

Novel Cloud Bookstore - Report

03-60-422

Agile Software Development

Dr. Xiaobu Yuan

December 8th, 2017

Bryce St Pierre (103998583)

Lavanya Bandla (104138580)

Mohammed Shamsul Arefeen (104707638)

Mohd Tazim Ishraque (104643749)

Robert Charron (103426769)

Sanchay Chawla (104103199)

Executive Summary

This report follows the development of our software system, the “Novel Cloud Bookstore”, by applying the agile software development methodology. We will be discussing each stage of the agile software development process followed during our project, including methods and techniques used, team contribution, the source code, and challenges we faced.

While the Novel Cloud Bookstore may not be the most advanced digital book store application on the market, the collective experiences of the developers who engaged in agile methodologies to create the system were quite formidable. Between engaging in the practices of Scrum, extreme programming (XP), and test-driven development (TDD), the developers grew closer together not only as a team of professionals, but also as individuals in terms of technical skill and overall personal growth.

Under the guidance of a Scrum master who took it upon himself to obtain a certification in scrum during the project, the team weathered a series of challenges. During the project, we discovered that it takes a significant amount of time to become accustomed to a technology stack the team has little to no experience with, which can cause delays. Accounting for the gaps in skill level is also important, as a gap too big would introduce inconsistencies in development strategy and design. We recommend that any team considering the utilization of Agile for their project keep the following things in mind: ensure that the team is composed of members with a high degree of skill and great team dynamics, ensure that the team is sufficiently equipped and highly skilled in the use of the technology stack to be used, ensure that the team is committed to the project by providing them with sufficient incentives to mitigate the any risk of abandonment part-way, and ensure that a proper communication channel is established between the various parties (e.g. the scrum master, the development team, and the scrum product owner) to optimize the requirements elicitation as well as project validation and verification. These steps will improve the odds of achieving project success using agile methodologies, and mitigate the risks which can lead to project failure. Through our trials and tribulations, our group was ultimately able to produce a comprehensive WPF applications using C#, Microsoft SQL Server, and Team Foundation Server (TFS).

Table of Contents

1. Introduction

- 1.1 Purpose of the report
- 1.2 Issues to be discussed and their significance
- 1.3 Project methods
- 1.4 Limitations and assumptions

2. Planning for Agile Development

- 2.1 Project management with Scrum in XP
- 2.2 User stories as product backlog items
- 2.3 Project planning and management
- 2.4 Prepare for sprints

3. Project Overview & Agile Methodologies

- 3.1 Project Overview
- 3.2 Agile Methodologies

4. Black-Box and White-Box Testing

- 4.1 Acceptance criteria in preparation for unit testing
- 4.2 Black-box testing as unit testing
- 4.3 White-box testing with class methods
- 4.4 Discussions about observations made during testing

5. Test-Driven Software Development

- 5.1 System metaphor and design
- 5.2 Test cases from acceptance criteria
- 5.3 Pair programming: from test project(s) to class libraries
- 5.4 Spike solutions and refactoring

6. Teamwork and Customer Involvement

- 6.1 The review of source codes
- 6.2 Customer feedback
- 6.3 Project evolution and system integration
- 6.4 Software deployment

7. Discussions and Recommendations

- 7.1 Recommendation 1

8. Conclusions

9. References

10. Contribution Table

11. Test Case Contribution Table

A. Appendices

Introduction

1.1 Purpose of the report

The purpose of this report is to outline the practices and challenges faced by our group in relation to the development of the Novel Cloud Bookstore, which we produced using Agile Software Development practices. The scope of this report includes all details of group decisions, methodologies, practices, work distribution, as well as individual decisions, challenges and overall workload manageability. While there will be some discussion on the general outline of the produced software code, this report will not go into specifics about any of the code, nor will there be any code presented within this report. The report assumes the reader has sufficient programming knowledge in areas specific to Object-Oriented Design and unit testing, without specific language or technological knowledge constraints, in order to understand the context and specifics presented.

This report is to be submitted to Dr. Yuan on behalf of group 6, Novel Cloud Bookstore, as a final report for 60-422 which will cover all details of our working sprints.

1.2 Issues to be discussed and their significance

The goal of our project was to expand upon an existing application, to enhance the user experience, and offer a wider variety of features to customers. In order to provide a better experience to the user, several aesthetic features had to be implemented, including a massive overhaul to the appearance of the program which focus on UI/UX design.

The scope of our application could be summarized using the high level user story “As a customer, I want the navigation of the Novel Cloud Bookstore to feel intuitive like other applications I’m familiar with and use day-to-day, so that I can purchase my books quickly and easily.” To further expand our user story, we can summarize the features as the following: “I should be able to login or register, easily find the books I which to purchase and add them to a cart. I should then be able to double check my items before checking them out and completing my order. I would also like to have the ability to ask a question if I am having trouble with anything, or if I have any requests to be fulfilled by the bookstore administrator.” Besides the scope from the customer user perspective,

there is also an administrative user perspective; “As an administrator, I would like to have my own dashboard where I can manage aspects of the bookstore such as users, or books, with ease from a central location so that I can quickly and efficiently make changes without going into the database.” Unfortunately, due to circumstances which arose, we were unable to implement the full extent of this use case, and had to compromise. This will be further discussed in the proceeding sections.

1.3 Project methods

Throughout the project we utilized several different types of development methodologies such as scrum, extreme programming, and pair programming. We formed teams in order to pair program based on balancing out the team members’ skill levels in order to develop efficiently and quickly. We simulated changing customer requests each sprint in order to practice extreme programming, all the while keeping our adherence to scrum practices in check.

Our project consisted of two sprints in which sprint 1 was a short, two-week sprint and mainly had user stories implemented for the functionality of the Novel Cloud Bookstore. The second sprint was about a month long that allowed us to add more functionality and design such as ask a question page, logout page, review a list of books, checkout window.

1.4 Limitations and Assumptions

The project currently consists of two sprints, it is assumed that development would continue on with future sprints and future customer requests. Classes created as well as database development were designed with future proofing in mind in order to allow simple continuation of the project.

One of the limitations we faced was communication, not every member of the group used a common social media/messaging platform (some us did not use Facebook so no groups were set-up, texting is a hassle as longer discussion texts with images can become irritating to view on a smartphone). One of our team members presented the idea of using Trello (creative task completion software) to collaborate on our pair programming tasks and for the communication side of things we used a communication productivity software called Slack. This allowed us to create various groups in the application to boost our pair programming communication. We also made use of Skype conference call features in group meetings, in the case some members were unable to make it.

Over the course of six weeks, there were several issues that were experienced within our group which we had to adapt to that had a impact on our workloads. The first major problem we had was a connectivity issue with our database. As our entire program is dependent on a working, functioning database, not having access to create, edit or delete new tables or schemas created a major problem for us. This was our number one priority at the start of the first sprint and requires assistance from outside the group to resolve. In the meantime, we focused on our scrum and planning of our sprint and prioritizing any UI aspects so that once we had database access, we could develop our tables and integrate them into our back end code.

Another issue we faced was health problems within our group. Two of our six members were sick over a course of 2 weeks in the middle of the sprint. In order to stay on track, we needed to adapt quickly and be agile. We sought guidance from the scrum master in deciding on how to restructure our workloads, and prioritize anything that our sick members had to do that were dependencies to be completed by a healthy member instead. We delegated the work to the sick member's pair programming partner as they were most familiar with their work, and several features were also delegated to other members who were working on a related or similar feature. This meant that we could leave work that could be completed last to be passed off to the sick members on their return. We also switched up our pair programming so that no one would be left alone and we could continue as an efficient team. Lastly, we decided to leave out certain features as a compromise, and planned to add them later if we had time (the extended administrator panel with user account control and library editing functionality, we were only able to implement one of these features). These functions were not dependencies and did not affect the completion of our user stories.

2. Planning for Agile Development

2.1 Project management with Scrum in XP

As we were fortunate enough to have a scrum master with an accredited certification in the practice, we began planning our ideas and development for the project productively. The team members as well as the certified scrum master were part of the entire development process. The initial step in our project was to create user stories for the desired functionality of our software. We tried our best to ensure that we stuck to our initial scope to avoid "scope creep", which is often a common consequence of dealing with external stakeholders who have increasing demands regarding the features they want added to the system. We used the different Scrum rules and practises as given below.

Scrum: There are three documents used for the project

- 1) **Product Backlog:** This gathers and prioritizes all the requirements in a list and used to describe the upcoming work on the product. It consists of the project features, technical work, bugs and product acquisition.
- 2) **Sprint Backlog:** This is a small iteration sprint usually about two weeks. Here we implement design, documentation, development and testing. The sprint backlog also includes the use cases for the sprint. We had two sprints for the project. Sprint 1 was for two weeks and Sprint 2 was for five weeks.
- 3) **Sprint Results:** The document describing the use cases completed during our sprint 1 and sprint 2.

Throughout the course of the project we had several different meetings on the progress and development. The scrum master over looked every meeting and made sure to keep any team members informed if they missed a meeting. We used different social media platforms like Trello, Slack and Facebook messenger to keep in touch and be up to date with everything that is happening in the meetings including the class as well. In both of the sprints a lot of pair programming was used due to some problems that have arised.

Scrum meetings: Before each sprint, we had scrum planning meetings to decide on what we will be going to deliver each sprint. In sprint 1 we did pair programming for the first time in which the six team members were divided into pairs to complete on given tasks that was to be delivered in two weeks.

Daily scrum stand-up meetings: At the beginning of each class we had for this course we met to discuss to the progress towards the goals in each sprint. We used the burndown chart to make note on how the team was doing and make adjustments to the sprint backlog according to our progres.

Sprint retrospective review meeting: At the end of each sprint we had sprint review meetings to review and discuss the results achieved in each sprint. Overall we had two sprint review meeting in which sprint 2 was the last meeting where we reviewed the final code and design for the project.

2.2 User stories as product backlog items

Our user stories are designed from the top down with a high level and simple customer-based user story. “As a customer, I want the navigation of the Novel Cloud Bookstore to feel intuitive in regards to its ease of use so that I can purchase my books quick and effortlessly.” From there we were able to break down what that meant into smaller user stories, and create our product from that. For the first sprint, we focused on 3 customer user stories.

The first requirement in order to order a book is for the customer to be logged in as a user. As such, all customers must register. From this we have the user story: “As a new user, I want to be able to register my account, so that I can purchase books and use the Novel Cloud Bookstore.”

The second requirement to purchase a book is to decide on which books to buy. For this task, customers will need to know as much as possible about the book, from which we developed two user stories.

The first would be information provided from the system: “As a user, I want to have descriptions for every book so that I can know more about the product that I am purchasing.” The second user story revolves around information from other users: “As a user, I want to know what other people who have purchased the book have to say about it so that I can make a better decision to buy the book or not.”

For our second sprint, we further expanded on our first user stories as per customer requests, but we also developed 5 new user stories and 2 developer stories. These user stories describe functions that we need to implement into the system. They are as follows:

- “As a customer, I want to be able add books to my order and checkout the order, so that I may receive confirmation that I have purchased those items.”
- “As a user, I want to be able to search the list of books available in the catalog, so that I can filter the listings and better locate what item(s) I am looking for.”
- “As a user, I want to have a shopping cart to keep track of my items so that I do not have to purchase books one at a time.”
- “As a user, I would like to view reviews of a book before purchasing it. And also be able to leave a review of a book I purchased.”
- “As a user, I want to be able to ask a question so that if I need help with something I have a place to seek help.”

- “As a user, I want to be able to easily navigate the Novel Cloud Bookstore so that I can purchase my books easily and efficiently.”

We also developed some developer user stories to simplify data access within our team development process, as abstraction features, not exactly for the purposes of meeting user requirements. We used these stories to plan our development of classes, methods, and database tables.

- “As a developer, I want to be able to query databases and receive only the information relevant to the primary key.”
- “As a developer, I want to be able to easily access data, so that there is no longer a need to create tightly coupled methods for inserting and retrieving data.”

2.3 Project Planning and Management

As previously mentioned, our project was deeply structure around agile methodologies. As such, a large portion of our team management process involved daily Scrum meetings and planning features in iterations. Planning iterations primarily consisted of scanning product backlog items and deciding on which were most prevalent in the implementation of useful Novel Cloud Bookstore features. Throughout our project we had scrum meetings before each sprint, daily scrum meetings and sprint review meetings at the end of each sprint.

Our qualified and certified Scrum master, as intended, would oversee the direction of these daily meetings and gauge the progress associated with the features within the sprint. He took a equitable approach by listening to thoughts and concerns of each of the developer’s within the team, which made it possible to weigh the priorities for the work to be done that day. By also consulting the burndown chart, this multi-faceted and transparent conversation between the team was the core engine by which the project tasks were planned and managed.

We moved our team members around to avoid knowledge loss or coding bottle necks. Cross training was implemented to make sure the team members move around the code base in combination with pair programming. So that not only one person knows about one given section of code but everyone in the team knows much of the code in each section. Pair programming made it possible not to lose productivity and so that the team can be flexible since the team members know enough about every part of the system to work on it.

2.4 Prepare for sprints

Once we had decided on our user stories, it was easy to prepare for our sprints. We allocated each of our user stories and additional tasks a required amount of time and assigned them to the team member according to their strengths. From these user stories, we broke down each into different related tasks that would combine to complete the user story and allocated those proportional time for each. We decided where and for which tasks or user stories would be best for pair programming or which would be best done individually.

3. Project Overview & Agile Methodologies

3.1 Project Overview

During the project, we used agile over waterfall because our requirements were unclear from the beginning and the project was small scale. We used xtreme programming because we had 6 group members with various skill levels, we split into 3 pairs for pair programming. We used test-driven development (TDD) because our development cycles were short. The specific details regarding each of these processes will be further discussed in the sections below, with justification for why we decided to take the approaches we did. The following is a comprehensive use case diagram for our system, which we kept in mind during the TDD process, as a guide.

3.2 Agile Methodologies

Throughout the development of the Novel Cloud Bookstore, several Agile Methodologies are applied. There is no wasted code within - all code, methods and classes serve a purpose. Everything that has been coded has been implemented to the Novel Cloud Bookstore. While we progressed and added more features, we amplified our learning by finding new ways to solve problems, or better ways to make our code work together coherently. Our database classes were an example of amplified learning; as we were developing the database, we decided it would be best to have a single class to build our SQL insert or delete queries as opposed to rewriting them over and over or copying and pasting the queries. We also made decisions on what SQL functions we needed within our classes as we went on and made these decisions based on future use planning. We would add things as they were required outside our guidelines when

they were needed which allowed us to deliver the product as quickly as possible and move on to developing further functionality.

Within our meetings we had discussions over how we wanted the classes to interact with each other, but it was the decision of the programmer to decide how the code would work, given that it completed its task. We would update our scrum master on how our classes were designed and applied so that he could make decisions on how to continue. Our team was empowered to create and program the way they choose while solving the problems we set out together as a team. This often worked well with pair programming which further applied amplified learning, as we could often see different ways of solving the same problem. During our standup meetings, we often simulated the customer's perspective and presented the system components to each other with all the working parts together. In doing so, we were able to ensure the whole team understands the full system, including parts they have not worked on, and built in integrity with our customer by being forward about the product and where it is, as well as receiving feedback from the customer. These practices allowed us to be flexible and efficient in our work.

4. Black-Box and White-Box Testing

4.1 Acceptance criteria in preparation for unit testing

For each of our user cases before we created our test cases to begin test driven development, we took each of our user cases and broke down all acceptance criteria to ensure integrity of each test. Creating our high level test case acceptance criteria on each individual user story helped us decide upon a path to take with our design further down the road. These are a list of our user-based user cases and their acceptance criteria:

- *“As a customer, I want to be able add books to my order and checkout the order, so that I may receive confirmation that I have purchased those items.”*
 - There must be a selection of books available for the user to select
 - There must be an order record kept for each customer to add the book
 - Customers simply click an add button which is next to each book in order to add the item to their cart

- Customers can view and modify their order at any time before they check out
- Customers must start out with an empty cart
- Cart should be emptied after checking out an order
- *“As a user, I want to be able to search the list of books available in the catalog, so that I can filter the listings and better locate what item(s) I am looking for.”*
 - There must be an existing catalog of books
 - There must be a database of books that can be queried
 - Users must be able to input their searches into the program
 - Search can be performed by entering a part of the book title in search bar
 - Search can also be performed by book category
 - Rows returned may not have any results (if search criteria doesn't match with existing books in the database)
- *“As a user, I want to have a shopping cart to keep track of my items so that I do not have to purchase books one at a time.”*
 - Users must have a cart independent for each user
 - Shopping cart must track all requested purchases
 - Users should have the ability to select different books
 - Users should be able to view their cart and remove or add items if they choose
- *“As a user, I would like to view reviews of a book before purchasing it. And also be able to leave a review of a book I purchased.”*
 - Each book should have its own reviews page, where all the reviews for the selected will be listed
 - Users must be able to access individual book review pages, without a need to log in
 - However, users must have purchased the book in order to leave a review
 - Items must be checked out completely (thus completing the order) before allowing a review to be submitted
 - Users should be notified when review has been successfully submitted
- *“As a user, I want to be able to ask a question so that if I need help with something I have a place to seek help.”*

- Customers must be able to ask questions on a new page
 - Link to new page added on MainWindow
 - Admins cannot ask questions
 - Admins must be able to view questions on a new window
 - Questions must be added to database and the user should be notified
- *“As a user, I want to be able to easily navigate the Novel Cloud Bookstore so that I can purchase my books easily and efficiently.”*
 - Navigation between should be easy
 - Users shouldn't have to search for content
 - UX should feel intuitive - users shouldn't need instructions
 - Buttons should be descriptive

4.2 Black-box testing as unit testing

For Test-Driven Development, we chose to do all unit testing as black-box unit tests in accordance with agile software development best practices. A good example of our black-box testing would be our UserRegistration test classes. We developed our user story:

“As a new user, I want to be able to register my account, so that I can purchase books and use the Novel Cloud Bookstore.” From there we created our acceptance criteria which lead to our test case: Please refer to figure 4 in Appendices Section A.

Within our black-box test, we create the variables we want to pass into our database. We directly insert them into the database and then retrieve the database results. We then check that the results retrieved from the database are the same as the results that we put in, and that the insert into the database was correct. As this is a black-box unit test, we do not care about the inner working restrictions on things such as username or password, or how it comes about those results. It only verifies that the inputs we created will work with the existing classes and databases.

From this point we were able to develop our method to register users. We knew the information we would need from the test case - the username, password and verification, and the full name of the user and we know it will work passing it off to the

Database classes once the test case returned true. Refer to Figure 5 in Appendices Section A..

Most of our test classes contain many different test cases which will all help for the development of a single or few methods. These test cases help us determine within our methods any limitations that need to be implemented. One example of this would be the “TestAddBookToLibrary” where we have a number of tests for different aspects of the add book feature in the admin user panel. The method will have 10 different inputs related to the book to be implemented so we have 10 different unit tests for each input possibility. Refer to Figure 6 in Appendices Section A.

In total we have 36 unit tests that we added or created for these two sprints for our development.

4.3 White-box testing with class methods

White-box testing is for internals-based tests and require knowledge to derive test cases from the code.

In our project for Test-Driven Development we did not use any white box testing methods as we were working with Black-Box Unit Tests in order to create our methods. None of the methods or functions within our final program required specific inputs where white-box testing would be the unit test of choice.

4.4 Discussions about observations made during testing

Test-Driven Development was very useful for working as a team. Once we had out specific test cases, any and all new methods implemented must pass as the unit tests do. These unit tests made it very easy to create these methods as we have working examples to create from which we know function properly. It was also very simple to decide what the goal of our test cases would be based on our acceptance criteria that we did prior to the test cases.

5. Test-Driven Software Development

5.1 System metaphor and design

As mentioned from the class the system metaphor “is a story that everyone can tell about how the system works, including customers, programmers, and managers. “ The most important detail is to explain the system design to new people without the need to dump huge documents on them. The metaphor used for our system was “a bookstore library” akin to a physical bookstore people would visit to purchase their books, which helped us decide on system class and feature names.

The software design and architecture was developed using user requirements defined in the acceptance criteria and test cases to features section. We implemented techniques such as modularization and concurrency to promote a modernized design framework. Before the final product was compiled, design verification took place to ensure the proper scaling of layouts, quality of windows produced and accuracy of the UX /UI design. We attempted to step into the shoes of the customer or the scrum product owner, and reviewed our design based on benchmarks obtained while searching for similar applications to ours and our expectations towards the aesthetic outcome of our product.

5.2 Test cases from acceptance criteria

Testing was used with test cases to execute the different components of the software under different specified conditions. We made sure to keep in mind while developing the test cases that each test case is effective in detecting errors or defects, is exemplary in testing to reduce the number of test cases being used overall, is economical to perform, analyze and debug the code and finally the test case is evolvable towards software changes in the code.

Test case: We used a set of inputs with certain conditions to receive the results for each test case. Each file in our project had numerous test cases to test the functionality and the desired outcome.

Test Suite: There is a test suite for each file implemented. Where each test suite contains a collection of test cases related to each file.

Test procedure: We followed proper test procedure with detailed instructions on what to expect for each test case.

5.3 Pair programming: from test project(s) to class libraries

Throughout the course of this project pair programming was implemented different tasks that needed to be completed. This allowed us to increase the software quality without any impact on the time to be delivered. The tasks were given to different pairs of team members based on their skills and their availability to work together. In pair programming we designed test cases and grouped them in files for each required program of code. If needed we implemented and tested more test cases for certain tasks such as database related files like when inserting data or deleting data into the database.

5.4 From test cases to features

Most of our non user interface related features were derived from test driven development practices which helped us secure an error-free development environment. Using best practice black-box testing methods, our team was able to narrow down our test cases to those that are most relevant to the features we were testing. Mentioned below are the test cases that were used:

Testcase 1 : TestAddBookToCart

The test methods in this test case involved us testing whether the books were successfully added to the checkout cart. Equivalence partitioning was used to test for edge cases to check whether the class item orderItem was successfully added to object bookOrder as a list item. Variables contained within the orderItem class are ISBN, Title, Price, Quantity, and description all of which undergo equivalence partition testing. Upon passing the test case methods, our intended functionality was achieved and therefore we were able to proceed onto the implementation of the adding a book to cart feature.

Testcase 2: TestAddBookToLibrary.

The test methods in this test case involved us testing whether we were able to add a book to the library. Equivalence partitioning was used to test for edge cases to check whether the value for book items were properly added and displayed in the bookstore database.. Upon passing the test case methods, our intended functionality was achieved and therefore we were able to proceed onto the implementation of the adding a book to the library feature.

Testcase 3: TestAddReview

This case comprised of four individual test methods and involved us testing whether reviews could be added to the database, given different parameters. Equivalence partitioning was used to test for edge cases to check whether the entered value matched the partition. Variables that were tested in particular were ISBN and orderID. Upon passing the test case methods, our intended functionality was achieved and therefore we were able to proceed onto the implementation of the add a book review feature.

Testcase 4: TestCheckoutTotal

The test methods in this test case involved us testing the functionality of the checkout feature. Equivalence partitioning was used to test for edge cases to check whether the value of the total order amount was zero. Upon passing the test case methods, our intended functionality was achieved and therefore we were able to proceed onto the implementation of the Checkout Total feature.

Test case 5: TestDatabaseDelete

The test methods in this test case involved us testing whether we were able to delete items and data from the database. Equivalence partitioning was used to test for edge cases to check whether the value has been entered . Upon passing the test case methods, our intended functionality was achieved and therefore we were able to proceed onto the implementation of the feature.

Test case 6: TestDatabaseInsert

The test methods in this test case involved us testing whether we were able to insert items and data from the database. Equivalence partitioning was used to test for edge cases to check whether the value has been deleted. Upon passing the test case methods, our intended functionality was achieved and therefore we were able to proceed onto the implementation of the feature.

Test case 7: TestDatabaseSelect

The test methods in this test case involved us testing whether we were able to select items and data from the database. Equivalence partitioning was used to test for edge cases to check whether the value was selected. Upon passing the test case methods, our intended functionality was achieved and therefore we were able to proceed onto the implementation of the feature.

Test case 8: TestDescription

The test methods in this test case involved us testing whether we were able to delete item and data from the database. Equivalence partitioning was used to test for edge cases to check whether the value blabla. Upon passing the test case methods, our intended functionality was achieved and therefore we were able to proceed onto the implementation of the view description feature in book details.

Test case 9: TestQuestions

The test methods in this test case involved us testing whether the user is able to ask any questions and if they are being entered in the database. Equivalence partitioning was used to test for edge cases to check whether the value for the string invalidQuestion since it is null and to check if it is entered in the database. The string cannot be a null or empty as the user is required to enter input on the page. Upon passing the test case methods, our intended functionality was achieved and therefore we were able to proceed onto the implementation of the ask questions feature.

Test case 10: TestQuestions1

The test methods in this test case are similar to Test case 9, with additional test coverage criteria to check for correct input of the Question string variable.

Test case 11: TestRegisterUser

The test methods in this test case involved us testing whether we were able to allow the user to register an account. Equivalence partitioning was used to test for edge cases to check whether the values for the sample input in the test case allowed a new account to be registered. Upon passing the test case methods, our intended functionality was achieved and therefore we were able to proceed onto the implementation of the register feature.

Test case 12: TestRegisterUser1

The test methods in this test case is similar to Test case 12, with additional Test Methods that checked to see if the user is able to create an account with two different passwords and another test method to see if the user can create an account without meeting the password requirements. Upon the passing of these test case methods, the functionality was achieved and were able to proceed with the register user feature implementation.

Test case 13: TestRemoveBookItem

The test methods in this test case involved us testing whether we were able to delete a specific book from the BookOrder class. This is in essence a complimentary test case to adding a book to BookOrder class(as can be seen in test case 1: TestAddBookToCart). Equivalence partitioning was used to test for edge cases to check whether the values stored in OrderItem class matched that which was stored in a BookOrder item. Upon passing the test case methods, our intended functionality was achieved and therefore we were able to proceed onto the implementation of the remove book feature from the shopping cart.

5.5 Spike solutions and refactoring

In cases of tough technical or design problems we used spike solutions in our project. When we were designing the UI for our project we had problems with the image not being displayed properly on our computers as each team member had a different type of computer which made the design and layout look not proper. So we tested and made simple different program and design on the side to keep as alternate solutions that give us the least number of technical difficulty and increase the implementation of the task. However, at the end with the help of another team member we were able to

figure out that the solution for our problem and that was due since each team member having a different laptop the pixel range had to be adjusted so the UI and design can be properly viewed. This immediately fixed our problem.

Refactoring was used when we had a lot of unused functionality and side program. We had programs that were a good guide post in the beginning of the project but towards the end of our final sprint some of them were just making a needless cluster and complexity in the project. For example we decided to implement a FAQ page where there are list of questions that a user can browse through if they need any help to use the bookstore. Later we decided to change the idea to Ask a Question page where instead of already having a bunch of question visible the user can ask and submit questions to the administrator regarding any concern or if any help is required. The users can submit there question along with their name and contact information so that administrator can reply their concern.

Hence, refactoring helped to eliminate unused and not resourceful programs, functionality and designs. By removing these materials from our project we were able to keep our code simple and avoid any unnecessary cluster of code.

6. Teamwork and Customer Involvement

6.1. The review of source codes

In the spirit of transparency and keeping a consistent perspective on the project, code files and associated test cases were always reviewed by at least one other person on the team.

While engaging in pair programming, the review of source code would usually be reviewed by each developer's partner in programming. However, in other cases, code files that were especially critical to the behaviour of the application were reviewed by the team as a whole.

This supported the notion of each team member maintaining an overall knowledge of the source code as a whole, but it also allowed for several team members to use their individual talents and weigh in on portions of the system that they may not have been directly developing at the time.

6.2. Customer feedback

Since explicit customers were not always available to be interleaved with the implementation phases of the application development process, we often assumed the roles of the customers ourselves during team meetings and discussions. This is to say that as consumers of books ourselves, and being familiar with other systems for purchasing books online or otherwise, we were able to use our own perspectives to simulate realistic customer feedback. We consulted Dr. Yuan when possible, and he steered us in the right direction, and addressed any major concern we had.

A significant aspect of our application's design is its aesthetic value. When presenting our user interface, its purposeful use of colors, graphics, and stylistic user inputs and controls were appealing to most people interacting with the application. As a consumer application for browsing and ordering books, we found that an attractive and simple user experience was of greater importance to customers than an extensive or complex set of features. The comments and criticisms the team received reinforced this thought process.

6.3. Project evolution and system integration

The project evolved through careful administration of the Scrum master and the individual participation of each team member. Pair programming was used throughout the project and this helped in making proper evaluations on the tasks.

During our scrum meetings and after finishing each task we would push our code and ask the other team members to evaluate and comment on it to see if there are any changes or implementations that can be made. And at times it was necessary for team members to collaboratively work on a module (sometimes outside our designation), especially where software features and requirements overlapped.

6.4. Software deployment

As the application was created using the collaborative and version control features provided by Team Foundation Server (TFS), the software was completed within each sprint by every developer committing and merging their changes to source code repository.

Once this was completed, the application build could be successfully executed on any local machine that pulled the latest version of Novel Cloud Bookstore. The built application took the form of a binary executable, which is transferable to any other machine running Windows and is either accessing the University of Windsor campus network directly, or accessing the campus network via tunneling with a VPN client.

Since we have opted to spread the software using an executable, there is no need to uninstall the application, a simple delete will remove it from the system. Version updates would be as simple as sending a new copy of the executable to potential customers. This process could have been streamlined if we had more resources to dedicate and incorporate other types of software deployment methods and techniques. Regardless, our customers will always be presented with the most frequent books data in the catalog window as it is connected via our back-end database, which reflects updates immediately regardless of the version the customer may be running.

7. Discussions and Recommendations

Novel Cloud Bookstore software project functions well and useful to a customer allowing them to view different books, buy books, write a review for the book, ask questions if they need help, create accounts, etc.,

However, there are definitely some changes and improvements that can be made in the future for this project. If there is a sprint 3 we can focus more on implementing a larger set of features into the Novel Cloud application. This is a brief list of features that should be considered if there were another sprint:

1. A rating system that aggregates all the user reviews for a book and displays it on the main window would be an additional good feature that can be implemented.
2. Another feature that can be implemented would be to allow the users to sort and view the books by the ratings given to them like for example like allowing a user to view the books from highest reviews to lowest reviews or vice versa.
3. Features could be made customer-oriented, providing coupons and discount functions could attract a larger user-base. This could also be used for promotional purposes.
4. Members discussion area, where users can converse about the books among other things. This would create more user interaction with the application, which in term may increase the amount of books purchased over time.

There are many features that can be considered when designing for various user levels (admin/member/guest). Some interesting discussions held by team members lead to the creation of features currently implemented in the book store application; such as reviewing of books, adding questions panel, creating user-levels among others.

With regards to the agile development practices, we faced many challenges which a regular professional agile team would not have faced during real world industry practices. As we are still students, we faced many hurdles. For the future, I would recommend that more time be spent during the first sprint to learn the technology stack so that issues and delays could be avoided down the line. Teams should be picked based on member skill level to optimize team coherence. Overall, there is still much to be learned, as agile is a tough and challenging approach to master.

8. Conclusions

Overall, our objective of developing a stable and aesthetic bookstore library system was achieved. Throughout the software development and project management process, our team grew with regards to both technical and soft skills. We learned a variety of new things which will enable us to succeed upon entering the professional software development industry, within which agile is quite popular. We were very fortunate to be able to experience the nuances which real life professional scrum teams go through, what challenges they face and how they tackle such challenges. It was a wonderful experience, it was a great pleasure being in this course. As a team. We'd like to thank Dr. Yuan for orchestrating our learning process. To wrap things up, agile is a different animal but the same beast; with practice our skills can be improved, and the agile approach can be mastered.

9. References

- [1] J. Eaglesham. *Scrum*. [Online]. Available: <http://epf.eclipse.org/wikis/scrum/index.htm>
- [2] D. Wells. (2014, Oct.). *Extreme Programming: A gentle introduction*. [Online]. Available: <http://www.extremeprogramming.org/>
- [3] (2017). *The Home of Scrum*. [Online]. Available: <https://www.scrum.org/>

10. Contribution Table

This table outlines the contributions that our team members made to the overall project. In the table the “responsibilities” column mainly concerns with the implementation of the different user stories and design components to the project. It also included the database task responsibilities that were completed. Each task that is outlined in each user story is described in detail with the work that was completed with respect to the “Responsibilities” task.

Responsibilities	Member	Task 1	Task 2	Task 3	Task 4
Improve on the Book Review System	Mohammed/Sanchay	Utilize the order number to track customers	Allow users to check submitted reviews	Give permissions to users to update customer reviews	Give permissions to admins to manage the reviews posted by users
Landing Page	Mohd Tazim	Back End Test Driven Development	Landing Page User Interface	User Access Abstraction	Landing page integration with book catalog
Alter UI Components to match user permission level	Bryce/Mohd Tazim Ishraque	Develop User Permission Framework	Develop User Views for Main Window	Integrate Administrator access to mainwindow	Develop GUI AdminWindow
Generic Database Access Procedure	Sanchay	Create test cases for separate database connection.	Create table specific database generic access using SQL Queries and implement test cases	Incorporate new SQL preparation methods into existing code	Provide the changes for all windows so GUI displays (using the datagridview) are accessed using the preset database generic queries

Develop Checkout Procedure	Bryce	Develop test cases	Develop Checkout Window GUI	Develop and update checkout DAL class	Integrate checkout window into the main application
Normalization of current database	Robert	Normalize all database tables	Integrate new tables into generic database procedure class	Ensure all existing code now references updated tables	Resolve any conflicts and remove unnecessary tables
Create register profile for user	Lavanya	Create a page for users to create their profile	Create a query to add user data to the database	Implement a way to navigate to other windows	Implement security to user profile
Search Function	Sanchay	Create a UI search bar	Create new method to work with search bar UI	Create search method test questions	Integrate search into existing menu
Develop Checkout System	Robert	Create test cases for checkout system	Update database checkout queries	Integrate new queries into the existing system	Create validation system for checkout
UI/UX Overhaul styling	Mohammed/ Mohd Tazim	Create a new design login page (Aesthetics)	Make changes to book order window	Add additional aesthetic changes to the main window	Recreate book catalog page and add additional visual appeal
Add shopping cart feature	Mohd Tazim	Create shopping cart UI	Integrate shopping cart into bookstore catalog page	Create shopping cart functions and UI with test cases	Add visually appealing UI features
Create a add questions section	Lavanya/ Mohammed	Create a new "as a question" page	Create new queries and add to generic database	Create permissions levels and UI changes for admins or	Create a page for admins to view all asked questions

			classes	guests	
Group Test Cases	Bryce	Develop TDD cases for registering new users	Create Test Cases for database access	Create test cases for reviewing books	Group all test cases together within the solution and ensure all tests pass
Bug fixes	All members	Ensure integrity of all functions	Input unexpected results to test inputs	Check functions work properly as all user types	Check UI alignment under different resolutions







11. Test Case Contribution Table

This table outlines the contributions that our team members made to the overall test cases used in the project.

Test Case	Member(s)	Functionality
TestAddBookToCart	Robert	Test if user can add book to checkout cart
TestAddBookToLibrary	Mohd Tazim Ishraque	Test if admin can add book to database
TestAddReview	Limon	Test if user can add a review of a book
TestCheckoutTotal	Bryce	Test if the total checkout amount is calculated as expected
TestDatabaseDelete	Sanchay	Test if the admin can delete item from the bookstore database
TestDatabaseInsert	Sanchay	Test if the admin can insert items from the bookstore database
TestDatabaseSelect	Sanchay	Test if the admin can select items from the bookstore database
TestDescription	Mohd Tazim Ishraque	Test if the admin can add descriptions to the books
TestQuestions	Limon	Test if the users can ask questions and have it entered into the database
TestQuestions1	Lavanya	Test more test cases to see if the users can ask questions and have them entered into the database
TestRegisterUser	Bryce	Test if the user can register a new account
TestRegisterUser1	Lavanya	Test more test cases to see if the users can register a new account
TestRemoveBookItem	Robert	Test if the admin can remove book items from the checkout cart

Project teammates signatures

We collaborated and agreed on both the contribution tables and hence all the team members have signed to confirm this in the below table.

Name	Signature
Bryce St Pierre	
Robert Charron	
Lavanya Bandla	
Sanchay Chawla	
Mohd Tazim Ishraque	
Mohammed Shamsul Arefeen	

A. Appendices

They contain detailed information, such as questionnaires, large tables, graphs, and diagrams. The figures are clearly set out and numbered in the order they are mentioned in the text.

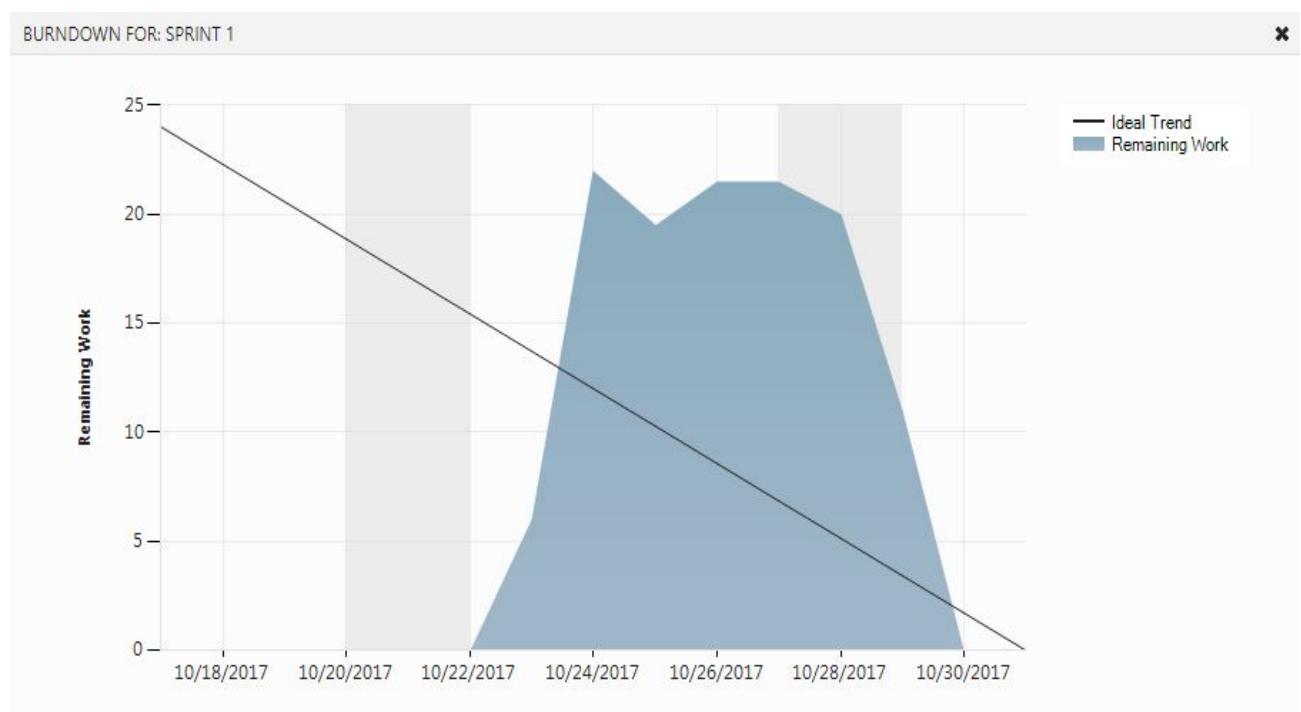


Figure 1: Burndown Chart - Sprint 1

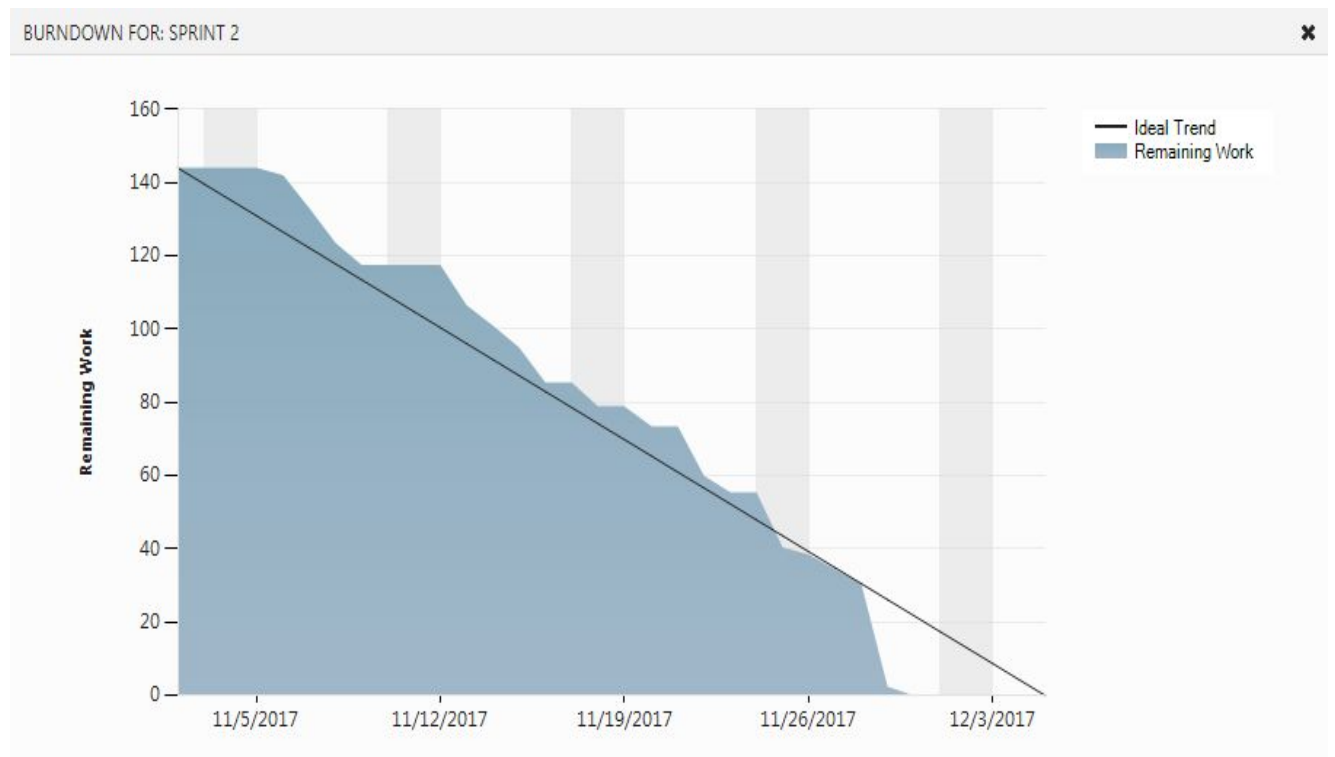


Figure 2: Burndown Chart - Sprint 2

Product Backlog

contents forecast off add items on

Create Backlog Query | Column options

Type: Product Backlog Item

Title: Add

Order	Title	State	Effort	Iteration Path
1	Add Book descriptions to GUI	New	6	BooksAgile6
2	Allow Users To Add Textbook Reviews	New	6	BooksAgile6\Release 1\Sprint 1
3	Order IDs always = 0	New	2	BooksAgile6\Release 1\Sprint 1
4	Users can check out without logging in.	New	1	BooksAgile6\Release 1\Sprint 1
5	Create Brief Descriptions for Books (1 - 2 lines/book)	New	8	BooksAgile6\Release 1\Sprint 1
6	netflix & code	New	0	BooksAgile6

Figure 3: Product Backlog

```

[TestMethod]
public void TestMethodRegisterUser ()
{
    inputName = "TestUser";
    inputPassword = "ts0987";
    inputPasswordV = "ts0987";
    inputFullName = "TestUserFull";

    bool expectedRegister = true;
    bool expectedLogin = true;
    bool expectedDelete = true;

    var c = new SqlConnection(Properties.Settings.Default.dbConnectionString);

    bool actualRegister = userRegister.Register(inputName, inputPassword, inputPasswordV, inputFullName);
    bool actualLogin = userData.LogIn(inputName, inputPassword);
    int actualUserId = userData.UserID;

    ds = dbQ.SELECT_FROM_TABLE("SELECT * FROM UserLoginData WHERE UserName = 'TestUser'");
    int expectedUserId = Int32.Parse(ds.Tables[0].Rows[0]["UserID"].ToString());

    Assert.AreEqual(expectedRegister, actualRegister);
    Assert.AreEqual(expectedLogin, actualLogin);
    Assert.AreEqual(expectedUserId, actualUserId);

    DBDelete dbD = new DBDelete();
    bool actualDelete = dbD.deleteRow("DELETE FROM UserLoginData WHERE UserName = '" + inputName + "'", c);
    Assert.AreEqual(expectedDelete, actualDelete);
    c.Close();
}

```

Figure 4: Test case

```

public bool Register(string username, string password1, string password2, string fullName)
{
    if (!UserRegister.VerifyRegistration(username, password1, password2, fullName))
    {
        ErrorMessage = "Please fill in all fields.";
        return false;
    }
    if (DALUserInfo.HasInvalidPassword(password1))
    {
        ErrorMessage = "Password requires at least six alphanumeric characters.";
        return false;
    }

    string newType = "CU";

    DBQueries db = new DBQueries();

    List<object> bookstore = new List<object>(
        new object[] { username, password1, newType, 0, fullName }
    );

    return db.INSERT_INTO_TABLE("UserData", bookstore);
}

```

Figure 5 : Black-box testing


```
public class TestAddBookToLibrary
{
    [TestMethod]
    public void TestMethodISBN()...
    [TestMethod]
    public void TestMethodCategory()...
    [TestMethod]
    public void TestMethodTitle()...
    [TestMethod]
    public void TestMethodAuthor()...
    [TestMethod]
    public void TestMethodPrice()...
    [TestMethod]
    public void TestMethodSupplier()...
    [TestMethod]
    public void TestMethodYear()...
    [TestMethod]
    public void TestMethodEdition()...
    [TestMethod]
    public void TestMethodPublisher()...
    [TestMethod]
    public void TestMethodAdd()...
```

Figure 6: Black-box testing