2016

University of North Carolina Wilmington

Master of Science in

Computer Science and Information Systems

Proceedings

https://csbapp.uncw.edu/mscsis

A CASE STUDY: IMPROVING A PROTOTYPE APPLICATION TO CURRENT DEVELOPMENT STANDARDS

Michael Fabbri

A Capstone Project Submitted to the
University of North Carolina Wilmington in Partial Fulfillment
of the Requirements for the Degree of
Master of Science

Department of Computer Science
Department of Information Systems and Operations Management

University of North Carolina Wilmington

2016

Approved by

Advisory Committee

Dr. Devon Simmonds	Dr. Douglas Kline

Dr. Thomas Janicki, Chair

Accepted By

Dean, Graduate School

Abstract

This study investigates the concerns that should be addressed when scaling source code from a single-developer prototype to a production code base with multiple active developers. The purpose of this study is to create a new foundation for a web application through selecting new a framework, configuring a deployment solution, analyzing the previous implementation, and managing the development process with multiple team members. In addition, this study addresses the supporting processes required for a production environment, such as the implementation of source control and initiating a development workflow. The web application used for this study was developed by multiple master's students across five years with no professional oversight. This study describes the process that the organization used to create a productive development environment while rewriting the prototype application.

I. Introduction

This capstone project will discuss the evolution of a prototype web application into a more robust and maintainable cloud-based system. Specifically, this paper will discuss the upgrade of the Universal Writing Continuum (UWC), a web application designed to improve the process of evaluating student writing during primary and secondary education. The enhancements to this application described throughout this paper were necessary in order to reduce the complexity of the software development process and provide a new foundation on which the product can grow.

This paper explains the improvements made to the UWC. The enhancements described in this paper are (1) the change of the programming language and framework, (2) the modification to the application's infrastructure, (3) the quality of systems analysis, and (4) the improvement of project management practices.

Chapter two provides the context of this project by explaining the background of the company, the existing web application, an explanation of the previous development to the application, a description of the current development team, and an overview of the enhancements made to the application.

The structures of chapters three and four are divided by the categories of the project deliverables which are software development, infrastructure improvements, systems analysis, and project management. The primary difference between chapters three and four is that chapter three explains the scope of the project and lists the deliverables that were accomplished while chapter four analyzes of the completion of the deliverables.

Chapter five reiterates the project deliverables and summarizes the benefits of each.

2. Background

The purpose of this section is to provide background information necessary for an understanding of the work completed for this capstone project. This section is composed of the background of the company, a summary of the product that the company offers to its customers, a description of the product's development prior to this project, a description of the product's development during the project, and a summary of the enhancements that were made to the product as a result of this project.

2.1. Uni-SPIRE

This capstone project was completed in cooperation with the company Uni-SPIRE. Uni-SPIRE is an educational technology company located in Wilmington, North Carolina. Uni-SPIRE was founded in 2012 by Dr. Deborah Powell, an associate professor in the University of North Carolina Wilmington's Watson College of Education, in affiliation with the University of North Carolina Wilmington.

Uni-SPIRE was founded to develop and market resources for educators. The company's core objectives are summarized in the company's mission statement:

> "Uni-SPIRE aims to impact PK-12 teachers' understanding of effective instruction in English language arts, mathematics, science, history and integrated curriculum by producing quality research-based assessments, data analyses, professional development and curricular materials that lead to an increase in

> student achievement, not only on standardized assessments but for readiness for college and careers." (Powell)

To accomplish this ambitious mission, Uni-SPIRE began with the development of a formative assessment tool to assist with the evaluation of student writing skills and supplementary professional development services.

Uni-SPIRE's formative assessment tools are the culmination of Dr. Powell's research spanning from 2004 until present day. In 2004, Dr. Powell developed a formative assessment framework to evaluate writing for students from kindergarten through grade 5. The framework is rooted in international standards and research performed in the area of writing education. The framework has been continuously improved through Dr. Powell's research since its inception, most notably including Common Core standards in 2010. The formative assessment tool is now a compilation of standards from multiple sources, including the Common Core Standards for Writing, the national standards for English from Australia, United Kingdom and various provinces in Canada, as well as other published continua and rubrics on writing and spans from pre-kindergarten through college freshman English.

The formative assessment framework developed by Dr. Powell is based on ten distinct elements of writing proficiency that students should gain throughout the entirety of their primary and secondary education. These ten elements of writing are:

1. Ideas and Content
2. Organization and Structure
3. Voice and Point of View
4. Word Choice and Description
5. Sentence Structure and Fluency

6. Conventions
7. Presentation and Publishing
8. Writing Process
9. Research and Writing to Learn
10. Attitude and Range of Writing

The framework also specifies fourteen developmental levels that students experience throughout their education:

1. Pre-kindergarten to beginning kindergarten
2. Middle of kindergarten
3. End of kindergarten to beginning of1st grade
4. End of 1st grade to beginning of 2nd grade
5. End of 2nd grade to beginning of 3rd grade
6. End of 3rd grade to beginning of 4th grade
7. End of 4th grade to beginning of 5th grade
8. End of 5th grade to beginning of 6th grade
9. End of 6th grade to beginning of 7th grade
10. End of 7th grade to beginning of 8th grade
11. End of 8th grade to beginning of 9th grade
12. End of 9th grade to beginning of 10th grade
13. Beginning of 11th grade to end of 12th grade
14. Honors, dual credit & advanced placement

Thus, the formative assessment framework provides detailed description of the ideal developmental progression of a student in each element of writing across all fourteen developmental levels.

The professional development services offered by Uni-SPIRE are intended to teach educators about evaluating writing and how to effectively implement the formative assessment tools offered by Uni-SPIRE. A minimum of seven hours of professional development is required for all of Uni-SPIRE's customers to ensure that educators have a thorough understanding of the formative assessment tools.

2.2. The Universal Writing Continuum

After developing the formative assessment framework, Dr. Powell conducted market research on the framework's acceptance by educators. Feedback from the educators led to the decision that the formative assessment framework would be most valuable if it were accessible online. From this realization, The UWC was created.

The UWC is currently Uni-SPIRE's only product. Development on the UWC began in 2012 as a collaborative effort between Dr. Powell and a graduate student in the University of North Carolina Wilmington's Masters of Computer Science and Information Systems program to develop a web application capable of serving the powerful formative assessment platform to educators around the world.

2.3. Previous Development

The initial software development for the UWC was written in Visual Basic using Microsoft's ASP.Net Web Forms development framework interfacing with a Microsoft SQL Server database. The application was hosted by a local cloud service provider free of charge.

The web application the previous developers created was intended to be a prototype of what the UWC would eventually become. According to Uni-SPIRE's marketing material, final deliverables created for the prototype were:

- A K-12 formative assessment continuum of 14 levels and 42 growth points, correlated to the Common Core
- Goal setting for writing improvement based on ongoing assessment.
- Student self-evaluation checklists and graphic organizers for all text types
- Students' and parents' online access to teachers' writing feedback
- Teaching strategies and learning engagements that match learners' goals
- Help videos to guide teachers in technical and content support
- Online professional development available at point-of-need.
- An electronic portfolio from Kindergarten to graduation with seamless vertical articulation K-12.
- Administrative data reports on individual students, classes and grade levels by school or district

As an indicator of the scale of the application, there were approximately 53,387 lines of code written across 538 files.

Funding for the development of the UWC has been provided by the University of North Carolina Wilmington Research foundation, grants, the UNCW Graduate School, and the Watson College of Education. Due to the university and grants being the sole sources of funding for this product, hiring was limited to part-time, student employees. Approximately 75 University of North Carolina Wilmington graduate and undergraduate students have contributed to this project in various ways. Of these students, there have been four graduate students from the Masters of Science in Computer Science and Information Systems (MSCSIS) and eight undergraduate students that have contributed in the development of web application's code base. The development of the UWC web application has always been led a single graduate student developer. While this project has provided rich, applied learning experiences to all of these students beyond what they received from their coursework, each developer's time came to an end upon graduation. After a graduate student would leave, a new graduate student with a different skillset and level of experience would take their place. Very little communication took place during the transition from one graduate student to the next about the implementation details of the UWC. This led to a highly segmented code-base with very little reuse or architectural design. Additionally, the lack of communication between project leads caused features that were only half-developed after a student developer left.

I began working with Uni-SPIRE in May, 2014 after the prior developers had left the company. I was given no instruction about the state of the project other than what Dr. Powell, who has no experience with software development, was able to provide me. Often, her feedback was limited to only identifying bugs, without being able to explain the specific programming factors that led to the introduction of said bugs. The first six months of my employment at Uni-

SPIRE were spent debugging and understanding the relationships in the software product before new feature development could begin.

2.4. Current Situation

In August, 2015, Dr. Powell and I hired three part-time undergraduate students to join the development team. Each of these students was selected due to their specialization in specific aspects of web application development. One of the students was a front-end developer with a significant portfolio of web sites that showcased his user-interface design abilities. Another student participated in a summer internship with Uni-SPIRE, during which he developed significant experience with back-end development. The final student had a background in system administration. The structure of the development team was created to utilize specialization of each developer's expertise.

2.5. Recommended Enhancements

A study of the UWC completed for this capstone revealed a lack of standard development practices and the short duration of students' involvement in the project led to various issues. As a result of this study, the parts of Uni-SPIRE's goals related to software development practices for the UWC were adjusted to include the following:

- Establish development practices to improve source code maintainability
- Foster programming accuracy
- Facilitate communication across business units

- Establish a scalable infrastructure
- Decrease learning barriers for new developers

Each of these current major technology challenges are described in the following section.

Establishing development practices to improve source code maintainability is among the most important and critical goals for UWC. Uni-SPIRE has experienced problems associated with source code that is difficult to maintain. Development time consistently exceeds predictions and unintentional coupling regularly leads to the introduction of new bugs.

The goal, *foster programming accuracy*, aims to remedy the problem that developers do not always create the desired product. Previously, enhancement projects in UWC often fail to satisfy the intended requirements. There are many factors for this phenomenon, however the result is always wasted time.

The goal *facilitate communication across business units* was selected to increase collaboration and productivity. Uni-SPIRE personnel have varying levels of technical experience in each technology employed in UWC. This lack of shared knowledge presents a boundary for communication across business units. This primarily affects the way the management and development teams communicate.

The goal *establish a scalable infrastructure* is absolutely necessary for Uni-SPIRE to grow from a small company to a medium-size company. A prototype of the UWC was developed on a cloud infrastructure provided free of charge from a local cloud service provider. This had always been intended to be a temporary infrastructure for the UWC. Uni-SPIRE plans to replace the use of this prototype infrastructure with one that will grow with the company.

The final goal of Uni-SPIRE, *decrease learning barriers for new developers*, also deals with scalability concerns. The decision to pursue this goal is important due to the unique

development model of the company. As Uni-SPIRE could only hire UNCW students at the time of this project, they were all expected to leave the company. It was projected that new students would need to be trained. Additionally, as the company grows, it will require a full development team. This goal was set to reduce learning curve for new employees.

With this capstone project, Uni-SPIRE improved its software development processes to accomplish the aforementioned strategic goals. These software development process enhancements contributed to the realization of each component of the strategic technology goals. Key software development process enhancements are:

- Implementation of a scalable cloud infrastructure
- Adoption of the ASP MVC C# framework and standard development practices
- Inclusion of systems analysis for future enhancements
- Project management enhancements

Each enhancement will be described in depth in Chapter 3 of this document. Table 1 matches the specific enhancements to the technology goals.

		Goals				
		Maintainability	Accuracy	Communication	Scalability	Learning Barriers
Enhancements	Development Practices	X			X	X
	Cloud Infrastructure				X	
	Systems Analysis	X	X	X		X
	Project Management		X	X		X

Table 1: Technology Goals/Enhancements Relationships

3. Deliverables

This capstone project was comprised of five primary components. These are: (1) identifying and defining an appropriate scope as it pertains to the quantity of the system rewrite, (2) the development of software deliverables, (3) improving the infrastructure of the system, (4) systems analysis for future software deliverables, and (5) improving project management procedures.

3.1.Declaration of Scope

An organized, systematic approach to the selection of improvements to the UWC and the associated management policies that satisfied the aforementioned limitations was deemed favorable to chaotically performing random improvements. For this reason, the declaration of this capstone project's scope was the first component of the project.

This capstone project is intended to create a new foundation for the UWC that contains complete implementations of the web application's key features. The new foundation of the UWC will then create a starting point for future development to occur without the hindrances of the original web application's implementation and management procedures. While having a complete application with perfect management procedures would be the optimal result of this project, it would be unachievable given the inherent time and budgetary restrictions of a capstone project. The available time for this project was limited to six months. The budget for software development staff limited our team to four student developers, and the budget for software and hardware limited the project to only expenditures on essential items. These limitations made it

necessary to identify an appropriate scope for software development, infrastructure improvements, systems analysis for future development, and project management improvements.

The software development component of this capstone project required more scope limitations than all of the other project components because of the size of the UWC's original implementation. The original implementation had been under development for four years prior to the initiation of this capstone project, resulting in a large, feature-rich code base. Given the limited amount of time available for this project and the lack of the development teams expertise regarding the newly introduced software development platform, it was apparent from the onset of this project that the scope of the software development deliverable would be greatly limited.

The process of selecting an appropriate scope for the software development deliverable was determined by collaboration between Dr. Powell and me. As Dr. Powell was the CEO of Uni-SPIRE, it was within her responsibilities to determine the core features necessary for the UWC. As the project manager, it was my responsibility to assess which software enhancements were realistically achievable considering the available resources. I had to not only estimate the amount of developmental effort required by each enhancement, but also approximate the rate at which the software development team would be able to produce the necessary source code, while taking into account the investment required to train the staff on new developmental practices. I made these estimations by relying on my previous experiences with the product and the development team. The process of estimating these project requirements allowed me to provide feedback to the CEO as to which software enhancements could be accomplished complete to create an appropriate foundation for the new implementation of the UWC.

To determine the scope of the software development component of this project, we first identified what groups of features existed within the prototype application. We then determined

which groups of features were essential for the foundation of the UWC. The feature groups found in the prototype are:

- application management features
- evaluation features
- reporting features.

After the major feature groups were identified, specific features from each group were selected for implementation. The implementation details of the features were then evaluated. If the original implementation sufficiently accomplished its goal, the scope of the feature development in the new implementation was to reproduce the feature in the new application on the new software platform. If the original implementation did not sufficiently accomplish its goal, the scope of the feature development was to redesign the feature in the new application.

Application management features are features that do not directly provide value to the application, but are necessary to create a foundation for the features that do provide value. For example, before a teacher can evaluate a student's paper, the data for the student must exist within the system. In this example, all features that correspond the management of a student's data are considered application management features. For example, the data related to a student includes the student's name, identification number, and which course the student is taking. Application management features were separated into two sub-groups, internal and external application management features. Internal application management features are components of the application management group that were not directly accessed by end-users. The internal application management features contained within the scope of the project are:

- User authentication and authorization
- User session data management

External application management features are components of the application management group with which the end-users interact directly. The external application management features contained within the scope of this project are:

- Student data management
- Student account management
- Parent account management
- Classroom management
- School roster management

Reproduction of these seven features were determined to provide the foundation for the application management features group.

Evaluation features compose the primary value proposition of the UWC as they deliver the formative assessment platform and supplementary tools to end-users. All of the evaluation features from the original implementation of the UWC were included in the scope of this project. The evaluation features are:

- Evaluate papers using the formative assessment platform (the intellectual capital of this company)
- Upload and manage student papers
- Set class goals
- Set individual student goals
- Deliver a lesson planning framework

The *evaluate papers using the formative assessment platform* feature required a complete redesign to improve the user interface and make the feature data-driven, i.e. the functionality is determined by data, rather than the source code. The redesign of the user interface was an

additional requirement because the original implementation of this feature was difficult to use effectively. The data-driven aspect of this feature stems from potential changes that may be made to the writing guidelines that would not be achievable with the original implementation. The other four features in the evaluation group are reproductions of the origin implementation.

The final group of features if reporting. There are three reporting features that are within the scope of this project:

- Class data reports
- Benchmark reports
- Individual student data reports

Each of these features are used by multiple types of users, which is explained via a use case diagram that may be found in appendix O. Each of these features are implemented separately for each user type as implementation details vary between the specific reports. All three features of the reporting group are reproductions of the original implementation.

The scope of infrastructure improvements includes the selection of cloud platforms for the application and database servers, configuring the cloud platforms, and deploying the application components to the cloud platforms.

The scope of the systems analysis component of this project is limited only to features that were not fully implemented in the original implementation of The UWC. These new features were under development at the beginning of this capstone project, however development was discontinued to allocate resources for the implementation of the new version of the UWC. The systems analysis is included as a deliverable for this capstone project because, for each undeveloped feature, there is no current implementation that future developers can reproduce. The features that within the scope for the systems analysis portion of this project are:

- Delivery of teaching strategies and learning engagements
- Student self-evaluation
- Student peer-evaluation

The scope of the systems analysis deliverables pertaining to this project include the introduction of a version control system, the establishment of a developer workflow, and the investigation of project management tools.

3.2.Software Development

The motivation for including a software development component in this capstone project is to provide an improved foundation for the UWC with significant benefits over the original implementation. The application's foundation was improved through the implementation of Microsoft's ASP MVC framework using C#. The reason for this new platform are to make the application easier to maintain, to increase the scalability of the application, and to decrease learning barriers for new developers.

The transition from Microsoft's ASP Web Forms framework using Visual Basic to Microsoft's ASP MVC required the entire code base to be rewritten. The adoption of a different framework prevented the translation of the Visual Basic code into C# code, which would otherwise be possible, because the structures of the platforms are completely different. The structure of ASP MVC improved the source code by enforcing a consistent programming pattern and establishing a separation of concerns throughout the application.

The consistent programming pattern enforced by ASP MVC greatly enhanced the UWC. While the structure of ASP Web Forms simplified the process prototyping the web applications,

it did not force consistent development practices. As the development team within Uni-SPIRE has not been consistent, neither have the development practices employed in the creation of the UWC's source code. The structure of ASP MVC, however, demands a specific programming pattern to be adhered to throughout the entirety of an application in order for features like routing to operate properly. As Uni-SPIRE's development team is expected to continue to change, a required programming pattern will enforce consistency in future development.

The separation of concerns within the MVC pattern derives from the types of objects from which the pattern's name originates: Models, Views, and Controllers. Each type of object within MVC serves a separate purpose: Models represent the application's data; views define how the data will be presented; and controllers handle user interaction. The separation of concerns provided by ASP MVC created an improved implementation of the UWC by aligning with the structure of the Uni-SPIRE development team, reducing the complexity of each component of the source code, and allowing the reuse of code throughout the application.

As discussed in section 2.4 of this document, the development team of Uni-SPIRE is composed of developers that specialize in front-end or back-end development. The MVC model fits this composition very well because it separates the front-end components, the views, from the back end components, the models and controllers. This improves the implementation of the UWC because it allows developers with different skill sets to develop only pieces that align with their skill sets.

The separation of concerns provided by the MVC pattern reduces the complexity of each component of the application's source code. In the original implementation of the UWC, each web form had a single class that handled the modelling of data, the presentation of data, and responses to user interaction; which led to massive, complex code files that were difficult to

enhance or debug. In MVC, each of these functions are provided by distinct, independent objects. This improves the implementation of the UWC because it is easier to enhance features and remove bugs.

The original implementation of the UWC was difficult to maintain due to the lack of code reuse throughout the application. There are many instances of identical code written in multiple places within the original implementation, which increases the complexity of maintaining the application because any change made to the code must be repeated multiple times. The separation of concerns provided by the MVC pattern reduces the amount of code repetition by limiting the scope of objects and forcing their reuse throughout the application.

The specific software development deliverables for this capstone project is described in the declaration of scope found in Section 3.1 of this document.

3.3.Infrastructure Improvements

A significant enhancement to the UWC as a result of this project is the implementation of a scalable cloud architecture. The process of enhancing the UWC's infrastructure began with assessing whether or not infrastructure improvements were necessary at this point in time. After concluding that improving the infrastructure was a necessary component of this project, the team analyzed the available options for improved infrastructure. Finally, a selection was made based on the needs of the business.

When I began working with Uni-SPIRE in May, 2014, the previous lead developer informed me that the infrastructure was designed to be temporary and that I would likely need to replace it during my employment with the company. Prior to this capstone project, there were

two distinct environments, the production environment and the development environment. Each environment presented significant concerns that prevented them from adequately solving the needs of the UWC.

The infrastructure of the production environment consisted of a virtual machine hosted by a local cloud services company. Web hosting is not a service offered by the cloud services company and was only available to Uni-SPIRE as a personal favor by the management of the cloud services company. It was agreed upon at the time of the infrastructure's implementation that the UWC would only be hosted by the cloud services company temporarily. Thus, the decision to replace the infrastructure of the production environment was made to honor the previous agreement.

The infrastructure of the development environment was originally used as the production environment for the UWC. The development environment was composed of a shared web server and a shared server from a popular web hosting provider. Prior to my employment with Uni-SPIRE, the infrastructure was deemed ineffective at hosting the platform and was replaced by the production environment. While scalability concerns are not necessarily an issue with a development environment, the driving factor for replacing this infrastructure was the lack of congruency between the infrastructure of the production and development environments. This meant the environments could not have identical configurations, which could lead to the introduction of bugs to the production environment that were not present in the development environment. The driving factor to improve the infrastructure of the development environment during this capstone project was to provide developers with a development environment that is similar to the production environment.

The objective for making enhancements to the infrastructure of the UWC is to improve the scalability of the application. Therefore, a potential replacement infrastructure for the application could only be considered if it addressed the issue of scalability. Two cloud service providers offer such services: Amazon Web Services and Microsoft Azure.

Both cloud service providers offer Infrastructure as a Service (IaaS) and Platform as a Service (PaaS) products. An IaaS provides virtualized hardware that can be scaled on demand but requires manual configuration to deploy an application, similar to a traditional web server. A PaaS is similar to an IaaS, but provides additional tools to simplify application deployment. To reduce development time, our team elected to implement a PaaS architecture.

Microsoft Azure was selected as the new infrastructure after the completion of a pricing comparison. Additionally, the UWC's application framework and database are both Microsoft products. Uni-SPIREs previous positive experiences with Microsoft products also influenced the decision to implement their cloud services to host the application.

3.4.Systems Analysis

The systems analysis requirements for this capstone project were greatly reduced due to the four years of prototyping that occurred before this project began. As a result of the massive prototyping effort, the majority of the system's requirements were already adequately implemented. Thus, the systems analysis for this project was limited to enhancements to the database schema before feature development and the analysis of features that had not been implemented in the original implementation of the UWC.

Performing a rewrite of the code base for the UWC granted us the unique ability to alter the schema of the database without introducing bugs to the application. The previous database schema had a few short-comings that impacted the effectiveness of the application. The actions of the development team to address these short-comings are:

- Normalize and extend of the tables that support key features
- Alter the cardinality of relationships
- Rename tables

When this capstone project began, there were features under development in the original implementation of the UWC. Development of these features was abandoned in order to allocate resources for the completion of this system rewrite. Additionally, completing the development of these features on the original implementation of the UWC was unnecessary because the application is going to be replaced. Unfortunately, as these features had not yet been implemented, systems analysis needed to be performed in order for future developers to complete them on the new implementation of the UWC.

The systems analysis methodology used to analyze the features that had not been prototyped is composed of three sections:

- Entity Relationship Diagrams to detail the data model of each feature
- Use Case Diagrams to explain which actions users can perform regarding each feature
- An informal document discussing implementation considerations

The features that were analyzed during this process are:

- Teaching strategies and learning engagements
- Student self-evaluation
- Student peer-evaluation

3.5.Project Management

The project management processes previously used by Uni-SPIRE, while appropriate at the time, required significant enhancements in order to accommodate the growth of the company's development team as well as to establish a foundation for the management of future projects. The motivation for the improvement of Uni-SPIRE's project management methods is to improve the accuracy of development, facilitate communication between business units, and decrease learning barriers for new developers. Deliverables for this capstone project pertaining to the enhancement of project management methodologies include:

- the implementation of version control
- the formalization of a software development workflow
- the investigation of project management tools.

3.6. Intended Project Plan

To complete all of the deliverables of this capstone project, it was necessary to establish a project plan.

The development team at Uni-SPIRE was comprised of graduate and undergraduate students which posed significant challenges to the establishment of a project plan. Challenges arose from the need to train developers on the new platform and the team's lack of software development experience.

At the onset of this project, none of the members of Uni-SPIRE's software development team had any experience with ASP MVC. Additionally, only one developer had previous

experience with the C# language. Every employee needed to be trained on the new language and platform to the extent required for them to fill their role in the project. The need for this training led to the conclusion that development would take a longer period of time during the early parts of this project than in later parts of this project. To account for this lack of experience and to provide training opportunities to the development team, the least complex features were planned to be completed first.

Another challenge faced when devising this project plan was the lack of software development experience of the development team. The team's lack of experience meant that all code created would need to be analyzed and appropriate changes had to be made in order to assure that it produced a high quality code base. The process of analyzing the code and making changes could add a significant amount of time for the development of the new implementation of the UWC. The actual amount of time that would be required for this process was unknown when this project plan was created.

Taking into account the challenges for creating this project plan, it was determined that the best approach was to avoid setting specific dates that features would be completed upfront. Instead, the team elected to set a series of benchmarks to be accomplished sequentially. Table 2 lists the software benchmarks.

Order	Software Benchmark
1	Internal Application Management
2	External Application Management
3	Evaluation
4	Class Data Reports
5	Benchmark Reports
6	Individual Student Data Reports

Table 2. Software Benchmarks

4. Tasks Completed

The purpose of this capstone project was to systematically enhance the implementation of the UWC. This was accomplished through the development of a new code base for the application, the improvement of the application's underlying infrastructure, the completion of systems analysis, and the adoption of new project management methodologies.

4.1. Software Development Enhancements

The software development component was the most time consuming portion of this capstone project. The software development deliverables enhanced the UWC by providing an improved foundation for the application through the implementation of superior development practices alongside a programming framework that better aligns with the structure of the development team. This section describes the methodology used to develop the new implementation of the UWC and describes the deliverables that were completed using the implementation methodology.

4.1.1. Implementation Methodology

The development process used to implement features in the new version of the UWC had to be reconsidered for this project in order to adhere to the MVC pattern. The development methodology was divided into two major sections, back-end and front-end development

processes, to utilize the separation of concerns inherent in the MVC pattern and to optimize development according to the structure of the development team.

Back end development process consisted of four distinct steps. Each step is dependent on the step before it, thus the structure of this section accurately represents the workflow performed by the back-end developers. The four steps are:

1. Creating model classes
2. Develop logic to handle the two-way binding between each model and its corresponding database entity
3. Aggregating models into view-models
4. Implementing logic to respond to user actions

The first step in the back-end development process is to create a representation of the application's data that can be used within the application. Representations of the application's data are created through the construction of model classes. The structure of each model class was derived from the entities within the database. For example, the AcademicYearModel class contains the properties ID and YearDescription just as the AcademicYear table in the database contains the columns ID and YearDescription.

The second step in the back-end development process is the development of logic to handle the two-way binding between each model and its corresponding database entity. The logic involved in this process makes up the Data Access Layer (DAL) of the application. This step is necessary because it allows the application to store data in the database as well as retrieve data from the database. To accomplish this, the back-end development team created a DAL class for each model. Each DAL class is located within the code file for the related model. Each DAL class provides methods that interface with stored procedures from the database. The stored

procedures called within the methods of the DAL classes were created for use in the original implementation of the UWC and were reused for the new implementation.

The third step in the back-end development process is to aggregate models into view-models. A view-model is a single class that contains all of the models that will be consumed by a view. Aggregating the models into view-models simplifies the transfer of data from controller to view by providing a single access point for the entirety of the view's data. For example, a view may need to show a detailed list of the courses a teacher has instructed filtered by academic year. To accomplish this goal, the view would contain a drop-down list of academic years and a table showing the associated courses. The view would need academic year and course objects. The implementation of a view-model allows a single access point for both types of objects used in the view.

The final step in the back-end development process is to implement the logic to respond to user interaction. The logic to respond to user interaction is implemented within a controller class. Given the above example of a view that shows a detailed list of courses a teacher has instructed filtered by academic year, there are various user interactions that need to be handled within the controller. When the view is accessed for the first time, the view-model needs to be instantiated and provided with the default data. In this example, the default data is the models of available academic years for filtering. The user would then select an option from the drop-down list. The controller would then need to put the models for the appropriate courses into the view-model.

Front-end development for the new implementation of the UWC was composed of the development two types of features: Features that consume back-end components, and features that are independent from back-end development.

Front-end development of features that consume back-end components comprise the majority of front-end development for the new implementation of the UWC. The developing these involves creating graphical elements to display data in a consistent manner and provide opportunities for user interaction via HTML. These features could not begin until all four steps of the back-end development process were completed for the feature because they relied on the data modelling and user interaction responses implemented in the back-end development process.

While the front-end development team awaited resources from back-end development team, they developed features that were independent from back-end development. These features consisted of:

- Making improvements to the user interface
- Developing static, public facing webpages
- Developing static webpages to serve teacher resources

Improvements made to the user interface of the UWC involve the implementation of Twitter Bootstrap. Twitter Bootstrap is a framework for developing mobile-friendly web pages. Twitter Bootstrap requires pages to be implemented in specific ways, meaning the front-end development team had to redesign the structure of every page. The navigation menu of the original implementation of the UWC contained up to four levels of sub-menus. To implement Twitter Bootstrap, the number of levels of sub-menus could not exceed two. As a result, the front-end development team had to redesign the navigation structure for the UWC.

An important piece of the UWC is the group of public-facing webpages that provide non-users with details about the product. These webpages are static, meaning they do not require any data modelling or user interaction. As a result, development of these pages did not require back-

end development. These webpages are basic reproductions of the public web pages from the original implementation of the UWC.

Additional static webpages are required within the application to provide resources to teachers. These resources are PDF files, video tutorials, and relevant links. These webpages were also reproductions of webpages from the original implementation of the UWC.

4.1.2 Software Development Deliverables Completed

The software development deliverables completed for this capstone project are listed in section 3.1 of this document. Many of the software development deliverables were simple reproduction of features from the original implementation of the UWC. This section discusses development that is specific to the new implementation of the UWC; omitting all development activities that were reproductions from the original implementation. During this project, the development of application management, evaluation, and reporting features was completed.

The majority of the application management features that were developed during this capstone project were reproductions of the original implementation. The application management features that were not reconstructed are the authentication and authorization features.

User authentication in the original implementation of the UWC required significant enhancements. The new implementation of authentication implements a third party encryption library called BCrypt to encrypt user passwords before they are stored in the database, whereas the original implementation did not encrypt user passwords at any point. The inclusion of encryption for user passwords created the need for authentication functionality that did not exist

in the original implementation of the UWC. This authentication functionality can be observed in the LoginModel class diagram found in appendix E.

User authorization in the new implementation of the UWC utilizes both browser cookies and server-side session data. Upon authentication, a cookie is created within the browser to indicate to the application that authentication has occurred which is the first step of authentication. When the user requests a restricted page, the browser transmits the authentication cookie to the server. This authentication cookie allows the server to identify which user sent the request. This information is then used to retrieve the user's role from the application's session object which is used to perform authorization.

All of the evaluation features that were developed during this capstone project were reproductions of the original implementation with the exception of the evaluate papers using the formative assessment platform feature. The evaluate papers feature was redesigned to meet both aesthetic and functional requirements.

The original implementation of the evaluate papers feature was not aesthetically pleasing. The two main factors that led to the unappealing appearance of the evaluate papers feature were the reliance on page post-backs and the inefficient use of space on the screen. For reference, a screenshot of the original implementation is located in example 3 of appendix M.

The original implementation of the evaluate papers feature relied on page post-backs to respond to user actions. When a user interacts with the evaluate papers feature, they view data regarding evaluation for each of the ten elements of writing. The previous implementation contained a link button for each element of writing. When the user clicked a link button, a post-back was initiated, causing the server to recreate the page with the details for the writing element visible. This was an issue because when the page was refreshed the browser took a noticeable

amount of time to render the elements, causing the page to flash. The new implementation of the evaluate papers feature avoids this problem by implementing a small JavaScript application within the browser, eliminating the need for post-backs. The JavaScript application interfaces with the rest of the application using a small REST API embedded within the application, which is further explained in appendix I.

The original implementation of the evaluate papers feature inefficiently used the space on the screen. When the writing guidelines associated to an element of writing were displayed, the entire text was made visible. This was an issue because many of the writing guidelines were exceptionally long, especially for high-school level evaluations. To use the space on the page more efficiently, the writing guidelines scroll independently of the other page elements, forcing them to use a pre-set amount of space regardless of text length.

The evaluation feature's redesign also resolved functional issues inherent within the original implementation. Functional issues that were resolved with the new implementation of the evaluation feature are: The inability to navigate throughout all developmental levels, the feature's inability to adapt to changes in the data model, and reliance on the outdated database schema.

The original implementation of the evaluate papers feature was only able to display writing guidelines from five developmental levels. The levels that were displayed were the level that was proficient at the student's developmental level, the two levels prior to proficient, and the next two levels past proficient. This resulted in the inability for a teacher to evaluate an element of writing if the correct level was not included in the five levels that were displayed. To work around this, teachers were forced to create a new course within the application to evaluate students that were far beyond their expected developmental level. The system treated the two

courses as distinct courses which caused issues with reporting. The new implementation of the evaluate papers feature allows the teacher to evaluate an element of writing at any developmental level, eliminating the need to create additional coursed within the system.

A proposed enhancement to the evaluate a paper feature is to allow schools to define custom writing guidelines to be evaluated alongside the provided writing guidelines. The rigid structure of the old implementation made this an impossibility. The new implementation, however, generates the user interfaced directly from the data it receives from the API. If a school were to add custom writing guidelines, the user interface will adapt to include the additional writing guidelines in the evaluation process.

The database schema that supports the evaluation feature has been replaced during this project. The alteration to the database schema includes the ability to filter writing guidelines according to the type of paper, such as an argumentative or informative paper, that is being evaluated. The new database schema required changes in the new implementation. The database schema change is diagrammed in appendix J of this document.

While all three reporting features were reproductions of the original implementations, the back-end components of the benchmark reports were redesigned to simplify implementation.

Benchmark reports are special reports that are available to teachers, school administrators, and district administrators. Benchmark reports provide an overview of student performance in a course or school. Each type of user is able to view different benchmark reports. Teachers can only view benchmark reports for the courses they teach; school administrators can view benchmark reports for any course in the school as well as aggregate benchmark reports for courses in specific grade levels or the entire school; and district administrators can view the same benchmark reports as school administrators, but have access to the benchmark reports for each

school in the district. Every benchmark report has the same structure; the only difference is the data that composes the report. For this reason, we were able to create a single benchmark report class and populate it with different methods within the DAL based on which report the user requests.

4.2. Infrastructure Improvements

The improvements made to the infrastructure of the UWC were of two separate technologies: the web server and the database server. Both of the technologies infrastructures were improved by utilizing Microsoft's Platform as a Service (PaaS), Azure. The implementation of PaaS solutions is intended to increase the scalability of the application, as the performance-impacting resources of the underlying infrastructure can be scaled up on-demand. The PaaS solutions for the web server and database servers are Azure Application Services and Azure Database, respectively.

Azure Application Services were implemented in both the production and development environments for the UWC. Azure Application Services provides a free-tier product, allowing the creation of additional application services at no additional charge. As the demand on the production server increases, the application service will need to be scaled up to a tier that offers higher performance.

Azure Database was implemented as a central data storage solution for both the production and development environment. Azure Database does not have a free-tier product. To avoid additional expenditures, a single database was implemented to support both the development and production environments. As Uni-SPIRE grows, the development team should

implement an additional database for the development environment. Additionally, Azure Database has feature limitations when compared with a traditional SQL Server tool. These limitations were analyzed before implementation and the results of the analysis are addressed in Appendix K of this document.

4.3. Systems Analysis

The systems analysis component of this capstone projects consists of two sections, schema enhancements and future development analysis. Schema enhancements were implemented before the associated software development components were created, to reduce the impact of the alterations. The systems analysis for future development was completed throughout the course of the product, as none of the software development deliverables were dependent on this analysis.

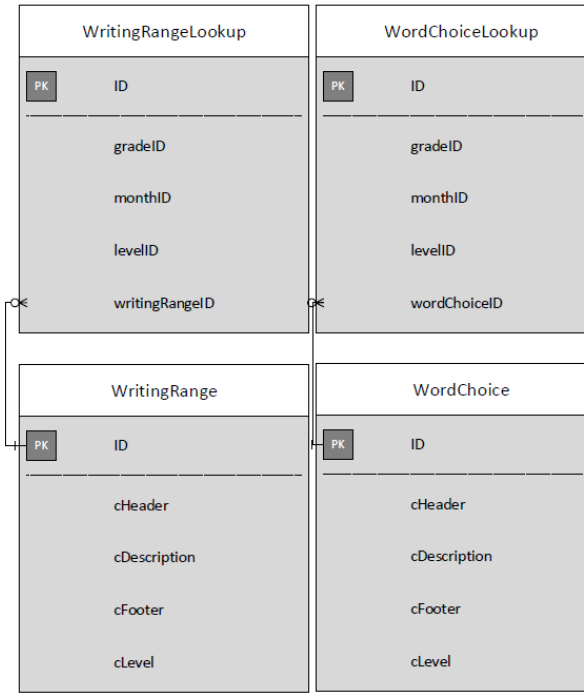
4.3.1. Schema Enhancements

This section will describe the categories of schema enhancements and provide an example for each one. The categories for schema enhancement deliverables for this capstone project were:

- Normalize and extend of the tables that support key features
- Alter the cardinality of relationships
- Rename tables

The process of normalizing and extending the tables that support key features was necessary because the requirements of the key features were not adequately defined prior to their implementation in the original application. Many of these concerns have been known for some time, however they had not been addressed due to the amount of work that would have been required to update the application to support the changes. While there were a number of sections of the database schema that required a redesign, this paper will only describe the process in the context of the storage of writing guidelines. Additional details about this example can be found in appendix J.

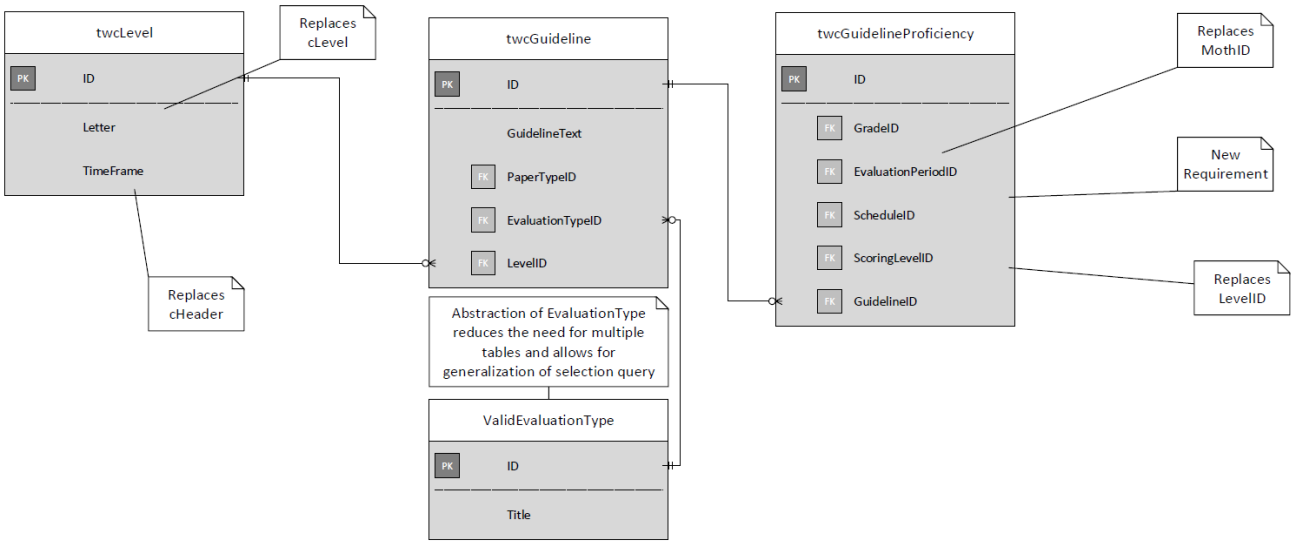
The Entity Relationship Diagram (ERD) shown below demonstrates the storage of the writing guidelines in the original implementation of the UWC.



This ERD demonstrates only four of the twenty tables used to represent the data behind the writing guidelines feature, however the remaining tables follow the same pattern: a lookup table associated with a writing guideline table. Each lookup table contains the following foreign key

fields gradeID, monthID, and levelID; as well as another field that refers to the primary key of the associated writing guideline table. Each writing guideline table contained the fields cHeader, cDescription, cFooter, and cLevel. None of these fields, except for cLevel, had an adequately descriptive title as they merely describe their location on the associated form as opposed to the actual data they represent. Additional requirements that this schema was unable to fulfil were the division of each guideline by paper type as well as the addition of a schedule identifier into the lookup process.

A new schema, demonstrated by the ERD below, was implemented to resolve these issues.



The twcLevel table replaces the cHeader and cLevel fields of the previous schema's data tables. Additionally, this solves data integrity concerns as letter and timeframe are interdependent. Future developers will not be able to input an invalid Letter/Timeframe pair.

The twcGuideline table works with the ValidEvaluationType table to resolve the issue of having ten separate data tables. The ValidEvaluationType table contains information about

which element of writing each guideline is associated to. The twcGuideline table now contains an PaperTypeID field to resolve the requirement of dividing writing guidelines by the paper type.

The twcGuidelineProficiency table replaces the ten individual lookup tables. It maps a gade, evaluation period, schedule, and scoring level to the appropriate writing guidelines. It is important to note here the distinction between twcLevel and ScoringLevel. ScoringLevel refers to the proficiency of a writing guideline given the time elements (grade, evaluation period, and schedule), while the twcLevel refers to the developmental level of a writing guideline.

The cardinality of the relationship between schools and students within the database was original one-to-many, meaning that a school could be associated with many students, while a student could only be associated with a single school. This relationship inhibited the actualization of the previous development deliverable *an electronic portfolio from Kindergarten to graduation with seamless vertical articulation K-12* because it limited the ability to access the portfolio of a student that transitioned between schools. While it was possible to work around this problem by structuring a query that selected the school from the course associated with each of the students the paper, this was an ineffective approach the only desired information of a query was a list of the schools a student had attended. To address this problem, an association table between the school and student tables was implemented, making the cardinality between the entities many-to-many.

Various tables in the original database schema were given names that didn't accurately define the data they contained. When the schema was transferred to the new infrastructure, the names of these tables were adjusted. An example of this was the renaming of the table that contained the data related to the academic years within the database, from yearSchool to academicYear.

Each of these schema alterations were performed before software development began for the new implementation of the UWC to prevent the introduction of bugs into the new system. All data access logic related to these tables was rewritten when features that consumed the data were created.

4.3.2. Future Development

The systems analysis for future development was a process of documenting the features that had been under development when this capstone project began. This systems analysis was performed by creating an ERD to explain the data required for the feature, a use-case diagram to explain how different users would interact with the feature, and an informal document discussing implementation considerations. An example of this process is detailed in appendices G and H.

4.4. Project Management

Uni-SPIRE's project management methodologies, prior to this project, were completely acceptable for a single developer working on the UWC. In the months preceding this project, Uni-SPIRE began expanding its product development team. As a result, the project management processes became increasingly ineffective. The project management enhancement deliverables of this capstone project are the implementation of version control, the creation of a development workflow, and the investigation of project management tools.

One of the most significant challenges that arose from the inclusion of multiple developers within Uni-SPIRE's development team was the physical management of the source

code. In order to overcome this challenge, the development team implemented the use of a version control system called Git. Git provided developers with the ability to work in the same file, review what changes occurred within a file, and restore previous versions of each file.

Prior to the implementation of Git, developers were unable to effective work within the same file. If two developers wanted to work on the same file, each developer would make their own copy of the original file from the UWC's source code. Each developer would then make changes to their local copy, leading to three versions of the file; the original version and each developer's version. To update the master copy of the source code, a developer would replace the original file with their local copy. If the second developer were to then attempt to replace the original file, the prior developer's work would be erased. Git has a feature that eliminates this problem by tracking the changes made to each file and either automatically merging the new versions, or providing the developers with instructions on how to merge the files by hand.

An additional enhancement to the project management of the UWC provided by the implementation of Git is the ability to review changes that are made to any file. This is useful because if a bug is introduced to the source code, the development team can review the most recent changes to easily assess which lines of code caused the problem. An additional benefit from this feature of Git is improved accountability as Git allows developers to determine which member of the team introduced the erroneous code.

The final enhancement to the management of the UWC provided by the implementation of Git is the ability to restore previous versions of the application. If a bug were introduced to the production version of the application, the issue can be resolved quickly by restoring the most recent version of the application that did not contain the bug. This can occur before the

developers analyze the issue and develop a solution, leading to faster response time in the event of a critical issue appearing in the production application.

The introduction of Git to Uni-SPIRE's development process required the creation of a workflow detailing the appropriate use of the version control software. An activity diagram of this workflow enhancement is provided in appendix F of this document. The workflow begins with a developer selecting an appropriate branch of the source code. For example, if the developer was working on a feature that a teacher uses to add a class to the database, the developer would checkout a branch entitled Teacher_AddClass. From there, the developer would begin working on their task, saving each small section of code they create using a commit, which allows the developer to provide a message describing what changes were made. The developer would continue creating commits until they believed the task was complete, at which point they would initiate a peer review. An appropriate peer would evaluate their work and either approve it for introduction into the central code base or request changes.

Another development workflow that was created to enhance the project management of the UWC explains the process of collaboration between front-end and back-end developers to accomplish tasks. An activity diagram of this workflow is provided in appendix O of this document. The project manager initializes the workflow by identifying a programming task. The project manager would then add the task to the queue of work for either the front-end or back-end developer, depending on the task's requirements. The front-end and back-end developers would then remove a task from the queue and begin development. If the developer was unable to finish the task because it required contributions that were outside of the developer's responsibilities, for example if a back-end developer identified a necessary front-end

contribution, the developer would add the task to the appropriate queue. This process would continue until the task was complete.

The final category of project management enhancements is the investigation of project management tools. The process of investigation consisted of researching each tool and attempting the use of each tool. If the tool did not fit the development practices of the company, it was abandoned. The three project management tools that were investigated were Visual Studio Team Services, GitHub, and Slack.

Of the three development tools, Visual Studio Team Services was the only one that the team abandoned. While Visual Studio Team Services provides a variety of features to assist with project management, the only features investigated during this process was the Kanban board. The Kanban board is a tool for agile development that allows development to be divided up into different tasks. These tasks would then be assigned a status, such as backlog, in progress, or completed. While the team was initially excited to use the tool, the time overhead it required prevented developers from using it.

To replace this project management need, the team used the issues feature of the product GitHub. GitHub's issue feature allowed the team to record which tasks needed to be completed. Additionally, the issues could be tagged to create searchable groups and assigned to developers. This feature of GitHub provided the same project management functionality of Visual Studio's Team Services, but required less time to use. In addition to the issues feature, the development team adopted the central repository and automatic deployment features offered by GitHub.

The central repository tool was the feature that led to our team's investigation of GitHub. Not only does this feature contain the application's source code in a single location, but it also provides a web interface with features that support the maintenance of the central repository. For

example, GitHub allows developers to create a pull request to initiate a merge of two branches. The pull request can be assigned to a developer who can easily view the differences of each of the files of each branch using GitHub's web interface. This simplified the process of maintaining the application's source code and adhering to the development workflow.

The final project management tool provided by GitHub that was investigated and adopted during this capstone project is the automatic deployment tool. This tool interfaces directly with Azure Application Services to deploy the code whenever changes are made to a target branch. For this project, separate branches were created for both the production and development environments and GitHub automatically deployed the branches to the appropriate environment, which simplified the process of deploying the source code. (Smith, 2015)

The final project management tool that was investigated and adopted during this capstone project is a collaboration tool called Slack. Slack provides a centralized platform for the entire Uni-SPIRE staff to communicate. Employees can create channels for group discussion or engage each other with direct messages. Slack also allows users to share files and archives all of the conversations for future reference. This tool increased the team's ability to collaborate, especially when developer could not meet in person.

5. Conclusion

The key deliverables for this capstone project for the UWC were, the development of a new code base using ASP MVC and C#, the implementation of a cloud-based infrastructure, the completion of systems analysis, and the enhancement of project management processes. The completion of these deliverables provides Uni-SPIRE with a foundation to continue the development of the UWC as lead developers are replaced within the company.

While the new implementation of the UWC does not include all of the features that were developed for the prototype, the improvements made to the application during this capstone project provide significant benefits. This new foundation has is more organized, decreasing the learning curve for new developers. Additionally, there is more code reuse, due to the separation of concerns from the MVC pattern, reducing the complexity of the source code. The new cloud-based infrastructure implemented will allow Uni-SPIRE to scale up the application when user demand increases. The systems analysis that was performed will give future developers a solid understanding of incomplete features from the prototype version to simplify the application development. Finally, the enhancements made to the project management practices will allow future development teams to complete tasks more efficiently.

References:

Byham, Rick. "Azure SQL Database Transact-SQL Differences." Microsoft Docs. Microsoft, 30 Aug. 2016. Web. 18 Sept. 2016.

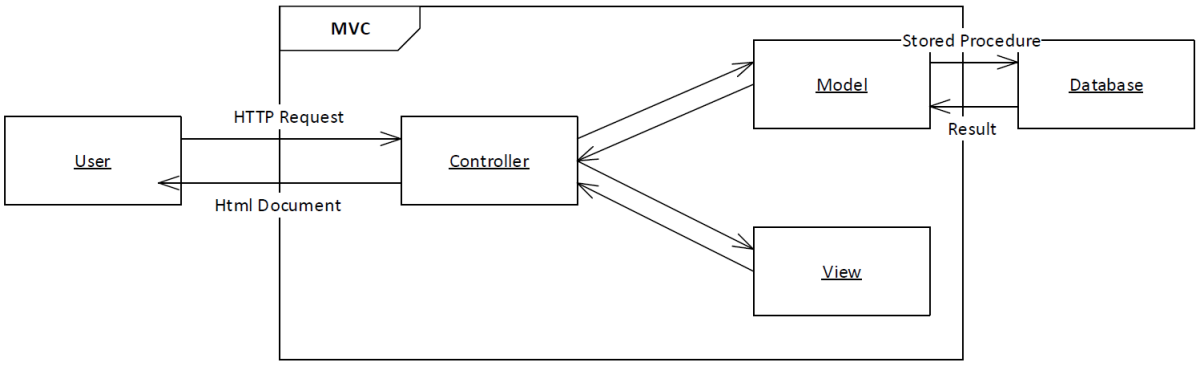
Powell, Deborah. “The Universal Writing Continuum.” *The Universal Writing Continuum.* Uni-SPIRE, n.d. Web 02 Dec. 2016.

Smith, Allen. "Automating Code Deployment with GitHub and Azure." GitHub. GitHub, Inc., 15 Sept. 2015. Web. 18 Nov. 2016.

Appendices

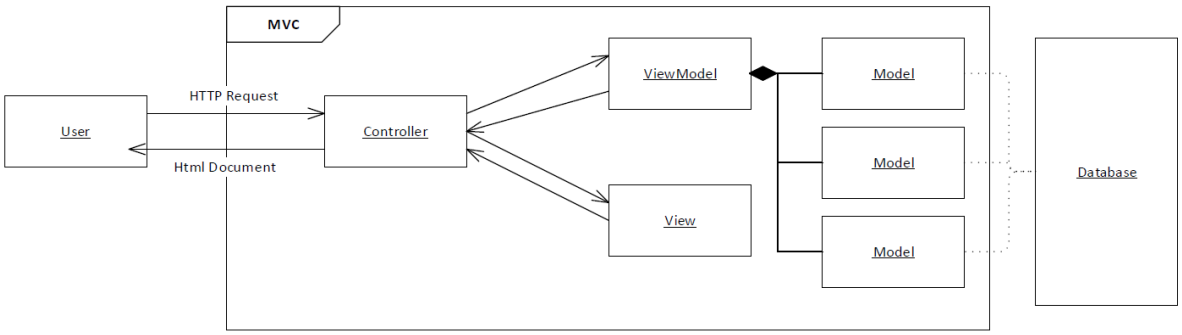
Appendix A: MVC Pattern

The following diagram explains the interaction between different aspects of the system using the Model-View-Controller pattern.



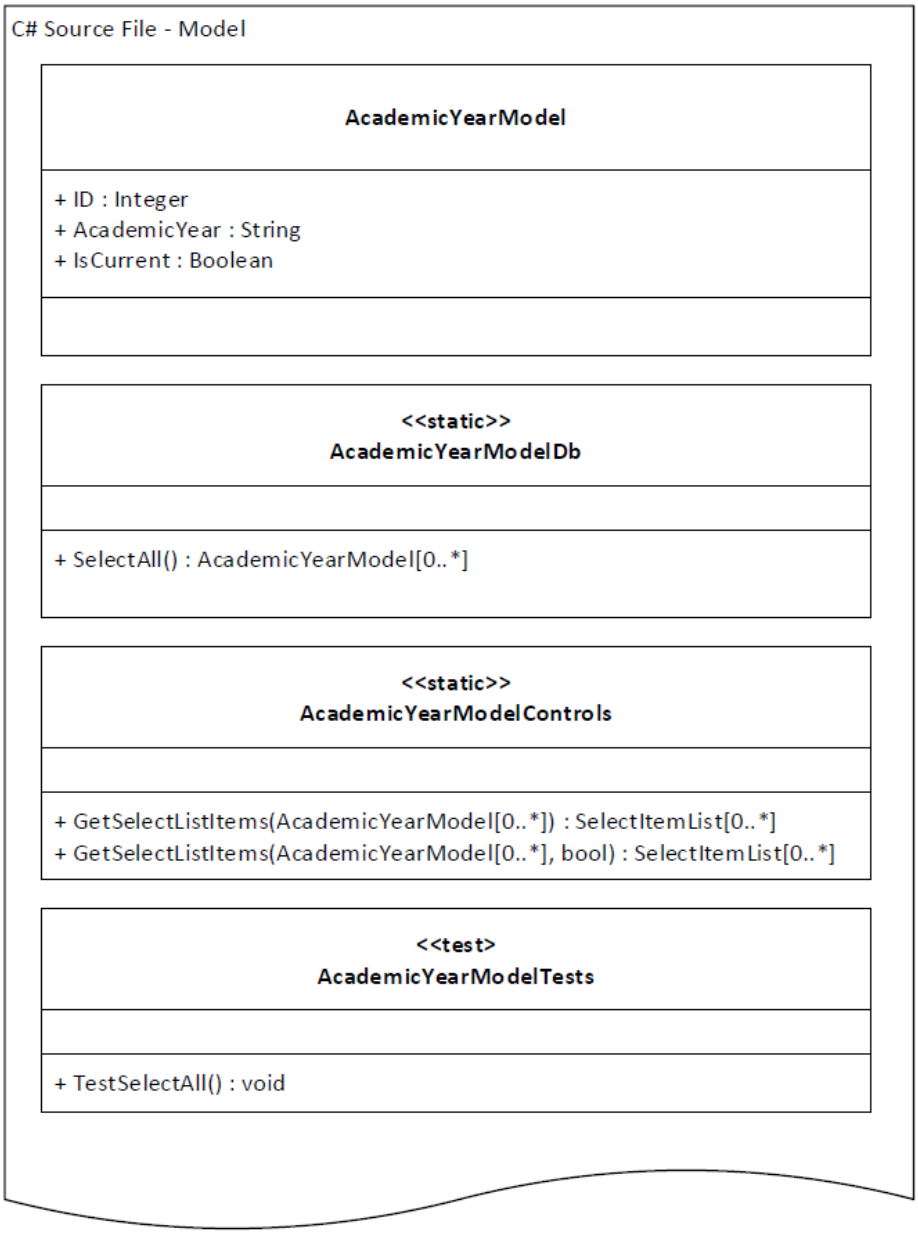
Appendix B: MVC Pattern with ViewModel

The following diagram shows the Model-View-Controller pattern with the addition of a ViewModel. This shows that a ViewModel is composed of multiple Models, while maintaining the ability to have class-specific data and methods.



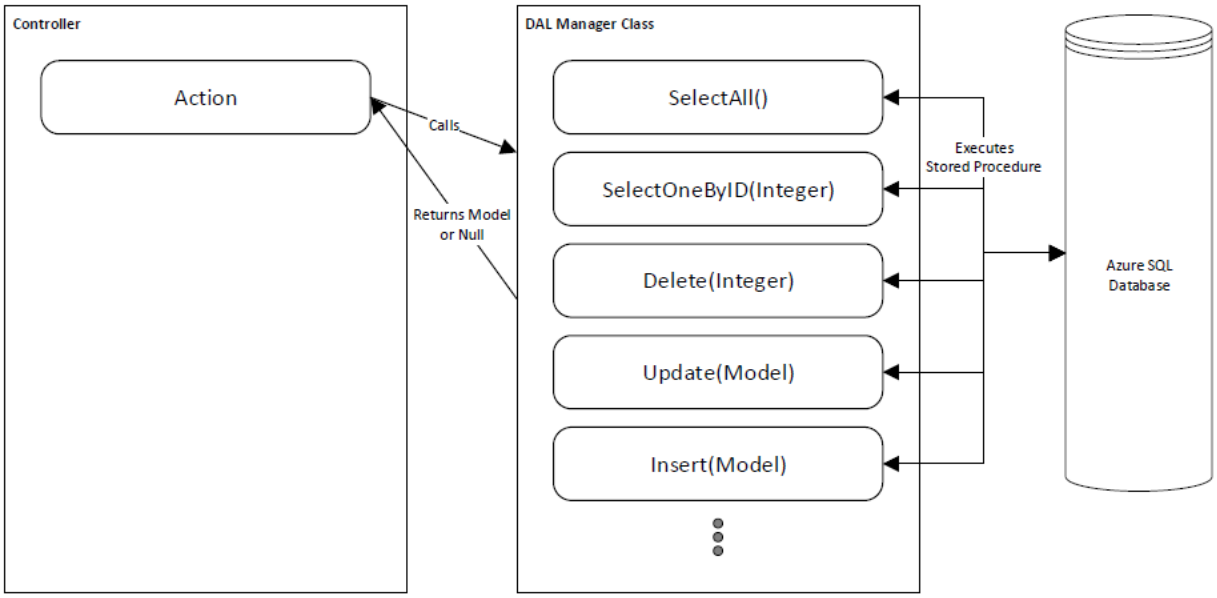
Appendix C: File-Level Organization for Models

As an effort to reduce the complexity of the source code, we implemented a programming pattern within the code files of individual Models. The following diagram shows how the class and its supporting manager and test classes are organized within a file.



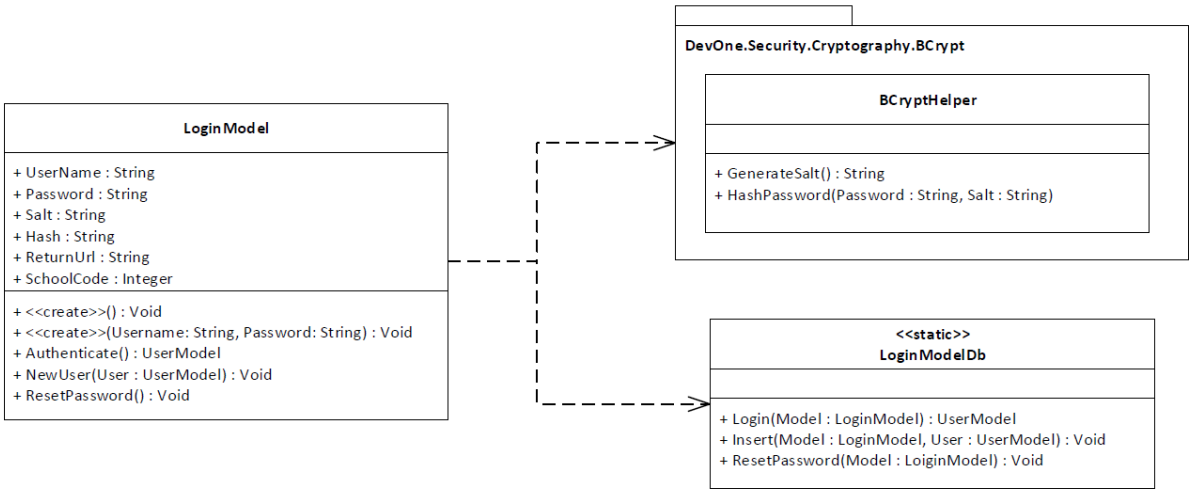
Appendix D: Data-Access-Layer Manager Pattern Diagram

The following diagram depicts the interaction between a controller and a model's DAL manager classes.



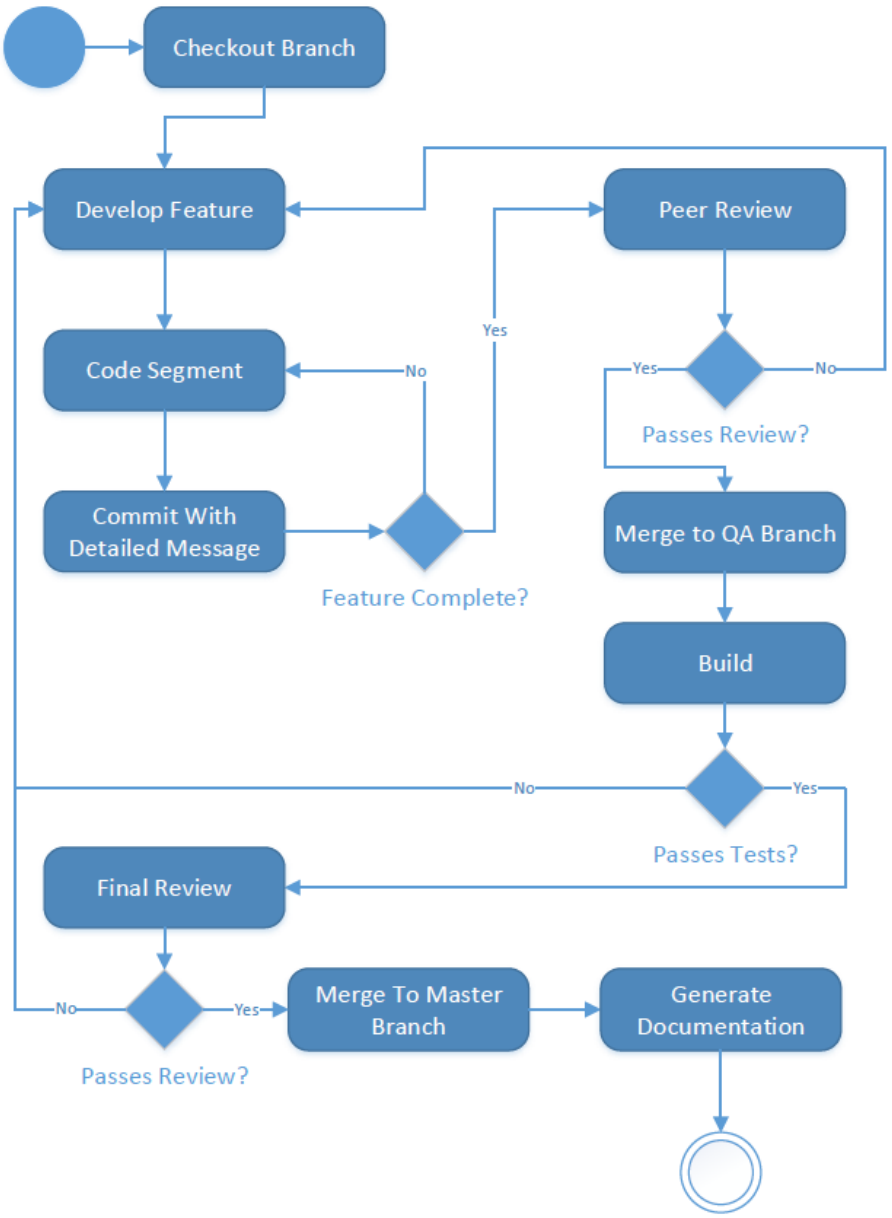
Appendix E: LoginModel Class Diagram

The following class diagram details the LoginModel, a specific type of model that breaks the pattern used within the rest of the application in order to provide the complex processes of user authentication to a various webpages. This class has a dependency to its manager class, as well as a third party encryption library, as indicated below.

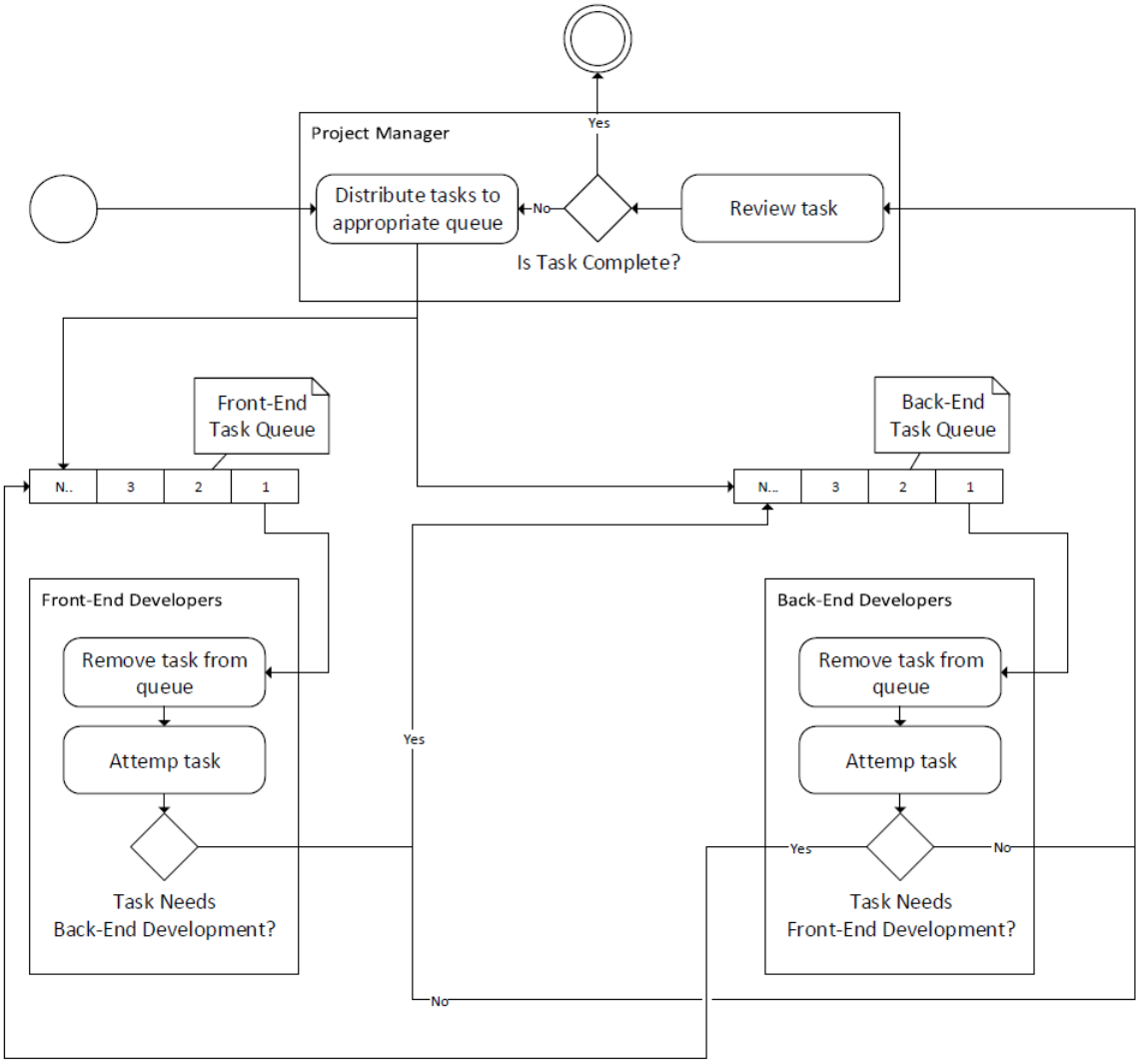


Appendix F: Developer Workflow – Version Control

The diagram below details the ideal workflow of developers within Uni-SPIRE. This is an iterative approach that is tailored to our implementation of version control software. There are multiple decision points throughout the process to verify that new code is accurate and does not inroduce any new bugs or break previous features.

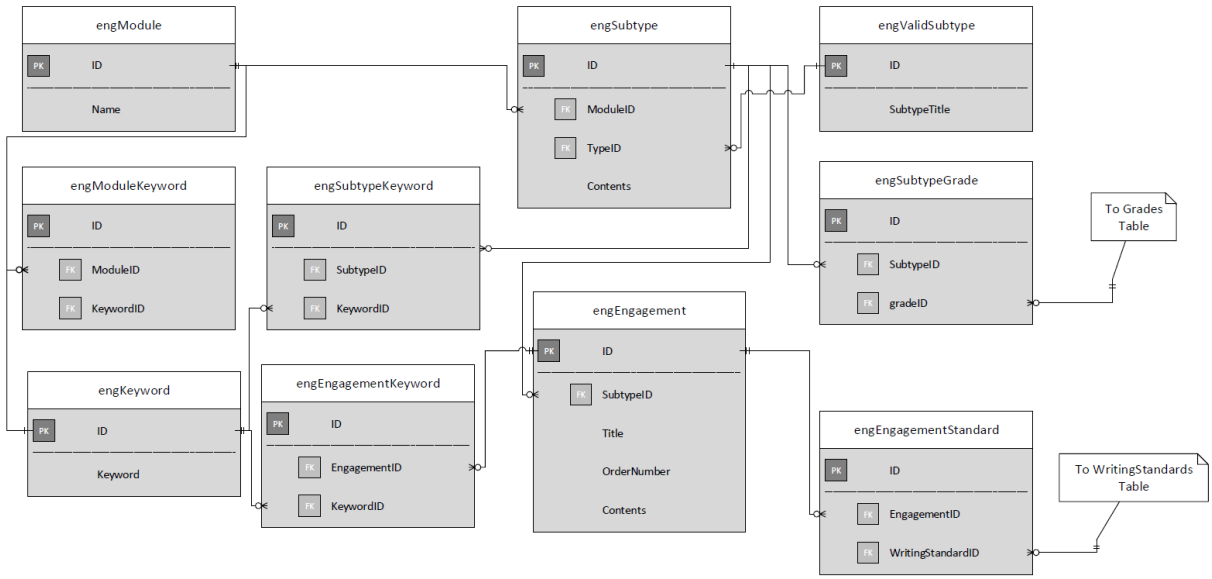


Appendix G: Developer Workflow – Task Management



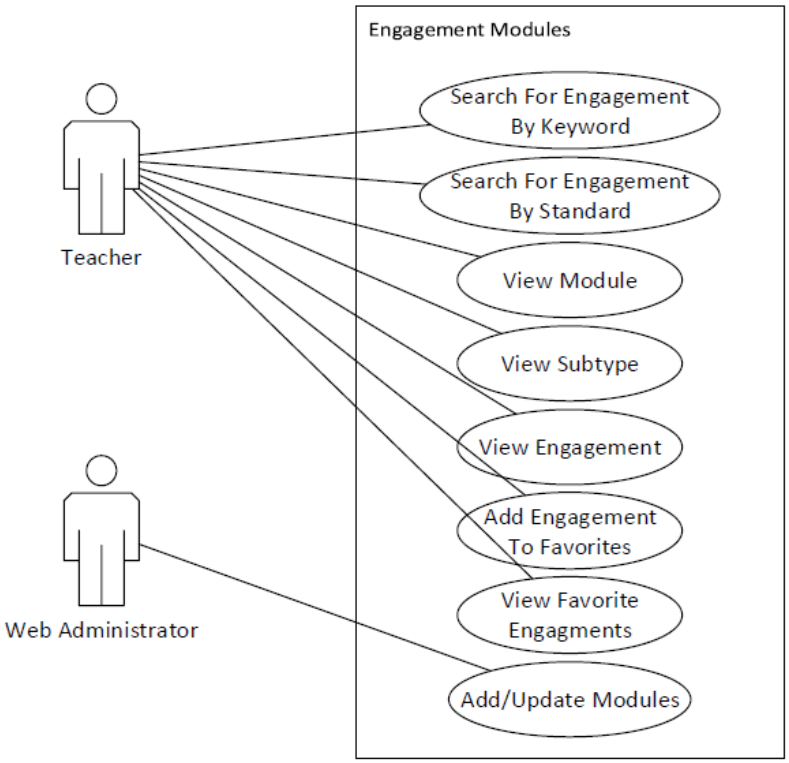
Appendix H: Future Development – Professional Develoment Platform – ERD

The following diagram provides an example for the systems analysis of a future development task. This entity relationship diagram shows how data should be constructed and stored for the professional development content delivery feature of the application. The motivation behind this feature is to provide teachers with strategies to accomplish specific learning objectives. The documents are divided into three distinct layers, allowing teachers to query and save entire documents or individual components of the documents. Each layer is associated with keywords to enable the searching of these components. A search algorithm will need to be implemented that prioritizes results in the following order: Titles then keywords text text.



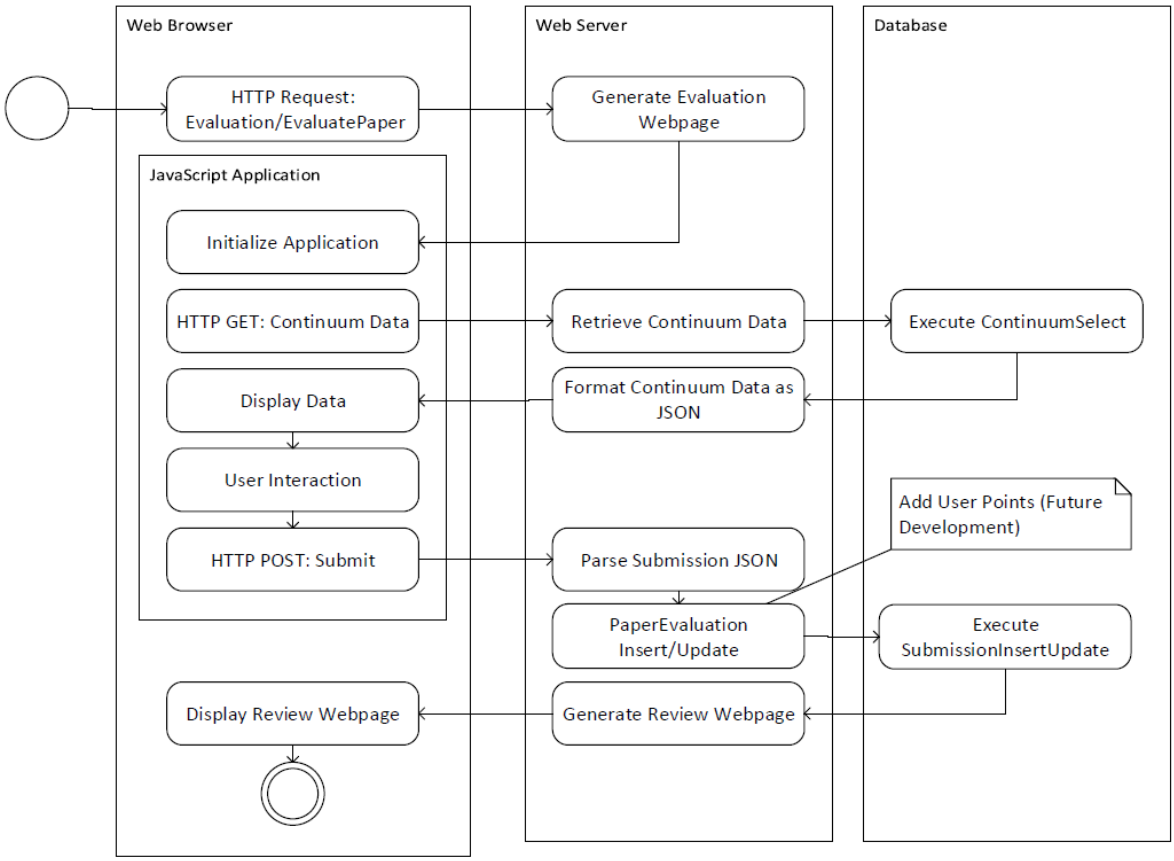
Appendix I: Future Development – Professional Development Platform – Use Case Diagram

The following diagram provides an example for the systems analysis of a future development task. This use case diagrams describes the types of actions that can be performed within the professional development content delivery platform and indicates which types of users have access to each use case.



Appendix J: Evaluation Process Activity Diagram

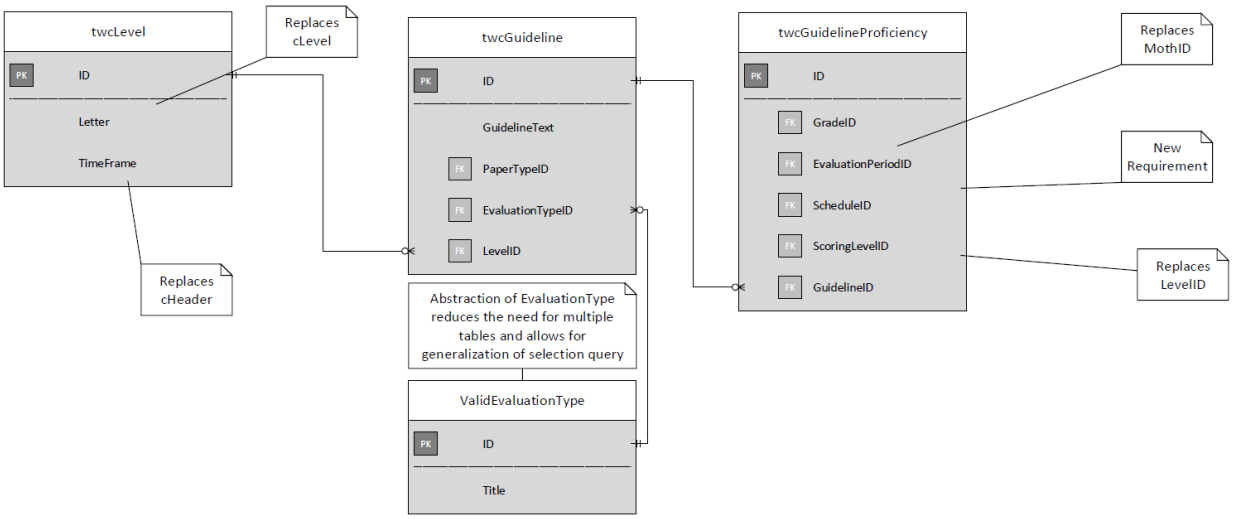
The following activity diagram demonstrates the processes that occur when the new evaluation application is accessed. This activity spans three distinct systems, the client's web browser, the web server, and the database. Within the web browser, the javascript application is created to handle user interaction.



Appendix K: Database Schema Alteration Example– Storage of the Continuum

This sections shows an example of a database schema change that has been made during the transition from the prototype to the production application. The figure below shows the ERD for the old method of storing continuum data, followed by the ERD for the new method of storing continuum data. After the figure are two expanded figures, allowing one to view the details of the diagram at the expense of truncating the old method's ERD, accompanied by a description of the changes.

The above figure shows the old method for storing continuum data in the database. There are obvious flaws in this design that limit the ease of development as well as the maintainability of the site. A proposed maintainence task will be to add a new evaluation type to the continuum or separate an evaluation type into two distinct evaluation types. The only method to accomplish this would be to add additional, indipendant tables that follow the established pattern to the schema and generate new access queries that targe the newly created table. This limits the flexibility of the schema.



The above figure describes the new schema implemented in the production version of the application. The guideline table has been generalized to support new evaluation types, allowing the entirety of continuum data to be selected using a single query.

Appendix L: Azure Database Limitations Investigation

A full list of known Azure feature limitations is available via Microsoft's Azure Documentation. The table below aggregates the limitations of the system and notes the expected current and future impact of each limitation. Issues can be marked as "Non-Issue", "Potential Issue", or "Red Flag". If any limitations had been marked as a red flag, indicating that Azure SQL lacked a critical feature to our product, we would have been unable to use Azure SQL as our database solution. All potential issues indicate that the feature is non-critical and is currently in use, or the feature could be useful in the future. All potential issues are discussed below the table.

	Azure Database Limitations	*Non-Issue*	*Potential Issue*	*Red Flag*
1	Collation of system objects	X		
2	Connection related: Endpoint statements, ORIGINAL_DB_NAME. Windows authentication is not available for logins or contained database users.	X		
3	Cross database queries using three or four part names. (Read-only cross-database queries are supported by using elastic database query.)	X		
4	Cross database ownership chaining, TRUSTWORTHY setting	X		
5	Data Collector	X		
6	Database Diagrams		X	
7	Database Mail	X		
8	DATABASEPROPERTY (use DATABASEPROPERTYEX instead)	X		
9	EXECUTE AS logins	X		
10	Encryption: extensible key management	X		
11	Eventing: events, event notifications, query notifications	X		
12	Features related to database file placement, size, and database files which are automatically managed by Microsoft Azure.	X		

13	Features that relate to high availability which is managed through your Microsoft Azure account: backup, restore, AlwaysOn, database mirroring, log shipping, recovery modes. For more information, see Azure SQL Database Backup and Restore.	X		
14	Features that rely upon the log reader: Replication, Change Data Capture.	X		
15	Features that rely upon the SQL Server Agent or the MSDB database: jobs, alerts, operators, Policy-Based Management, database mail, central management servers.	X		
16	FILESTREAM	X		
17	Functions: fn_get_sql, fn_virtualfilestats, fn_virtualservernodes	X		
18	Global temporary tables	X		
19	Hardware related server settings: memory, worker threads, CPU affinity, trace flags, etc. Use service levels instead.	X		
20	HAS_DBACCESS	X		
21	KILL STATS JOB	X		
22	Linked servers, OPENQUERY, OPENROWSET, OPENDATASOURCE, BULK INSERT, 3 and 4 part names	X		
23	Master/target servers	X		
24	.NET Framework CLR integration with SQL Server	X		
25	Resource governor	X		
26	Semantic search	X		
27	Server credentials	X		
28	Sever-level items: Server roles, IS_SRVROLEMEMBER, sys.login_token. Server level permissions are not available though some are replaced by database-level permissions. Some server-level DMV's are not available though some are replaced by database-level DMV's.	X		
29	Serverless express: localdb, user instances	X		
30	Service broker	X		
31	SET REMOTE_PROC_TRANSACTIONS	X		
32	SHUTDOWN	X		
33	sp_addmessage	X		
34	sp_configure options and RECONFIGURE	X		
35	sp_helpuser	X		
36	sp_migrate_user_to_contained	X		
37	SQL Server audit (use SQL Database auditing instead)	X		
38	SQL Server Profiler		X	

39	SQL Server trace		X	
40	Trace flags	X		
41	Transact-SQL debugging		X	
42	Triggers: Server-scoped or logon triggers	X		
43	USE statement	X		

(Byham, 2016)

Current Potential Issues

5 – Database diagrams: The prototype version of the application relied upon SQL Server's database diagrams feature to document the ERD and as a development feature. Uni-SPIRE will have to adopt a separate method for documentation and replace the development practices by writing scripts as opposed to interacting with the GUI

Future Potential Issues

38 – SQL Server Profiler: This is a powerful tool that allows developers to find inefficient queries. Without this feature we will be unable to measure the performance of our database. A substitute method will need to be developed in the future.

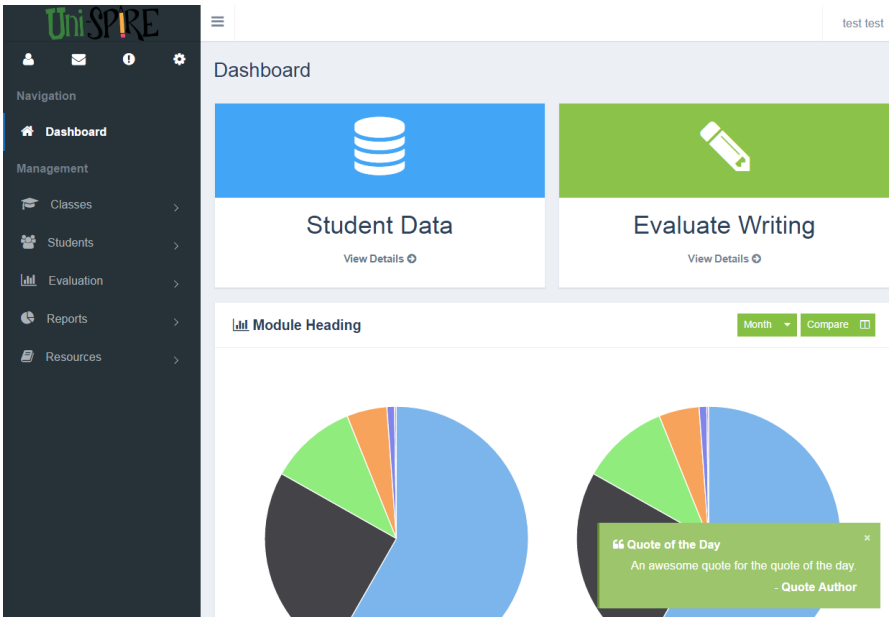
39 – SQL Server Trace: This tool is similar to the SQL Server Profiler, but is less automated. Without this feature we will be unable to measure the performance of our database. A substitute method will need to be developed in the future.

41 – Transact-SQL debugging: The Transact-SQL Debugger allows developers to observe the run-time behavior of a query so they can find errors. Debugging will have to be done without the use of this feature.

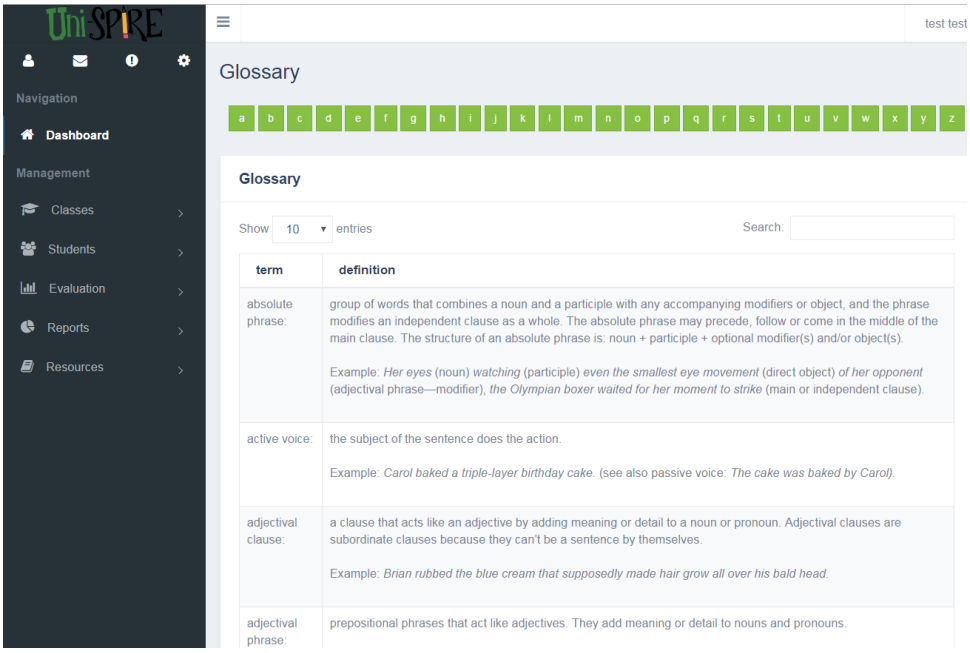
Appendix M: User Interface Comparisons

All user interface comparisons are ordered by prototype followed by production user interfaces. The purpose of this appendix is to provide examples of how the new user interface is more modern and intuitive.

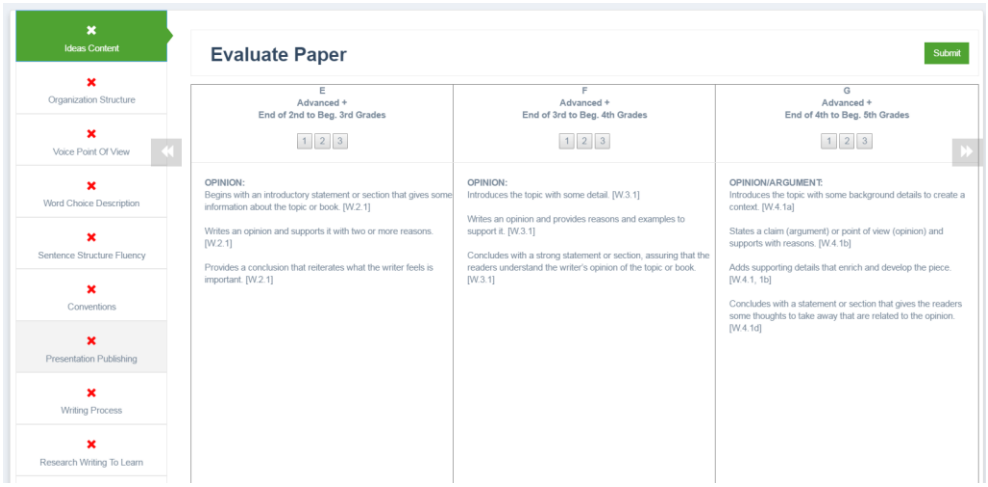
Example 1: Teacher dashboard



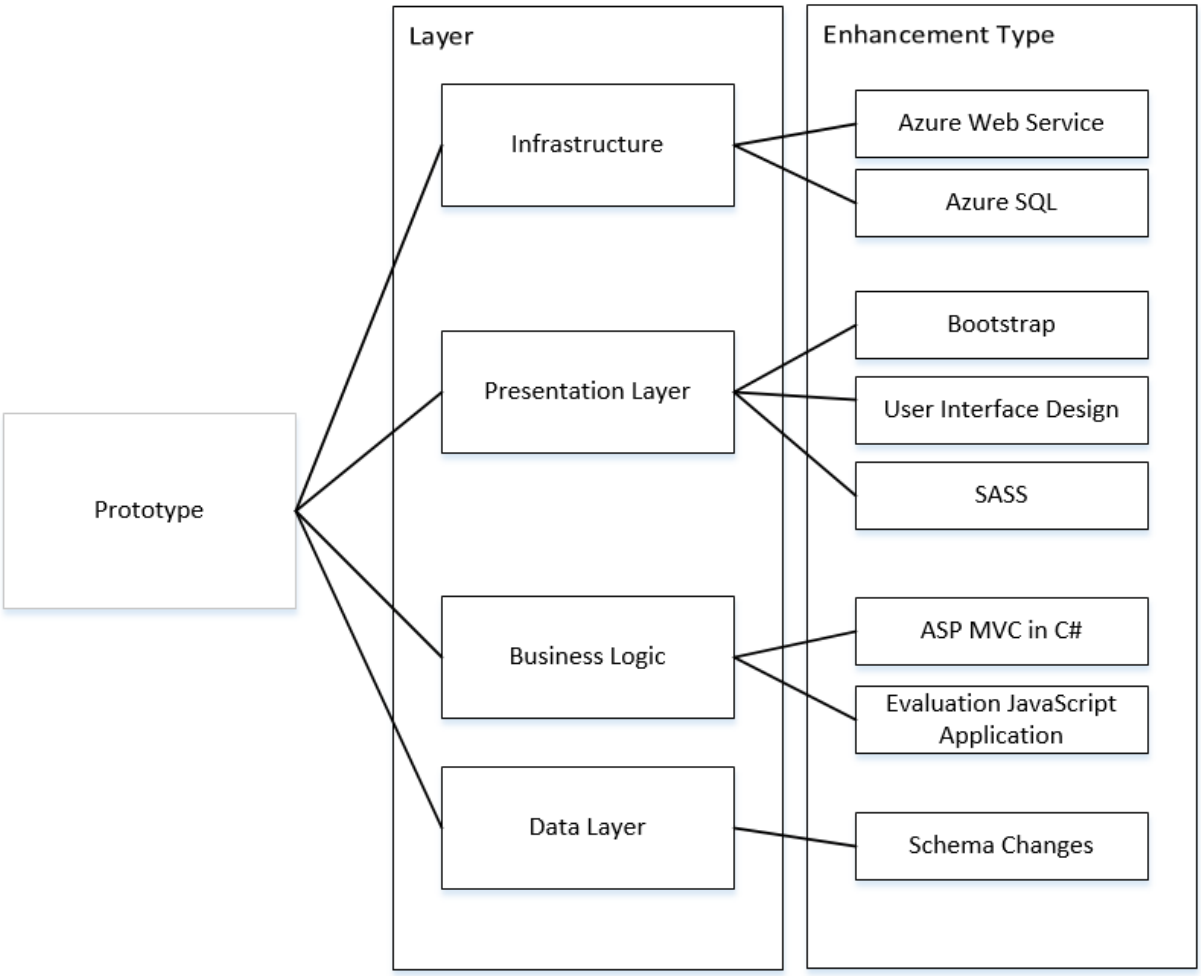
Example 2: An example Teacher Resource, the glossary of terms



Example 3: The evaluation component



Appendix N: Enhancement Types by Layer



Appendix O: Reporting Features Use Case Diagram