FINAL PROJECT

[Date]

Assignment 9

Team members:

Tony Cheng

Jose Chavez

Thien Nguyen

Ben Halpern

Jesus Sierra

Itz Rodriquez

Santa Ana Community College

CMPR 131

# 

[Introduction: 2](#_Toc85485333)

[Executive Summary: 3](#_Toc85485334)

[SDLC Flow and plan: 4](#_Toc85485335)

[Specification: 3](#_Toc85485336)

[Roles and Responsibilities: 4](#_Toc85485337)

[Student Class: 4](#_Toc85485338)

[VectorDriver Class 4](#_Toc85485339)

[ListDriver Class 4](#_Toc85485340)

[IntContainer Class 4](#_Toc85485341)

[Goals and objectives: 4](#_Toc85485342)

[Design Decisions: 3](#_Toc85485343)

[Functional Description: 4](#_Toc85485344)

[Design Objectives: 4](#_Toc85485345)

[Flow-Chart/ Program Hierarchy: 4](#_Toc85485346)

[UMLs: 4](#_Toc85485347)

[Solution Details: 3](#_Toc85485348)

[Main Components: 4](#_Toc85485349)

[Classes: 4](#_Toc85485350)

[Student: 4](#_Toc85485351)

[ListDriver: 4](#_Toc85485352)

[VectorDriver: 4](#_Toc85485353)

[IntContainer: 4](#_Toc85485354)

[Source Code Documentation: 3](#_Toc85485355)

[Design Patterns: 4](#_Toc85485356)

[Coding Standards: 4](#_Toc85485357)

[Coding Naming Convention: 4](#_Toc85485358)

[Algorithms: 4](#_Toc85485359)

[Testing Documentation: 4](#_Toc85485360)

[Testing Plan: 4](#_Toc85485361)

[Test Cases: 4](#_Toc85485362)

[Test Results: 4](#_Toc85485363)

[References 4](#_Toc85485364)

[Glossary: 4](#_Toc85485365)

[Terms: 4](#_Toc85485366)

[Coding Convention Reference: 4](#_Toc85485367)

[Naming Conventions: 4](#_Toc85485368)

[Conditional Documentation: 4](#_Toc85485369)

[Class Invariants: 4](#_Toc85485370)

[Flow Charts: 4](#_Toc85485371)

[Hierarchy Charts: 4](#_Toc85485372)

[UMLs 4](#_Toc85485373)

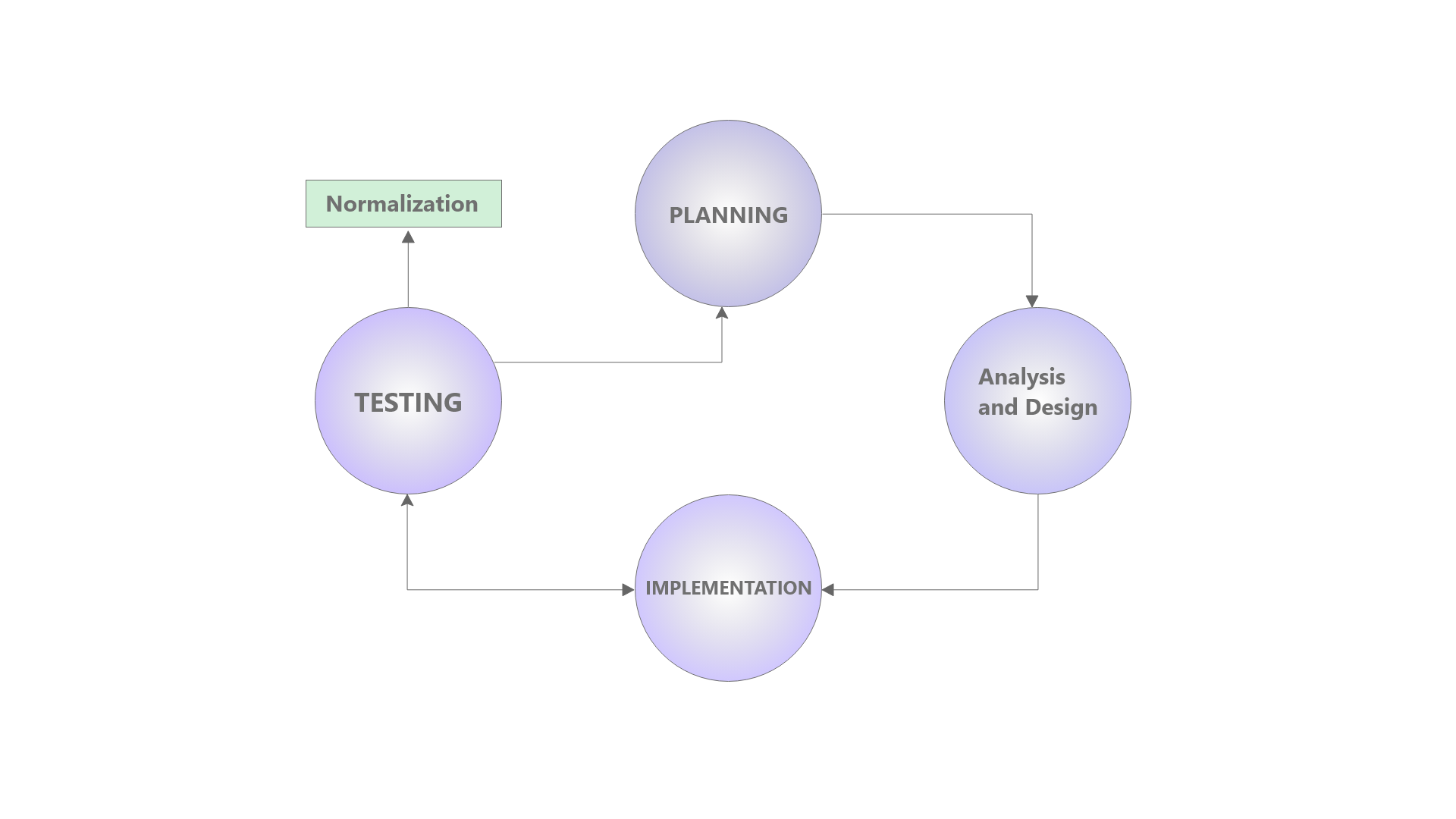
[Table of Figures: 4](#_Toc85485374)

# 

# Introduction:

# Executive Summary:

## SDLC Flow and plan:

[i]

The *Software Development Life Cycle (SDLC)*[[1]](#footnote-1) follows the sequence shown above. Starting at the planning stage of the software development life cycle, the programming team takes the *specification*[[2]](#footnote-2) given for the program and breaks it apart into smaller digestible segments. The assigned parts, the small digestible segments, are then analyzed and further broken down in the *Analysis and Design[[3]](#footnote-3)* phases. During this stage of the life cycle, the specification is broken down into a program flow, to understand the flow of the overall program. Each digestible stage follows its own iteration of the life cycle, branching back into the main flow of the program as each task and subtask are completed. In the analysis and design phases of the life cycle, each task, that was broken down into components and the algorithms, are tested in the efforts to optimize them. The different *implementations[[4]](#footnote-4)* and components[[5]](#footnote-5) of the program are mapped out, in forms such as *hierarchy charts[[6]](#footnote-6)*, *pseudo code[[7]](#footnote-7)*, *flow charts[[8]](#footnote-8)* and more *logical design elements[[9]](#footnote-9)*. Implementing these designs in the code and testing the code after each new addition, looping through the life cycle until the entire program is complete. Once the program is complete the life cycle continues as the code evolves and is maintained.

Tasked with programming and application to utilize the list, and vector standard Template library classes in an application running on a Student object as the underlying data type and a final application demonstration running on an integer data type. Driven by a menu function prompting the user for various options through the various tiers of the application. We split up the work in an agile based approach, the programming that implemented the list and vector class was done simultaneously, then merging those methods taken and learned from the implementation into the application demonstration option. Testing each method after it was written, basing each method of the UML and updating the UMLs for each class as the programming progressed. Coordinating regularly, at a rate of meeting every other day, to divide up the remaining work at each completed stage following the Development Life Cycle.

# Specification:

Given an example executable, we used the example executable as a reference for the style of the user interface of the program, and reverse engineering the program in order to understand the algorithms and methods within it. Following the specification given, we had to also debug the executable file given to us as it was not without *logic errors[[10]](#footnote-10)*.

We were specified to…

## Roles and Responsibilities:

### Class:

### Class:

### Class:

### Class:

## Goals and objectives:

Upon analyzing the design… Once the … class was finished then a more *agile[[11]](#footnote-11)* approach was able to be implemented and the … section and … section of the code could be … *Standard Template Library (STL)[[12]](#footnote-12)*…

## 

# Design Decisions:

With the specification in mind …

## Functional Description:

## Design Objectives:

Our objectives in designing the program are directed to optimize the transversal and manipulations of the Vector and List STL classes to that of the given specifications. Designing the classes with self-containment in mind to allow for both code reusability and ease of use when implementing the meu-driven classes. In addition, our objectives are aligned with displaying and demonstrating our understanding of the Standard Template Library classes used during the reverse engineering of the executable given during the specification.

## Flow-Chart/ Program Hierarchy:

[See the Glossary for reference to the Flow-Chart and Program Hierarchy Chart.]

The Flow-Charts referenced below in the glossary were designed during the design phase of the SDLC. They provide a way of developing a deeper understanding of the flow of the code and allow for different analysis to be posed in regards to things such as time analysis, and efficiency along with the general program flow. The first Figure is a flow chart showing that the….

(Program Flow Chart: pg.14, Figure 1)

The next design elements referenced are both Hierarchy Charts showing the program broken into smaller components. This helped in the dividing up of the various tasks and how the program worked together with the other classes.

(Program Hierarchy: pg. 17, Figure 2)

(Program Hierarchy Zoomed In: pg. 16, Figure 3)

## UMLs:

[See the Glossary for reference to the UMLs.]

UMLs are design elements of classes outlining the methods and the components in the function along with description for each function and the preconditions for those functions. The UMLs describe the purpose of each function and act as a roadmap for the programmer as well as those who desire to implement the code.

(… Class: pg. 17, Figure 4)

(… Class: pg. 18, Figure 5)

(… Class: pg. 19, Figure 6)

(… Class: pg. 20, Figure 7)

### 

# Solution Details:

## 

## Main Components:

The program consists of driver functions which drive the classes

### Classes:

### 

### [class name]:

### [class name]:

### [class name]:

### [class name]:

# Source Code Documentation:

A best practice when programming is adhering to a set of standards and patterns when coding and when coming up with names for classes, variables, and various function types. The standards we used in our solution, which we implemented throughout and at the end during the *normalization[[13]](#footnote-13)* of our code, are outlined in the following sections.

## Design Patterns:

At the header of each file, we placed header comments identifying the file name, the team members, and other identifying information about each file including a description. We organized the classes, using the structural pattern placing the accessors first followed by the mutator functions all after the constructor definitions inline in the classes. Placing navigating comments throughout the classes to help identify each section within the class. Some classes have an overloaded constant member version in order to provide access to the member within other constant defined member functions.

## Coding Standards:

We attempted to adhere to coding standards that attempt to mimic those of the Standard Template Library in order to better fit implementation with other classes within the STL to allow easy usability with those other classes.

## Coding Naming Convention:

When naming classes, we capitalized the first letter of the class in *Camel Hump[[14]](#footnote-14)* format, and made private members of the class begin with the prefix ‘m’ to easily identify it as a member variable. We then used the prefix ‘p’ to identify formal parameters passed to each function to identify them as parameters. For the … and the … Classes the postfix “Driver” was placed after the underlying data container type used in the class in order to avoid confusion, the same was applied to the application section of the program which implements the … Class, in a similar self-explanatory convention. The public member functions follow camel hump format or mimic that of the STL container class functions which are called within it, to add a layer of abstraction to the underlying container.

Documenting our code with three forward slashes to identify and utilize InteliTips[[15]](#footnote-15) documentation within visual studio to better allow reading of documentation in Microsoft Visual Studio when implementing each function. Before each function, the precondition and postconditions are mapped and described. Short tag descriptions for commonly identifiable function traits are included within square brackets prior to the Precondition and Postcondition.

### Algorithms:

## 

## Testing Documentation:

Describe a little about the testing phase of the report and what it is for and what it helped us see.

### Testing Plan:

### 

### Test Cases:

### 

### Test Results:

# 

# References

Agile Alliance. *Agile 101*. n.d. Website. October 2021. <https://www.agilealliance.org/agile101/>.

Altexsoft. *Technical Documentation in Software Development Types Best Practices and Tools*. 01 December 2020. Website. October 2021. <https://www.altexsoft.com/blog/business/technical-documentation-in-software-development-types-best-practices-and-tools/>.

Ivanna. *types-of-software-development-documentation*. Ed. Alexandra Rostovtseva. 22 May 2020. Website. October 2021. <https://gbksoft.com/blog/types-of-software-development-documentation/>.

SCRUM.org. *What is SCRUM*. n.d. Website. October 2021. <<https://www.scrum.org/resources/what-is-scrum/>>.

Gaddis, Tony. “Introduction to C++.” Starting Out with C from Control Structures to Objects, 9th ed., Pearson Education, Inc, 2017 , print.

# Glossary:

## Terms:

1. Abstraction: Abstraction is a method of encapsulation, hiding the inner working and inner components, limiting the visibility to the scope defined by the programmer.
2. Software Development Life Cycle: The Software Development Life Cycle may differ depending on the model and the application at play, however generally the Life Cycle consists of a planning phase, an analysis phase, a design phase, an implementation phase, and a testing phase. The Software Development Life Cycle may be abbreviated as SDLC.
3. Specification: The specification of a project is the task which has been assigned to the programming team. The specification should also include the scope of work and the expectations of the job. Generally included in the specification is the medium in which the work is to be submitted and the desired format as well.
4. Analysis and Design: Analysis and Design is the second stage of the SDLC, in which the specification is broken down and UML and Flow charts are started and completed as the analysis and design progress. It is important to consider the different factors such as compile time, memory allocation and efficiency when analyzing and designing the program based on the given specifications. The time analysis can be done using Big-O notation.
5. Implementation: Implementation is the fourth stage of the SDLC. During this stage, the code starts to come together as the code is translated from the logical designs such as UMLs, into source code.
6. Components: Components are the inner working parts of a whole. When applying abstraction, components are the inner working parts which are abstracted away from the end-user.
7. Hierarchy Chart: A Hierarchy chart is a Logical Design Chart which is used to show the breakdown of the various functions and what tasks or methods they may branch off into. This is a helpful design element following the creation of the flow chart. Paired with the flow chart, one may get a better picture at the design, architecture and flow of the code and program.
8. Pseudo Code: Pseudo code is code that is abstracted at a level closer to human readable languages such as English. Pseudo code combines a coding language and another language in order to produce a rough outline of the functions and classes that are encapsulated within the Pseudo-Code.
9. Flow Chart: A Flow Chart is a graphical logical design element which describes the runtime flow of the program and provides a visual representation of how the code may operate. Another type of Flow Chart may be a Steak Diagram, which outlines the steps and conditions in a bubble format.
10. Logical Design Forms: Logical Designs are those that help describe the logical process that goes into the design of the code, as well as the methods of operation. Logical Designs are crucial to the design phase of the Software Development Life Cycle.
11. Logic Errors: Logic Errors are errors which occur in the logical flow of the program. They do not necessarily throw exceptions or cause the program to halt, however they do lead to unwanted results within the program and tend to be the hardest error to find.
12. GitHub: Github is a free online resource which allows for teams to coordinate using the git protocol which is a workflow tracking protocol which keeps track of changes in the working directories.
13. Agile Workflow: The agile workflow is a workflow process in which the tasks are divided up and completed asynchronously, reconvening at a regular frequency to redivide the work as time progresses. This allows for various parts of the code to be reviewed at asynchronous times and may allow for more review of code depending on the implementation of the workflow. The Agile workflow is more adaptive that the waterfall counterpart which is generally used in bigger teams and organizations.
14. Standard Template Library (STL): A general library of data structures and functions for lists, stacks and many more.
15. Normalization: The practice of standardizing the format and the style of a program code to a state of great consistency.
16. Reference Variable: A reference variable is a variable that passes the memory address and references the original data, so when the variable is manipulated, the original variable is manipulated, versus the alternative which passes a copy.
17. Camel Hump Format: The way of writing out expressions in which there are no spaces or underscores and the first letter of the first word is written in lowercase and the following words are started with a upper case letter (e.g. exampleOfCamelCase)
18. InteliTips: InteliTips are small toolkit style popup boxes which appear when typing in visual studio. They provide tips and recommendations as well as live documentation based on an internal XML document which Visual Studio maintains.

## Coding Convention Reference:

### Naming Conventions:

[tags] are used to define design element comments such as this one and those of [const] for constant members and accessors, and [mutator] for mutator methods

When the tag is lower case the design element is a local comment, when the first letter is capitalized the design element is a global tag relating to a comment with a scope pertaining to the document as a whole.

The names of classes are capitalized following the CamelHump naming convention. 'm' is used as a prefix to define private members of a class following the CamelHump naming convention. 'p' is used as a prefix to define formal parameters of methods following the CamelHump naming convention

### Conditional Documentation:

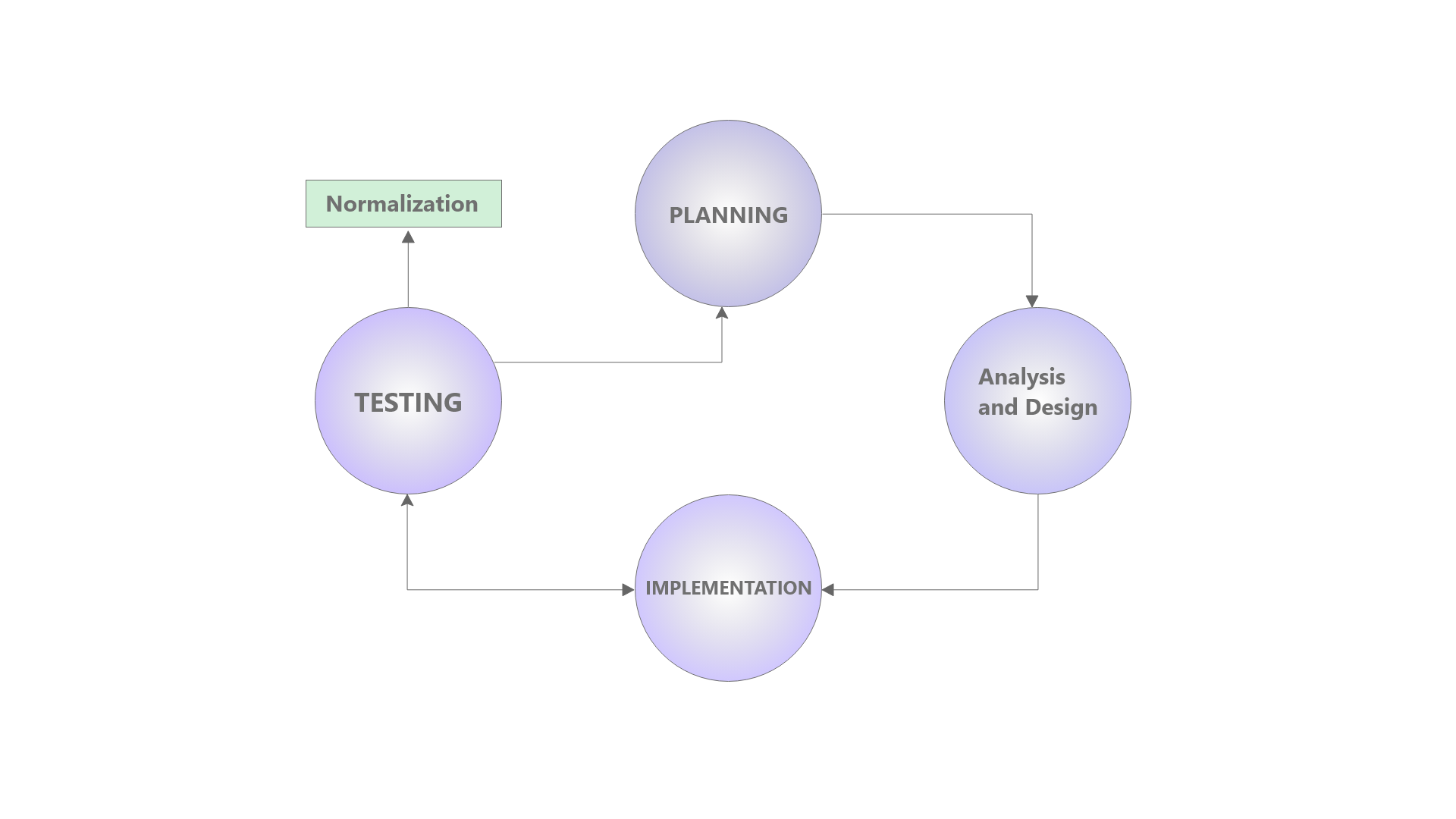
Prior to functions, the documentation of the precondition and postcondition are described with three brackets so that the descriptions would work with Microsoft Visual Studio InteliSense, which produces would then display the precondition and postcondition when as a tip when the function is being implemented in the various parts of the program.

### Class Invariants:

Class invariants provide a list of the various methods in each class and provide a short description outlining the use and implementation notes for that class. The methods in the class invariant don't provide formal parameters but instead they show the datatype of each formal parameter in a similar format and style that might be seen in a prototype declaration.

Flow, Structural, and Document comments are seen throughout the code and are noted by the use of only two forward slashes '//' when defining the comments. A series of '###' may be used to denote flow indicator to help the programmer navigate the various classes

## Flow Charts:



[i]

… [ii]

## Hierarchy Charts:

… [iii]

…

[iv]

## UMLs

… [v]

… [vi]

… [vii]

… [viii]

## Table of Figures:

Figure 1 Software Development Life Cycle 3,14

Figure 2 Program Flow 14

Figure 3 Program Hierarchy 15

Figure 4 Zoomed in Hierarchy Chart 16

Figure 5 [class name] UML 17

Figure 6 [class name] UML 18

Figure 7 [class name] UML 19

Figure 8 [class name] UML 20

[i] Figure 1: Software Development Life Cycle

[ii] Figure 2: Program Flow Chart

[iii] Figure 3: Program Hierarchy Chart

[iv] Figure 4: Program Hierarchy Chart zoomed

[v] Figure 5: [class name] Class UML

[vi] Figure 6: [class name] Class UML

[vii] Figure 7: [class name] Class UML

[viii] Figure 8: [class name] Class UML

1. Software Development Life Cycle: Software Development Life Cycle: The Software Development Life Cycle may differ depending on the model and the application at play, however generally the Life Cycle consists of a planning phase, an analysis phase, a design phase, an implementation phase, and a testing phase. The Software Development Life Cycle may be abbreviated as SDLC. (Glossary pg. 18) [↑](#footnote-ref-1)
2. Specification: The specification of a project is the task which has been assigned to the programming team. The specification should also include the scope of work and the expectations of the job. Generally included in the specification is the medium in which the work is to be submitted and the desired format as well. (Glossary pg. 18) [↑](#footnote-ref-2)
3. Analysis and Design: Analysis and Design is the second stage of the SDLC, in which the specification is broken down and UML and Flow charts are started and completed as the analysis and design progress. It is important to consider the different factors such as compile time, memory allocation and efficiency when analyzing and designing the program based on the given specifications. The time analysis can be done using Big-O notation. (Glossary pg. 18) [↑](#footnote-ref-3)
4. Implementation: Implementation: Implementation is the fourth stage of the SDLC. During this stage, the code starts to come together as the code is translated from the logical designs such as UMLs, into source code. (Glossary pg. 17) [↑](#footnote-ref-4)
5. Components: Components are the inner working parts of a whole. When applying abstraction, components are the inner working parts which are abstracted away from the end-user. (Glossary pg. 17) [↑](#footnote-ref-5)
6. Hierarchy Chart: A Hierarchy chart is a Logical Design Chart which is used to show the breakdown of the various functions and what tasks or methods they may branch off into. This is a helpful design element following the creation of the flow chart. Paired with the flow chart, one may get a better picture at the design, architecture and flow of the code and program. (Glossary Pg. 17). [↑](#footnote-ref-6)
7. Pseudo Code: Pseudo code is code that is abstracted at a level closer to human readable languages such as English. Pseudo code combines a coding language and another language in order to produce a rough outline of the functions and classes that are encapsulated within the Pseudo Code. (Glossary pg. 18). [↑](#footnote-ref-7)
8. Flow Chart: A Flow Chart is a graphical logical design element which describes the runtime flow of the program and provides a visual representation of how the code may operate. Another type of Flow Chart may be a Steak Diagram, which outlines the steps and conditions in a bubble format. (Glossary pg. 18). [↑](#footnote-ref-8)
9. Logical Design Forms: Logical Designs are those that help describe the logical process that goes into the design of the code, as well as the methods of operation. Logical Designs are crucial to the design phase of the Software Development Life Cycle. (Glossary pg. 18). [↑](#footnote-ref-9)
10. Logic Errors: Errors are errors which occur in the logical flow of the program. They do not necessarily throw exceptions or cause the program to halt, however they do lead to unwanted results within the program and tend to be the hardest error to find. (Glossary pg. 18). [↑](#footnote-ref-10)
11. Agile Workflow: The agile workflow is a workflow process in which the tasks are divided up and completed asynchronously, reconvening at a regular frequency to redivide the work as time progresses. This allows for various parts of the code to be reviewed at asynchronous times and may allow for more review of code depending on the implementation of the workflow. The Agile workflow is more adaptive that the waterfall counterpart which is generally used in bigger teams and organizations.

    (Glossary pg. 19). [↑](#footnote-ref-11)
12. Standard Template Library (STL): A general library of data structures and functions for lists, stacks and many more. (Glossary Pg. 19). [↑](#footnote-ref-12)
13. Normalization: The practice of standardizing the format and the style of a program code to a state of great consistency.

    (Glossary pg. 19). [↑](#footnote-ref-13)
14. Camel Hump Format: The way of writing out expressions in which there are no spaces or underscores and the first letter of the first word is written in lowercase and the following words are started with an upper-case letter (e.g., exampleOfCamelCase). (Glossary pg. 19). [↑](#footnote-ref-14)
15. InteliTips: InteliTips are small toolkit style popup boxes which appear when typing in visual studio. They provide tips and recommendations as well as live documentation based on an internal XML document which Visual Studio maintains. (Glossary pg. 19). [↑](#footnote-ref-15)