**Technical Writing Project Cover Sheet**



|  |  |
| --- | --- |
| **Capstone Proposal Project Name:** | **UML Flow Program** |
| **Student Name:** | **Kevin Fode** |
| **Degree Program:** | **BS, Software Development** |
| **Student Mentor Name:** | **Nancy Corea** |

**Table of Contents**

[Capstone Proposal Introduction 1](#_Toc409961027)

[Review of Other Work 2](#_Toc409961028)

[Project Rationale 4](#_Toc409961029)

[Systems Analysis and Methodology 7](#_Toc409961030)

[Goals and Objectives 9](#_Toc409961031)

[Project Deliverables 11](#_Toc409961032)

[Project Plan and Timelines 14](#_Toc409961033)

[References 15](#_Toc409961034)

# Capstone Proposal Introduction

One of the key steps in developing a program is drawing out a Unified Modeling Language (UML) diagram. This diagram acts as a visual flowchart. Often times, participants write out steps on post-its and place them on a white board where they draw arrows to indicate the logical flow from one step to the next. This type of process allows participants to move steps around, add and remove steps, and change the logical flow freely. Overall, UML diagramming is a very useful starting technique for getting a process out of people’s heads and into a form that a programmer can use to write code that a machine can understand.

Unfortunately, this type of diagram has limited utility while a programmer is testing and verifying the completeness of his program. This is true even if this diagram is converted into a drawing a programmer can keep at his desk. It is the responsibility of the programmer to make sure that the process outlined in the UML diagram matches what the program does. To do this, many programmers will trace through with a finger while stepping through the program. This way of doing things has a number of limitations.

For one, it is easy for the programmer to lose their place while tracing through the diagram. This can happen in several ways. A colleague may interrupt them. They may need to perform a two-handed operation. This can also occur when the diagram they are using is highly complex.

Another limitation of using a visual UML diagram is that any process can look far more complex than it actually is. This can hinder progress as the programmer steps through his code. When the programmer is testing and verifying his work, they are only concerned with a small portion of the diagram at any one time. However, the rest of the diagram is still there. From the programmer’s point of view, all this extra information is just noise. This idea serves as the basis for the proposed program.

The proposed program is a text-based, UML diagram builder. The user will be able to enter steps into the program. Each step can be edited, deleted, and reordered. The user will also be able to add decision trees which affect the step order. Once the UML diagram is complete, the user can walk through the process one step at a time. The user is responsible for advancing to the next step by hitting a key. This way, it is not possible for the programmer to lose their place in the diagram. In addition, by displaying each step one at a time, the overall layout of the process is hidden from the user, allowing them to focus on the content of each step. In this way, the complexity of the program’s workflow can be more easily digested.

# Review of Other Work

Visual UML diagrams can be key to breaking down complicated processes. One of the main benefits to UML modeling is a shared understanding of how a process is supposed to arrive at a result from an initial state. While UML diagramming is not considered as critical for smaller projects with smaller teams, it can be critical with larger projects with geographically dispersed teams (Chaudron, Heijstek, & Nugroho, 2012). Once completed, UML diagrams can serve three different purposes: they can model complex problems, they can help describe the proposed solution to other team members, and they can layout the architecture of the purposed solution.

One of the most basic goals of UML is to break down the problem into parts that users can better understand (Chaudron et al., 2012). Creating a UML diagram helps the programmer break down a solution into more manageable pieces. By documenting each step along the way, users can describe how a process can take an input and process it into a finished result. UML modelling allows a programmer to drill down from the big picture and focus only on what needs to be done next. This type of planning can be very effective for getting a programmer’s thinking from what needs to be done to how it will be done.

Another goal of using UML diagrams is to help teams hold a shared vision of what the solution will entail (Chaudron et al., 2012). Programs are seldom built by individuals working alone. A well-detailed UML diagram can be instrumental in getting everyone on the same page. Without it, a variety of time wasting miscommunications can result. A good UML process describes what kinds of inputs an activity receives, and what kinds of outputs result from each activity. Programming is a highly modular process, where programmers are given activities to code which are later integrated into the final product. When a programmer is assigned to write code for an activity, knowing exactly what the boundaries are for the activity keeps them from writing code already written by someone else and from neglecting to write code they thought was being handled by another programmer.

A third goal of UML diagrams is to act “as a blueprint for the implementation (Chaudron et al., 2012).” Groups who use this as a goal have highly detailed UML diagrams containing a variety of object-oriented information about a process. This usually involves a subset of UML called Object Management Group (OMG) (Pooley, Wilcox, 2004, p. 4).Very little is left to the imagination with this level of commitment to the diagramming process. This level of detail is mainly used when communications between team members is less frequent (Chaudron et al., 2012).

The more that a team commits to the UML process, the more detailed the output can become. However, as the models become more complex, they also become far more difficult to read (Drusinsky, 2006, p. 44). While the purpose of the modelling process is to break down an activity into manageable and understandable parts, the overall representation that is the UML diagram can make a complex project look completely unmanageable. As a visual guide, a UML diagram begins to fail at this point as a useful tool going forward. For those who decide to use visual UML diagrams, this results in an unfortunate choice. On one hand, they can choose to have a less detailed UML, which may lead to misunderstandings, duplicate work, and time lost. On the other hand, they can choose to have a more detailed UML, which results in a product that can be difficult to understand (Chaudron et al., 2012).

To make the UML diagram easier to understand, the proposed program will hide the complexity of the diagram by only presenting the piece the user is currently working on. This program will be produced using the small project management (SPM) methodology outlined by Sandra Rowe (Rowe, 2015, p. 48-56). This process is similar to the waterfall methodology, as it is a linear process designed for smaller projects. SPM uses the same basic life cycle defined in the PMBOK® guide, except that it combines the Executing and the Monitoring and Controlling phase into a single process (Rowe, 2015, p. 53). This document starts the initiating process. This process is complete once this document is approved. The planning process will break down project objectives into manageable pieces. Class diagrams will be drawn up and checklists are created which detail what is to be done in each component of the final product. The controlling phase uses these checklists to write the code and to check whether the component meets its specifications. The closing process finishes the development of the program. Closing documents, such as lessons learned, are generated at this time.

# Project Rationale

The purpose of the proposed project is to assist the programmer’s understanding of the desired functionality of the program they are attempting to create. This way, the programmer can spend less time trying to grasp the overall complexity of what they are trying to write, and focus solely on the task before them. For the purposes of this program, any information that is not the current focus of the programmer is considered noise.

To accomplish this, the program will display information step by step, and only proceed at the bidding of the user. The program will proceed in a linear fashion from beginning to end, while allowing the user to answer multiple-choice questions along the way that may result in iterative processes. The user is expected to enter information into the program that will then become the step by step process that can be used for testing and troubleshooting. In this way, the program is given a great deal of flexibility in how it is applied.

As an example, a user may decide to use this program to verify the completeness of the final program. To do this, the user could enter the contents of a UML diagram created during the planning process. They could then use the program to verify the final result by stepping through the program as the result is tested simultaneously. This would be considered to be at the complete picture level.

While the program can be used to test the entire product, it can also be used to build UML diagrams for components as well. This level is considered to be the class or component level of detail. UML diagrams created during the planning process often are not broken down to the point where the inner workings of components are revealed. This is because doing so would make the diagram overly complicated, and would also waste the time of any programmer not involved in the creation of the component in question. However, there is nothing stopping the individual programmer from creating a UML diagram specifically for the component they are working on. In fact, this process is recommended whenever additional understanding is required. After the programmer creates the UML, they can then use the program to test the component.

For a more granular level of detail, the programmer can use the proposed program to analyze the workings of a process line by line. This would be considered the method or process level of detail. The UML necessary for this level of detail would be highly specific, and most likely only used while troubleshooting. This level would include what the process needs to do every step of the way to return the proper result.

Due to the varying levels at which this program can be applied, the feasibility of its applications also vary. At the complete picture level, the user inputting the UML for testing has a choice of either being very general with each step, or spending a great deal of time entering more specific information. This will depend on the size of the finished result. The larger the result being tested, the less feasible it is that the proposed program will provide value.

However, at the component level, the proposed program has a much higher chance of adding value. At this level, the programmer can start his task by creating the UML diagram with the proposed program. This will provide a greater understanding of the problem and allow the user to focus on each step as they come by running through the proposed program.

At the method level, the proposed program is once again likely to have little value. Because this level requires such a precise level of detail, it is highly unlikely that a programmer would want to spend the required amount of time to implement the proposed program at this level. If they did choose to, they would probably only do so if they were completely stuck. Because the UML required at this level is so detailed, this process would most likely result in a half finished UML since the programmer would probably stop as soon as they happened upon the step they were missing. It is debatable as to whether it would be better for the programmer to try to use the proposed program for this level of troubleshooting, or simply start over.

For these reasons, the primary purpose of the proposed program will be to support programmers who are operating at the component level of testing and troubleshooting. Some users may find it useful at the other two levels, but this will not be the focus of this endeavor.

# Systems Analysis and Methodology

Code validation is a key component of the software development process. Currently, a programmer wishing to validate that their code preforms in the intended way is forced to trace through a UML diagram as they execute their program. The programmer can choose to do this in a number of ways.

One way would involve tracing through a program’s UML diagram with a finger while simultaneously executing the program. This method requires the creation of a UML diagram that can fit on the programmer’s desk. For complex systems, the diagram may need to be compressed to a point where it become difficult to read and hard to trace. Any interruption of this task will cause the programmer to lose their place on the diagram, causing them to either start over or waste valuable time trying to figure out where they left off.

Another way the programmer could try to validate their code would be to work off a whiteboard, making a checkmark at each step. This approach is also problematic. Any time that a checkmark needs to be removed, the programmer risks erasing valuable information. This is also makes it problematic to trace steps through iterative loops.

To alleviate these issues, the proposed program will be responsible for keeping track of where in the process the programmer currently is. No changes to the UML diagram will be made as the user steps through it, maintaining the diagrams integrity. Any interruptions to this process will be irrelevant. The programmer can simply pick up where they left off without having to worry about where in the code they were before they stopped. The proposed program will keep track of their location in the UML diagram for them.

As discussed before, the methodology used for the creation of the proposed program will be the small project management (SPM) methodology outlined by Sandra Rowe (Rowe, 2015, p. 48-56). This methodology is made for small projects like the proposed program. It is also made for projects with small teams.

The initiating process begins with the creation of this document. Stakeholders are identified during this process as well (Rowe, 2015, p. 58). For this program, the stakeholders are any programmers wishing to make their validation and troubleshooting processes easier, and are willing to invest the time it takes to load the UML diagram into the proposed program. The people who employ these programmers are considered secondary stakeholders. The initiating process completes when this document is approved.

The planning process starts with the collection of requirements, which are then used to build a scope statement (Rowe, 2015, p. 70). A work breakdown schedule (WBS) is created to take components of the scope statement and break them down into deliverables that are more manageable. Costs, risks, and communication plans which are normally a part of this process are considered irrelevant to this project. The only cost in this project is time. All risks are assumed by the programmer coding the proposed project, and will be dealt with in a manner of their choosing. No risks can be mitigated in this case. There will be no communication plan, since there will be no one outside of the project team to communicate with. Other steps in this process involve the creation of a schedule and a project plan. Since both of these tasks are requirements for this document, the planning process will be nearly complete once this document is approved.

The only portion of the planning process that will not be complete once this document is approved, is the creation of the checklists that will be used to verify the functionality of each component created during the controlling process. While there are a number of activities in the controlling process of SPM, the only two being used for the proposed project involve the reviewing and approval of deliverables and the managing of any scope changes needed (Rowe, 2015, p. 117). The rest of the activities are considered redundant in a one person project team.

The final process in SPM is the closing process (Rowe, 2015, p. 129). This is when the project will be evaluated for its completeness. A project closure report will also be generated to document any lessons learned while working on the project.

# Goals and Objectives

The only goal of the proposed program is to provide a method to easily validate the programmer’s code and to verify that the code performs the way it was intended. Validation is an important part of the quality assurance process. As discussed previously, this process can be difficult to manage using traditional methods. The proposed program provides an alternative to these methods, insuring the accuracy and timeliness of the results. The proposed program will achieve this goal by accomplishing several objectives:

* the program must have an edit mode to allow the user to create and change a UML flow chart
* the program must have a stepping mode to allow the user to logically flow through the UML flow chart once it is complete
* the program must have a method for storing entered values for future use
* the program must have a way for the user to edit values that were previously entered
* the program must link values together in a logical string
* the program must allow the user to create multiple paths to different following values
* the program must allow the user to delete values
* the program must allow the user to save/load data from a file

The first objective is that the program must contain an edit mode. This mode should allow the user to freely create and edit a UML flow chart. Unless the user has previously saved UML data, the user will always use this mode first.

Second, the proposed program will need a stepping mode. This mode will be used to step through the UML flow chart created in the edit mode. This is the process the user will execute to verify the functionality of their code.

The third objective is that the proposed program must be able to take user entered values and store them for later use. When the programmer later executes the stepping mode of the program, these values will be sent back to the programmer. These values should be used to let the programmer know what step is currently being executed and what tasks should currently be under way.

Fourth, the proposed program must allow the user to edit these values should the need arise. This will be another function of the edit mode. Without this, the programmer would be required to enter values correctly on the first try.

The fifth objective is that the proposed program must link these values together in a logical string. This string will be created and edited within the edit mode. This string tells the stepping mode which value will be the next for display.

The sixth objective is to allow for multiple paths from the current value to different following values. These junctures will allow the user to make multiple choice decisions as to the next step being used in the process. This objective is key. It is this decision path process that creates a great deal of confusion while trying to troubleshoot and while trying to validate a process. Without achieving this objective, the programmer would be left with a strictly linear process, which is not difficult to model or test. It is therefore imperative that the proposed program meets this objective.

The seventh objective is to allow the user to delete values if necessary. Creating a UML diagram can be a messy process. The user should not be forced to start over if a value entered previously turns out to not actually be part of the process being mapped.

The eighth objective is to create the ability to save and load the UML created during the edit mode process. The UML necessary to map a process can be quite large, and may need to be stored for later retrieval. Forcing the user to reenter this information would be tedious, and would limit the utility of the proposed program.

# Project Deliverables

The proposed program will have a number of deliverables to mark the progress of this project. Each deliverable will be thoroughly tested to maintain quality throughout the process. The deliverables map to the previously stated objectives as follows:

| **Project Deliverable** | **Objective** |
| --- | --- |
| Edit Mode | The program must have an edit mode to allow the user to create and change a UML flow chart. |
| Stepping Mode | The program must have a stepping mode to allow the user to logically flow through the UML flow chart once it is complete. |
| Value Storage | The program must have a method for storing entered values for future use. |
| Value Editing | The program must have a way for the user to edit values that were previously entered. |
| UML Path | The program must link values together in a logical string. |
| Decision Trees | The program must allow the user to create multiple paths to different following values. |
| Value Deletion | The program must allow the user to delete values. |
| Save/Load UML | The program must allow the user to save/load data from a file. |

The Edit Mode deliverable marks the completion of the bare bones menu system for the proposed project. This mode will be similar to the stepping mode, in that it will step through any created values called nodes. The edit mode will offer choices to add new nodes, edit the active node, delete the active node, or navigate to another node (next or previous). This mode will use the stepping feature to keep track of which node is currently active. The active node is available for editing and deletion in future deliverables.

The Stepping Mode deliverable will add a menu option to the previous deliverable which will allow the user to enter stepping mode. Once completed, this mode will offer a stripped down version of the edit mode. The only options available in this node are to proceed to the next node or exit stepping mode.

The Value Storage deliverable involves the planning and development of the data object, which will be used to store the values entered by the user. This deliverable will also include additional code added to the edit mode which will allow the program to accept input. Each input will be stored in its own data object. Each data object should also contain a unique identifier that the user can use in the future to make iterative loops.

The Value Editing deliverable will add additional functions to the edit mode of the program. These functions will allow the user to change the previously inputted value stored in the data object. The edit mode will only allow changes to be made to the active node.

The UML Path deliverable adds additional information to the data object. Each data object will need to store a link to the next data object in the chain. Additional coding will be added to both the edit mode and the stepping mode to allow the user to navigate to the next node.

The Decision Trees deliverable adds the ability to make multiple choice decisions as to which node is the next node in the path. The data object needs to be modified to allow for multiple next nodes. Both the edit mode and the stepping mode will be modified to present a multiple choice menu whenever a decision tree is detected. Since this is the point where navigation is as complicated as it will get, this will also be the point in which the previous node function will be incorporated into the edit mode. Also, there will be a function to map to another node using its unique identifier. This will allow the user to create iterative loops.

The Value Deletion deliverable will add the ability to delete a data object in the edit mode. This function creates a bit of a problem though. Since it is possible to have iterative loops in this design, it is also possible to remove a node that could be navigated to using the previous node function developed in the previous deliverable. To alleviate this issue, any time the user is deleting a node which is contained in the previous path, the user will be warned that the previous path will be purged of the entire path up to and including the first time the deleted node was used. The user will be unable to move to previous nodes after this deletion, but the program will not crash due to a null pointer exception. The edit mode will be further changed to allow the user to change the active node to the start node.

The Save/Load UML deliverable will add the ability to save and load previously entered node information. This information will be stored in a simple text file. Once this deliverable is accepted, the proposed project will be completed.

# Project Plan and Timelines

The projected timeline for the proposed project is laid out in the table below. Since the first four objectives are simple and interrelated, it is expected to be completed on the first day. Additional days may be added to the schedule if needed. Allowances have been made for weekends.

|  |  |  |  |
| --- | --- | --- | --- |
| **Project Deliverable or Milestone** | **Duration** | **Planned Start Date** | **Planned End Date** |
| Edit Mode | 1 Hour | 7/13/17 | 7/13/17 |
| Stepping Mode | 1 Hour | 7/13/17 | 7/13/17 |
| Value Storage | 3 Hours | 7/13/17 | 7/13/17 |
| Value Editing | 3 Hours | 7/13/17 | 7/13/17 |
| UML Path | 8 Hours | 7/14/17 | 7/14/17 |
| Decision Trees | 8 Hours | 7/17/17 | 7/17/17 |
| Value Deletion | 8 Hours | 7/18/17 | 7/18/17 |
| Save/Load UML | 24 Hours | 7/19/17 | 7/21/17 |

# References

Chaudron, M., Heijstek, W., Nugroho, A. (2012). How effective is UML modeling? *Software & Systems Modeling*, 11(4), 571-580.

Drusinsky, D. (2006). *Modeling and verification using UML statecharts: A working guide to reactive system design, runtime monitoring and execution-based model checking*. Burlington, MA: Newnes.

Pooley, R., Wilcox, P. (2004). *Applying UML: Advanced applications*. Oxford: Butterworth-Heinemann.

Rowe, S. (2015). *Project management for small projects, second edition*. Tysons Corner, Virginia: Management Concepts, Inc.