments:	
List of Equipment/requirements	A list of one hundred items that have only alphabetical characters in their names		
Setup Instructions	Create the list with the items – unordered – in it		
Failure Correction Procedure	Discuss with the software developer the best way to implement the sorting algorithm		
Engineer/Technician	Samuel Harrison/Stephen Anderson		
Individual Results:	Fail		
Screenshots	 *Figure 92 Sort One Hundred Items A-Z Test Code*		

```
TEST_METHOD(OneHundredItems)
{
    LinkedListString myList;
    LinkedListString sortedList;
    LinkedListString correctResult;

    // Loads a list of unsorted strings from a text file
    ifstream unsortedFile("../../TESTS/Test Resources/Unsorted Lists/100 Words.txt");
    if (unsortedFile.is_open())
    {
        std::string tempString;
        while (!unsortedFile.eof())
        {
            getline(unsortedFile, tempString);
            myList.Add(tempString);
        }
        unsortedFile.close();
    }

    // Loads a list of sorted strings from a text file
    ifstream sortedFile("../../TESTS/Test Resources/Sorted Lists/100 Words - Sorted.txt");
    if (sortedFile.is_open())
    {
        std::string tempString;
        while (!sortedFile.eof())
        {
            getline(sortedFile, tempString);
            correctResult.Add(tempString);
        }
        sortedFile.close();
    }

    // Sorts the list by A-Z and gets the results
    sortedList = myList.Sort(0);

    // Checks that the size of the resulting list is correct before comparing each value in order
    Assert::AreEqual(correctResult.Count(), sortedList.Count());
    for (int i = 0; i < sortedList.Count(); i++)
    {
        Assert::AreEqual(correctResult.At(i), sortedList.At(i));
    }
} // Sorts a list of 100 items to see how efficient the algorithm is at coping with more items
```

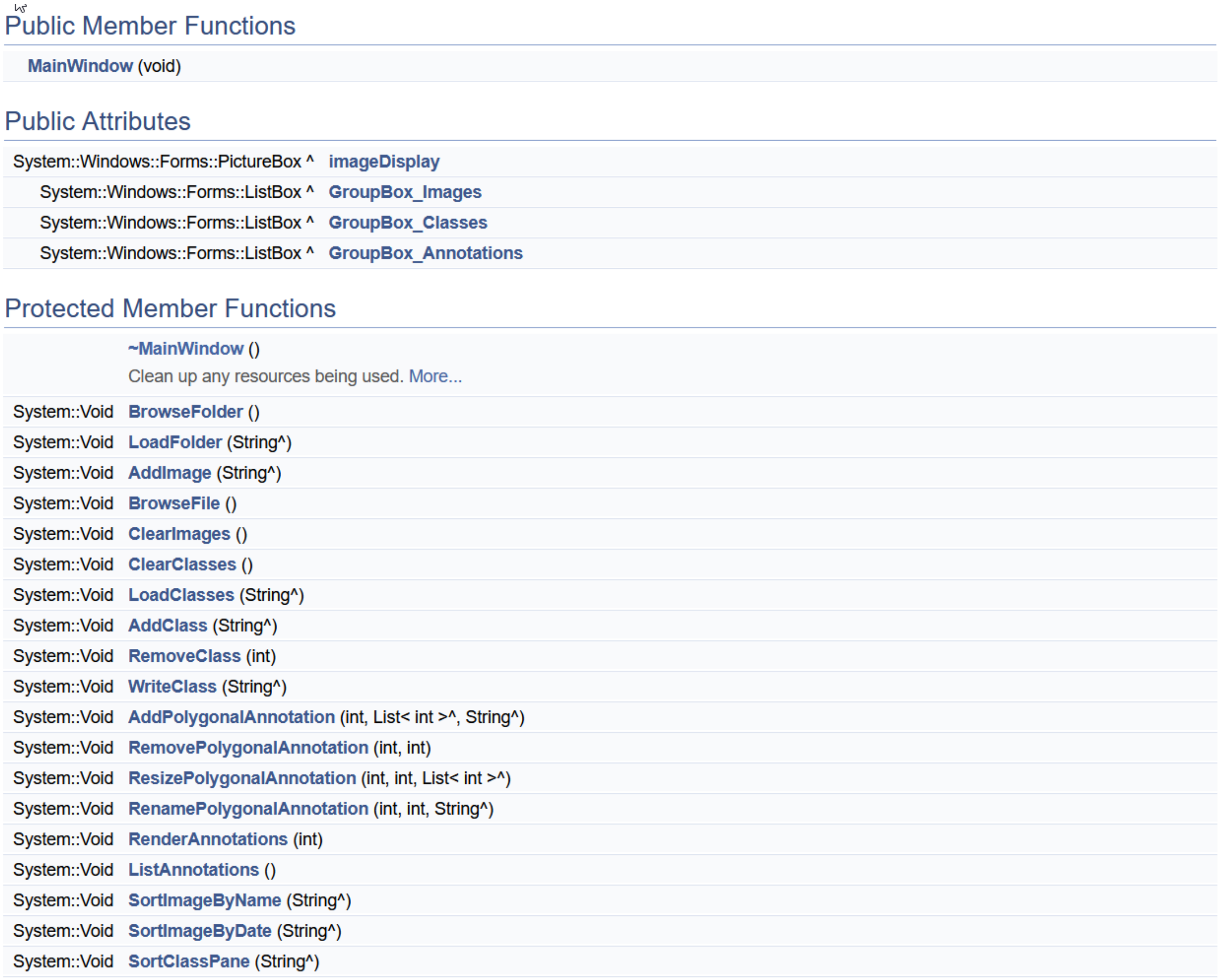
Test 16i is a black box unit test that is designed to push the sorting to its limit and see how efficient the algorithm is. If it takes too long, then the algorithm is likely not appropriate for the use case of a quick button to let the user choose which order to have items in a list displayed.

Table 71 - Test Number 17

ID:	17	Description:	Autosave
Test Type:	Quality	Success Criteria:	The software should automatically save any work after a 60 second interval (once the user has selected or created a file to save to)
Number of Attempts:	1	Comments:	
List of Equipment/requirements	At least one valid image, .names file and		
Setup Instructions	Create the list with the items – unordered – in it		
Failure Correction Procedure	Discuss with the software developer the best way to implement a timer on a separate thread		
Engineer/Technician	Samuel Harrison/Stephen Anderson		
Individual Results:			
Screenshots			

Appendix 3 – Reference Manual

The reference manual was generated by using Doxygen, all classes and functions will be documented within the reference manual. The full reference manual can be found in the /DOCUMENTATION/html folder by opening index.html.



Public Member Functions

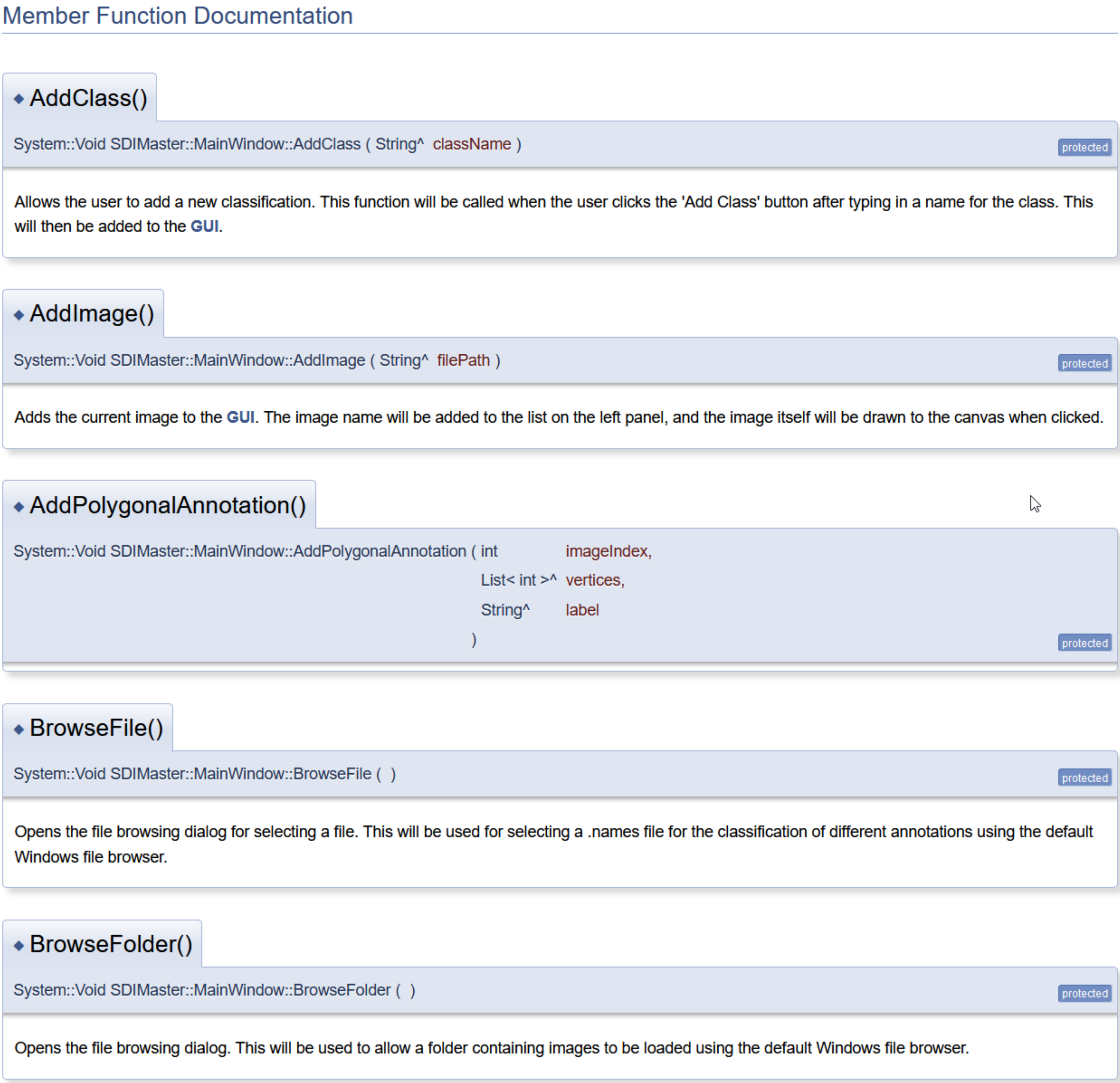
	MainWindow (void)

Public Attributes

System::Windows::Forms::PictureBox ^	**imageDisplay**
System::Windows::Forms::ListBox ^	**GroupBox_Images**
System::Windows::Forms::ListBox ^	**GroupBox_Classes**
System::Windows::Forms::ListBox ^	**GroupBox_Annotations**

Protected Member Functions

	~MainWindow () Clean up any resources being used. More...
System::Void	**BrowseFolder** ()
System::Void	**LoadFolder** (String^)
System::Void	**AddImage** (String^)
System::Void	**BrowseFile** ()
System::Void	**ClearImages** ()
System::Void	**ClearClasses** ()
System::Void	**LoadClasses** (String^)
System::Void	**AddClass** (String^)
System::Void	**RemoveClass** (int)
System::Void	**WriteClass** (String^)
System::Void	**AddPolygonalAnnotation** (int, List< int >^, String^)
System::Void	**RemovePolygonalAnnotation** (int, int)
System::Void	**ResizePolygonalAnnotation** (int, int, List< int >^)
System::Void	**RenamePolygonalAnnotation** (int, int, String^)
System::Void	**RenderAnnotations** (int)
System::Void	**ListAnnotations** ()
System::Void	**SortImageByName** (String^)
System::Void	**SortImageByDate** (String^)
System::Void	**SortClassPane** (String^)



Member Function Documentation

◆ AddClass()

```
System::Void SDIMaster::MainWindow::AddClass ( String^  className )        protected
```

Allows the user to add a new classification. This function will be called when the user clicks the 'Add Class' button after typing in a name for the class. This will then be added to the **GUI**.

◆ AddImage()

```
System::Void SDIMaster::MainWindow::AddImage ( String^  filePath )        protected
```

Adds the current image to the **GUI**. The image name will be added to the list on the left panel, and the image itself will be drawn to the canvas when clicked.

◆ AddPolygonalAnnotation()

```
System::Void SDIMaster::MainWindow::AddPolygonalAnnotation ( int          imageIndex,
                                                             List< int >^ vertices,
                                                             String^      label
                                                           )                        protected
```

◆ BrowseFile()

```
System::Void SDIMaster::MainWindow::BrowseFile (  )        protected
```

Opens the file browsing dialog for selecting a file. This will be used for selecting a .names file for the classification of different annotations using the default Windows file browser.

◆ BrowseFolder()

```
System::Void SDIMaster::MainWindow::BrowseFolder (  )        protected
```

Opens the file browsing dialog. This will be used to allow a folder containing images to be loaded using the default Windows file browser.

◆ ClearClasses()

System::Void SDIMaster::MainWindow::ClearClasses ( ) protected

Clears the classes currently loaded from memory.

◆ ClearImages()

System::Void SDIMaster::MainWindow::ClearImages ( ) protected

Clears the images currently loaded from memory.

◆ ListAnnotations()

System::Void SDIMaster::MainWindow::ListAnnotations ( ) protected

List all current annotations in the **GUI**. Shows all annotations, whether loaded from a file or created in the current session.

◆ LoadClasses()

System::Void SDIMaster::MainWindow::LoadClasses (String^ filePath) protected

Reads through the .names file and loads them to the panel. This utilises StreamReader for opening and loading the .names file to display all relevant classes found into the user interface.

◆ LoadFolder()

System::Void SDIMaster::MainWindow::LoadFolder (String^ folderPath) protected

Loads the contents of the selected folder into the **GUI**. In the case of this application, the images to be annotated will be loaded into the **GUI**, and can in theory load an unlimited amount of images, depending on the user's installed memory.

◆ RemoveClass()

```
System::Void SDIMaster::MainWindow::RemoveClass ( int  classIndex )
```

protected

Removes a class from the **GUI** and .names file. When the user clicks on a class in the **GUI**, they can then click the 'Remove Class' button removing it from both the **GUI** panel and the .names file.

◆ RemovePolygonalAnnotation()

```
System::Void SDIMaster::MainWindow::RemovePolygonalAnnotation ( int  imageIndex,
                                                                int  annotationIndex
                                                              )
```

protected

◆ RenamePolygonalAnnotation()

```
System::Void SDIMaster::MainWindow::RenamePolygonalAnnotation ( int      imageIndex,
                                                                int      annotationIndex,
                                                                String^  name
                                                              )
```

protected

◆ RenderAnnotations()

```
System::Void SDIMaster::MainWindow::RenderAnnotations ( int  imageIndex )
```

protected

Draws all current annotations to the screen. Adds the annotations to the current image in the canvas including both annotations drawn in the current session and loaded from the .names file.

◆ ResizePolygonalAnnotation()

```
System::Void SDIMaster::MainWindow::ResizePolygonalAnnotation ( int          imageIndex,
                                                                int          annotationIndex,
                                                                List< int >^ vertices
                                                              )
```

protected

◆ SortClassPane()

System::Void SDIMaster::MainWindow::SortClassPane (String^ order) protected

Sorts the class pane by name. Uses the implementation found in **SortAndSearch.cpp**, allowing the images to be sorted ascending or descending by name.

◆ SortImageByDate()

System::Void SDIMaster::MainWindow::SortImageByDate (String^ order) protected

Sorts the image by date. Uses the implementation found in **SortAndSearch.cpp**, allowing the images to be sorted ascending or descending by date.

◆ SortImageByName()

System::Void SDIMaster::MainWindow::SortImageByName (String^ order) protected

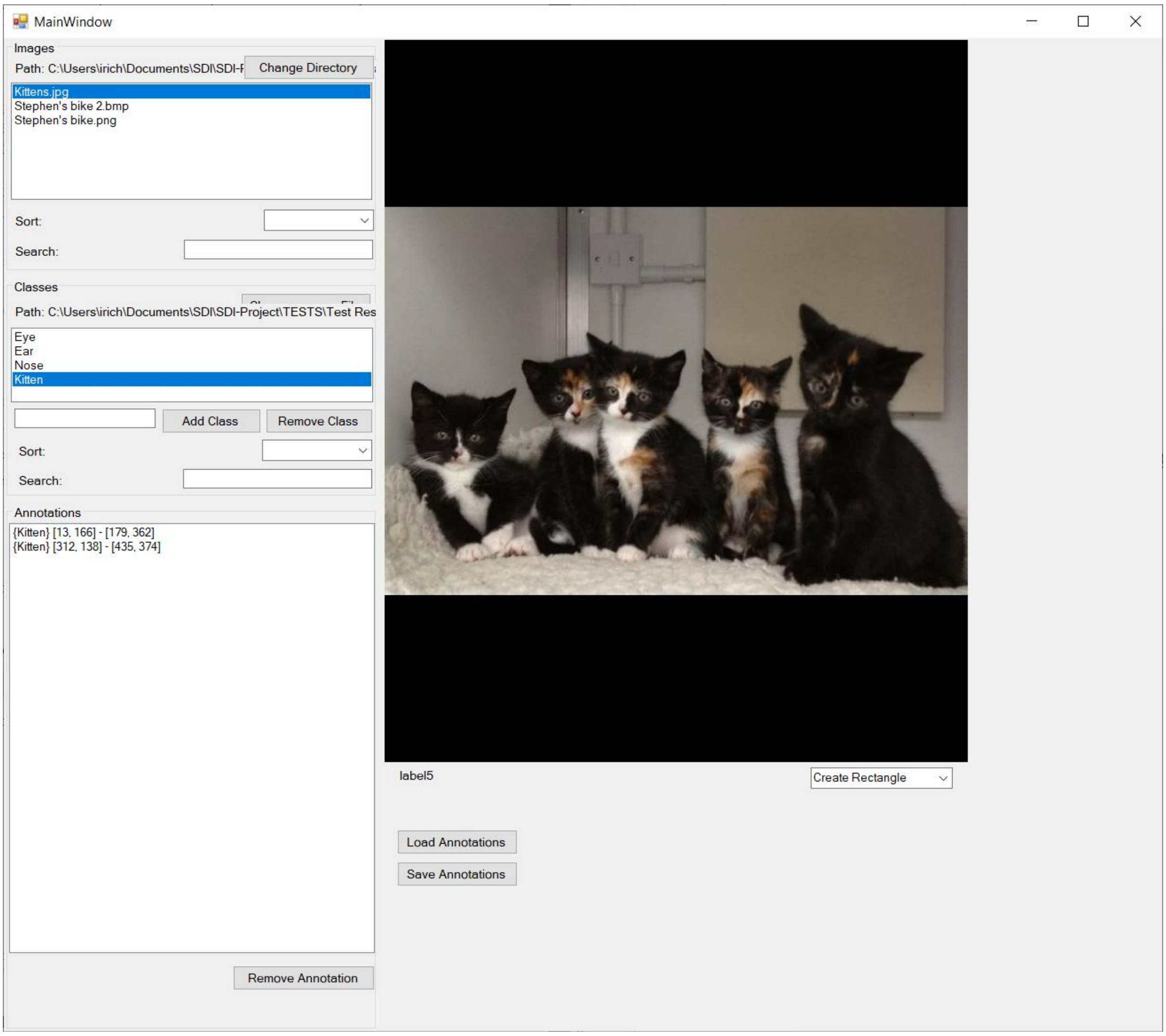
Sorts the images by name. Uses the implementation found in **SortAndSearch.cpp**, allowing the images to be sorted ascending or descending by name.

◆ WriteClass()

System::Void SDIMaster::MainWindow::WriteClass (String^ className) protected

Writes the previously added class to the .names file. Makes use of file handling to do so, and will add the class to a new line in the .names file.

Appendix 4 – User Interface



Path: C:\Users\irich\Documents\SDI\SDI-F

Kittens.jpg
Stephen's bike 2.bmp
Stephen's bike.png

Sort:

Search:

Path: C:\Users\irich\Documents\SDI\SDI-Project\TESTS\Test Res

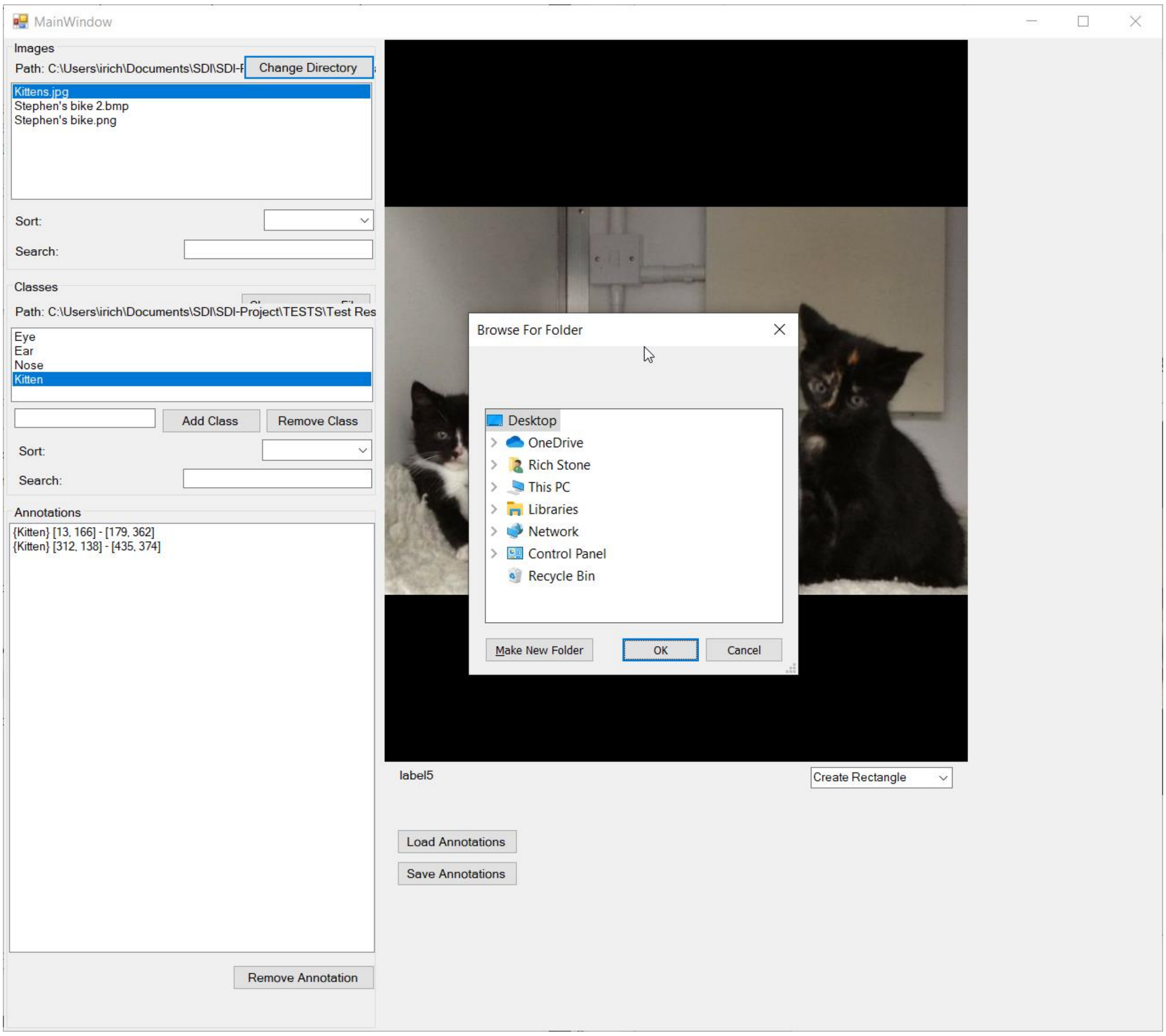
Eye
Ear
Nose
Kitten

Sort:

Search:

{Kitten} [13, 166] - [179, 362]
{Kitten} [312, 138] - [435, 374]

Figure 93 - Typical User Interface



Path: C:\Users\irich\Documents\SDI\SDI-F

Kittens.jpg
Stephen's bike 2.bmp
Stephen's bike.png

Path: C:\Users\irich\Documents\SDI\SDI-Project\TESTS\Test Res

Eye
Ear
Nose
Kitten

{Kitten} [13, 166] - [179, 362]
{Kitten} [312, 138] - [435, 374]

Figure 94 - Including File Browsing

Appendix 4 - Group Meetings

Meeting 1 – 13/1/20 Attended: All

-Richard assigned design aspects to each member
Richard: Use case, component
Stephen: Sequence diagrams
Thomas: State, Class
Samuel: UI Design

Meeting 2 – 21/1/20 Attended: All

-Reviewed all diagrams up to this stage, decided to create the UI earlier in visual studio
-Assigned each member to write a description of their diagrams, and for Thomas Harrison to combine them and finish the other report aspects

Meeting 3 – 3/2/20 Attended: All
-Reviewed the final design document, all agreed upon
-Samuel demonstrated the current GUI that they created in visual studio
-Decided to wait to see what needed to be changed from the design to delegate jobs for implementation

Meeting 4 – 10/2/20 Attended: Richard Stone, Stephen Anderson, Samuel Harrison
-Began work on implementing code into the program and linking to the GUI -Samuel created implementation to load an image into the UI
-Richard decided that he will do the coding style guide, Thomas would do the contribution guide, and Stephen would work on creating the test plan.
-Time constraints were emphasized

Meeting 5 - 17/2/20 Attended: All
-Samuel showed that she was able to draw multiple images into the UI
-Showed that they could be grouped together with labels -Decided to begin work on creating full annotations
-Richard, Stephen, and Thomas said that they had all made progress on their respective documents from last meeting

Meeting 6 – 23/2/20 (Discord meeting) Attended: All
-Richard finished and showed coding style guide, had been agreed upon with Samuel but some code needed adapting
-Stephen produced test plan but stressed that boost had difficulty working with both the C++ and C# aspects of CLI, would continue to try and get it working. -Agreed that Richard would create the video for the next deliverable.

Meeting 7 – 3/2/20 Attended: All
-Discussion following the most recent deliverable, decided that Thomas Harrison would need to work further on the contribution guide. -Samuel said they would continue working on the development, while Stephen would write tests when this had been delivered.

Meeting 8 – 30/3/20 (Emergency Discord meeting) Attended: All
-Discussion following most recent deliverable.
-Samuel Harrison would commit the code they have to the git regardless of the state of the code.
-Stephen Anderson would take a shared role with Samuel Harrison and become more heavily involved in development. Stephen Anderson and Richard Stone will then write the test cases together. Thomas Harrison will help where needed, in either development or testing.
-This will be completed within the next few days, Thursday at the latest.

Appendix 5 - Plagiarism Declaration

This report and the software it documents is the result of my own work, other contributing group members are acknowledged. Any contributions to the work by third parties, other than tutors, are stated clearly below this declaration. Should this statement prove to be untrue I recognise the right and duty of the Board of Examiners to take appropriate action in line with the university's regulations on assessment.

Name: Richard Stone
ID No: N0782914

Name: Stephen Anderson
ID No: N078602

Name: Samuel Harrison
ID No: N0787108

Name: Thomas Harrison
ID No: N0833220