Final Comprehensive Assignment

Winter semester 2017

Systems II (420-E21)

Philip Dumaresq

2017

Contents

[Part A: System Design 2](#_Toc480364244)

[1) Use case diagram 2](#_Toc480364245)

[2) Activity diagram 1 3](#_Toc480364246)

[3) Activity diagram 2 4](#_Toc480364247)

[4) Class diagram 5](#_Toc480364248)

[5) Sequence Diagram 6](#_Toc480364249)

[6) Sprint backlog 7](#_Toc480364250)

[Part B: System Architecture 8](#_Toc480364251)

[1) Choosing the architecture 8](#_Toc480364252)

[2) Layering the application 8](#_Toc480364253)

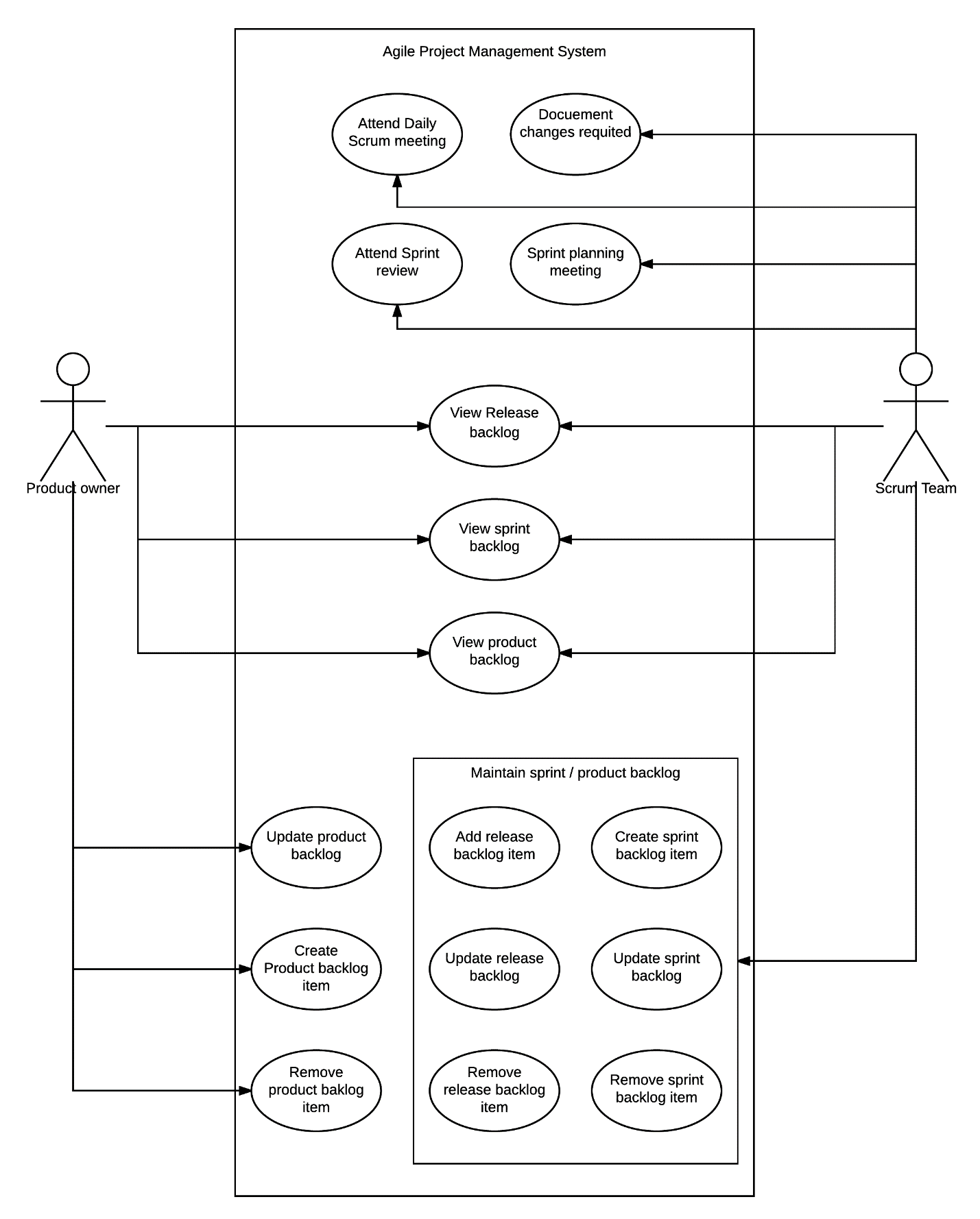
[3) Deployment Diagram 8](#_Toc480364254)

[Part C: System Development Methodologies 9](#_Toc480364255)

[1) Choosing the methodology 9](#_Toc480364256)

# Part A: System Design

## Use case diagram

Figure 1: Agile Project Management System class diagram

## Activity diagram 1

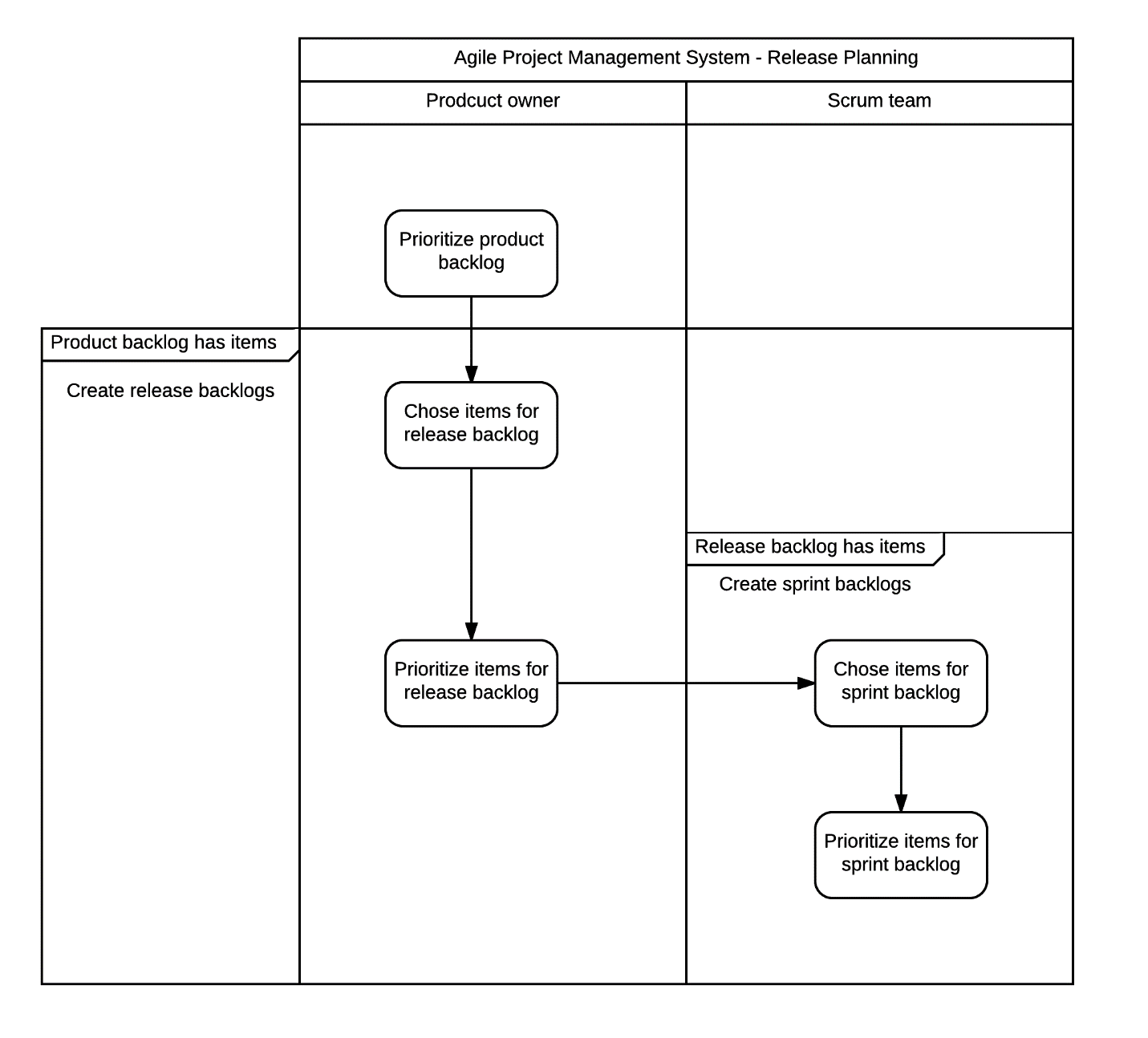


Figure 2: Release planning activity diagram

## Activity diagram 2

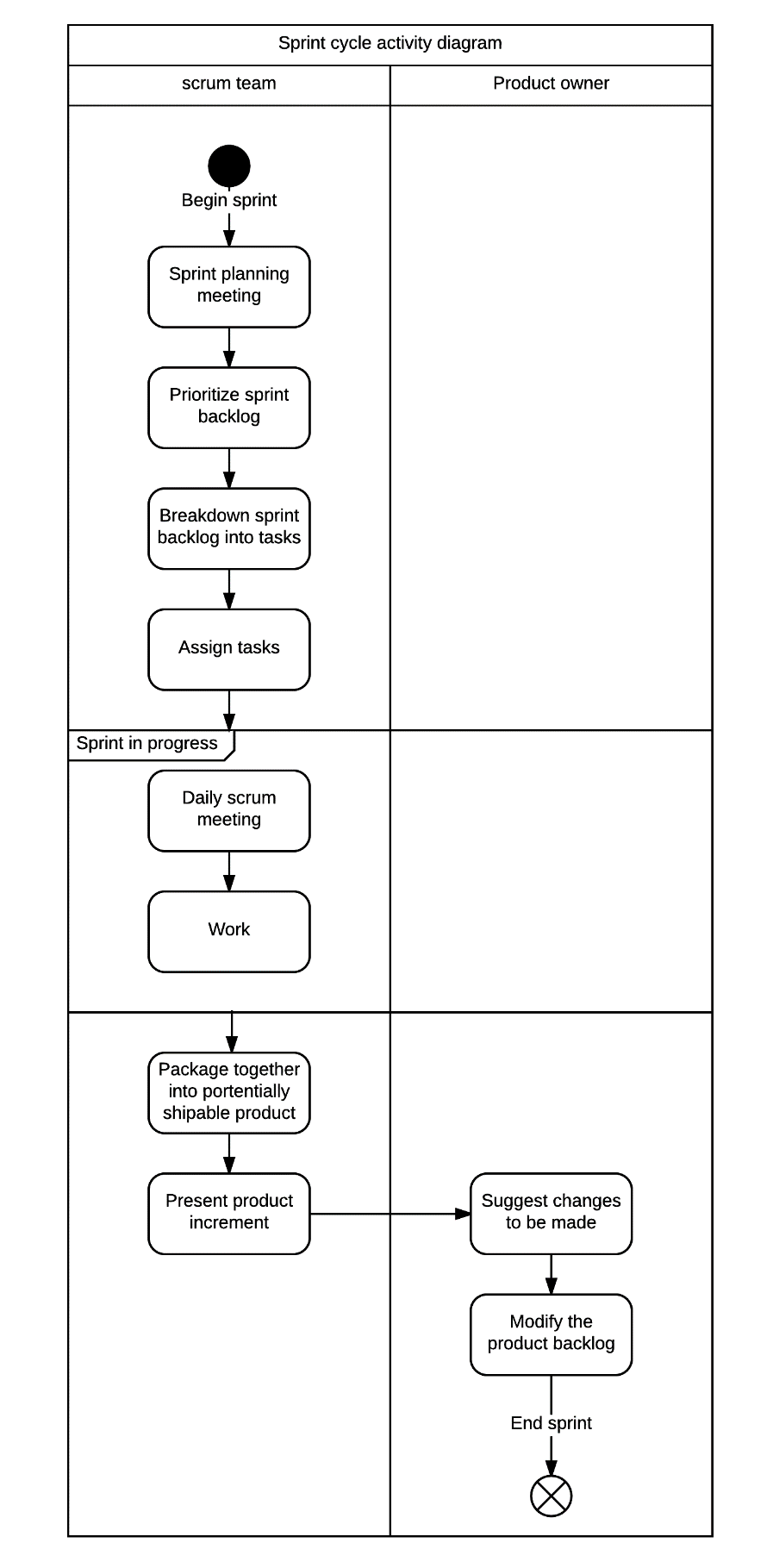


Figure 3: Sprint cycle activity diagram

## Class diagram

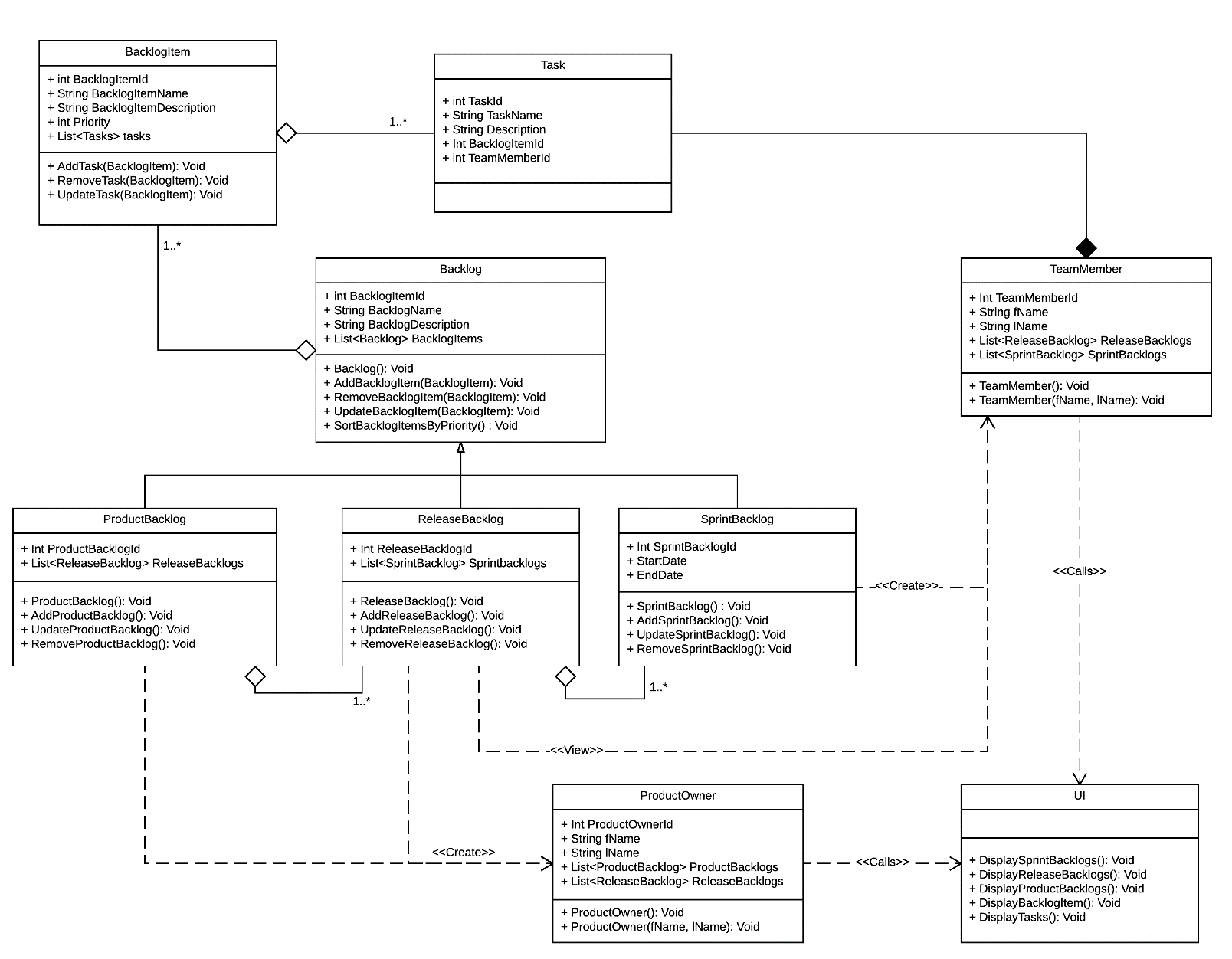
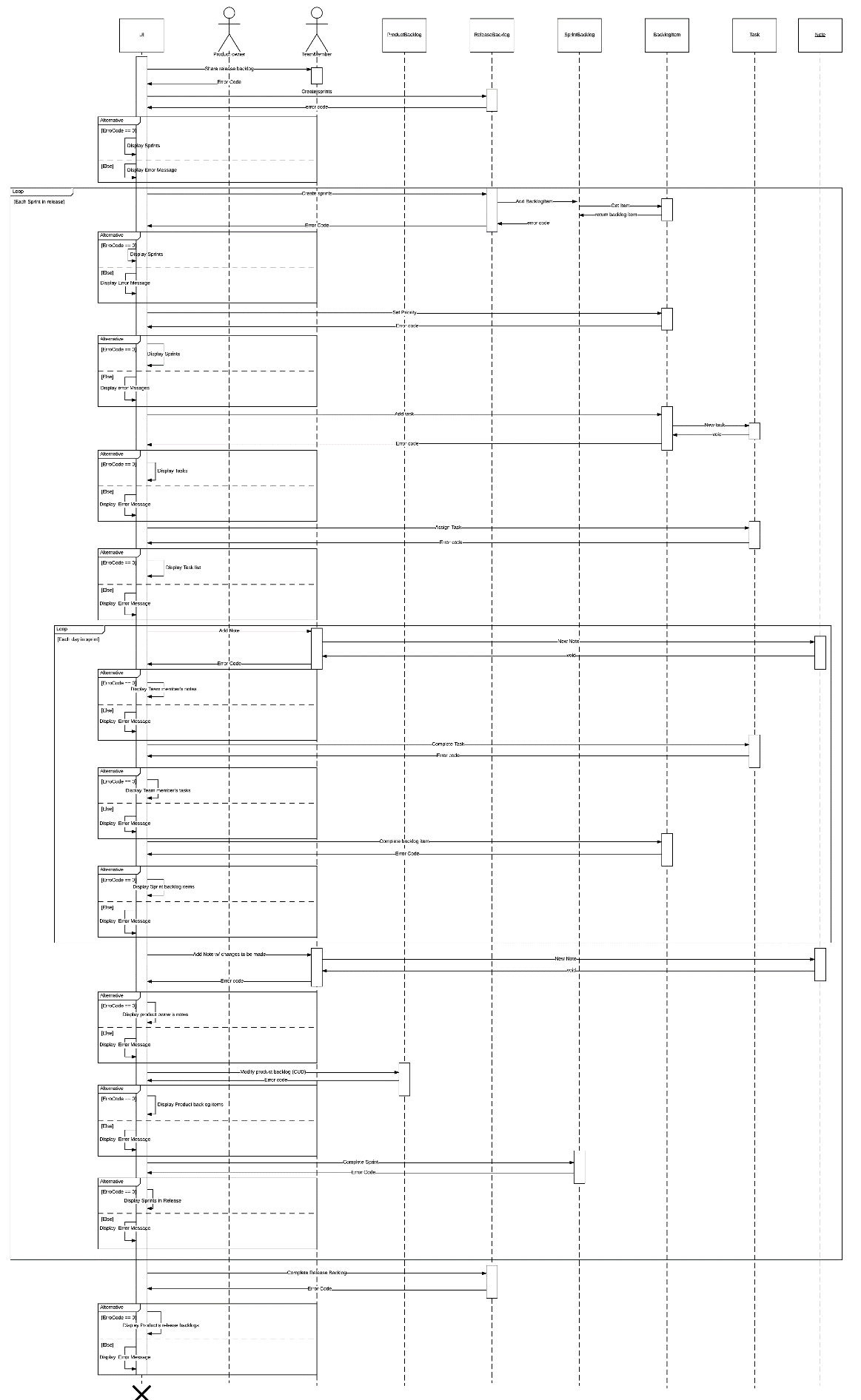


Figure 4: Agile Project Management System Class Diagram.

## Sequence Diagram



## Sprint backlog

|  |  |  |
| --- | --- | --- |
| Task | Time | Description |
| Design UI | 8 | The UI needs to be designed for a few different pages. It can be done as one page displaying different types of data since it’s all likely very similar what you’ll be showing on each page; or you can do it all in different pages for each different type of backlog/task. |
| UI.DisplaySprintBacklogs | 6 | This method will have to deal with grabbing the list of SprintBacklogs for the current ReleaseBacklog and display them to the user, which will likely need to be dynamically created. This will need to dynamically generate selectable date to bring you to the next page with a parameter passed to the page to know which tasks to load. |
| UI.DisplayReleaseBacklog | 6 | This method will have to grab all of the ReleaseBacklogs for the current ProductBacklog. It will need to be passed the product backlog in some sort of state parameter to know which release backlogs to load up. It will need to dynamically generate the table of release backlogs so that they can be selected to view all the SprintBacklogs. |
| UI.DisplayProductBacklogs | 6 | This method will need to display all of the Product backlogs for the Owner currently logged into the system. It will need to be passed an owner number to display all of their ProductBacklogs. |
| UI.DisplayBacklogItem | 8 | This method will be more complicated since you need to find a way to know which type of backlog this item belongs to. Once you determine that, you can grab the items from the list and just display them |
| UI.DisplayTasks | 6 | This won’t be any different than displaying a backlog item, you just need to find the SprintBacklog associated with it and |
| UI.DisplayErrorMessage | 2 | This will just print the error message to the screen that was passed to it from any of the Add/Update/Remove methods. It will display the error at the top of the screen, and won’t block the regular flow. |
| ProductBacklog.AddProductBacklog | 2 | Adding the item to the DB isn’t going to be hard, it’s just a quick insert. The Add/Update/Remove methods will contain all the DB connections, so there’s no separate DB layer. Since this is the first time an Add method is being written, I’ve upped the amount of time, so that you can work out any bugs before modifying it to use in other classes add methods. |
| ProductBacklog.UpdateProductBacklog | 2 | This is slightly harder since you need to find an existing record and modify it, but it still won’t take too long. You can likely just take the Add method and modify it to work with Updating. Since this is the first time an Update method is being written, I’ve upped the amount of time, so that you can work out any bugs before modifying it to use in other classes’ Update methods. |
| ProductBacklog.RemoveProductBacklog | 2 | This method will require just a simple delete in the database, and shouldn’t take very long. Since this is the first time a Remove method is being written, I’ve upped the amount of time, so that you can work out any bugs before modifying it to use in other classes’ Remove methods. |
| ReleaseBacklog.AddReleaseBacklog | 1 | This can likely be just copying some code from the ProductBacklog’s Add method and modifying it to work on the ReleaseBacklog table instead. It won’t take long to implement. |
| ReleaseBacklog.UpdateReleaseBacklog | 1 | Since writing the update method has already been done in the ProductBacklog class, it won’t take too long to reimplement it in this class. |
| ReleaseBacklog.RemoveReleaseBacklog | 1 | The delete has already been done as well, so it can be copied. Ideally, since there’s a lot of copying stuff so far, you could create a single class that you just pass a select statement to that will handle any table. |
| ReleaseBacklog.SortSprintsByDate | 4 | This will require writing a sort method that uses the object’s BacklogItem list priority attribute. This shouldn’t take 4 hours, but just giving it extra time in case something goes wrong. All it should require is a quick implantation of a quicksort or merge sort algorithm. |
| SprintBacklog.AddSprintBacklog | 1 | This will be minimal work again, you’re pretty much just changing the table name in the select statement. |
| SprintBacklog.UpdateSprintBacklog | 1 | Just changing the table name and columns grabbed in the select statement from the UpdateReleaseBacklog method. |
| SprintBacklog.RemoveSprintBacklog | 1 | More copy paste from other classes. Just change the table name and the primary key, that’s really all that needs to be done. |
| Backlog.AddBacklogItem | 1 | This method is essentially the same code as the other Add methods. You have a couple different attributes in your BacklogItem class than in your Backlog classes, so things will need to change up a bit, but it won’t be too long. |
| Backlog.UpdateBacklogItem | 1 | Again, you can copy paste your other Update methods into here and modify the attributes getting passed in and modified. |
| Backlog.RemoveBacklogItem | 1 | Same stuff as the other deletes since all you’re passing to it is a primary key. Really just need to change a variable name here. |
| Backlog.SortItemsByPriority | 4 | This will require the same stuff as the SortSprintsByDate method, except now it will be using the priority in the BacklogItem class as opposed to the startDate in the SprintBacklog class. It will also be using a different object type. It would be a lot faster to use the same sorting algorithm for both classes, but it might be favorable to use a different algorithm on each one depending on how much data you’re actually dealing with. |
| BacklogItem.AddTask | 1 | This is some more copied code, but you’ll have to change all the different attributes, which won’t take long. |
| BacklogItem.UpdateTask | 1 | Same as the Add, just copy one of the Update methods and change some attributes and variable names. |
| BacklogItem.RemoveTask | 1 | Again, Copy a Remove method and change some attributes and variables |
| BacklogItem.SortTasksByPriority | 2 | This should just be implementing the same sort as the Backlog.SortItemsByPriority, except you change the Data type you’re working with to be Task instead of BacklogItem. This time should be overestimated, but I’ll leave it at 2 hours just in case. |
| Write unit tests | 10 | Unit tests should take more than 10 hours, but since this is an agile system, I allotted less time to the unit tests since they can get completed over the course of multiple sprints. Right now, we want to get this part of the system working, minor bug fixes can come later, as long as the system works for now. But 10 hours will not cover all tests. |

# Part B: System Architecture

## Choosing the architecture

This system would be designed in a three-tier architecture. The system has 3 distinct tiers; the client, the web server, and the database server. Those are the parts of the system that will be involved in this system, while the Microsoft Exchange server exists for email, it’s not actually a part of the system. It will get used by the system administrators, but it’s another system that’s separate from the rest of the actual event registration system.

## Layering the application

The lowest layer of the system architecture would be the data access layer, which would exist within stored procedures in the database for this system. The Business logic will execute within the web server. The business logic layer inside the web server would manipulate data and store it within classes to pass off to the client, which would act as the presentation layer. The client would see the data passed to it from the business logic and format it to be displayed in a presentable way.

## Deployment Diagram

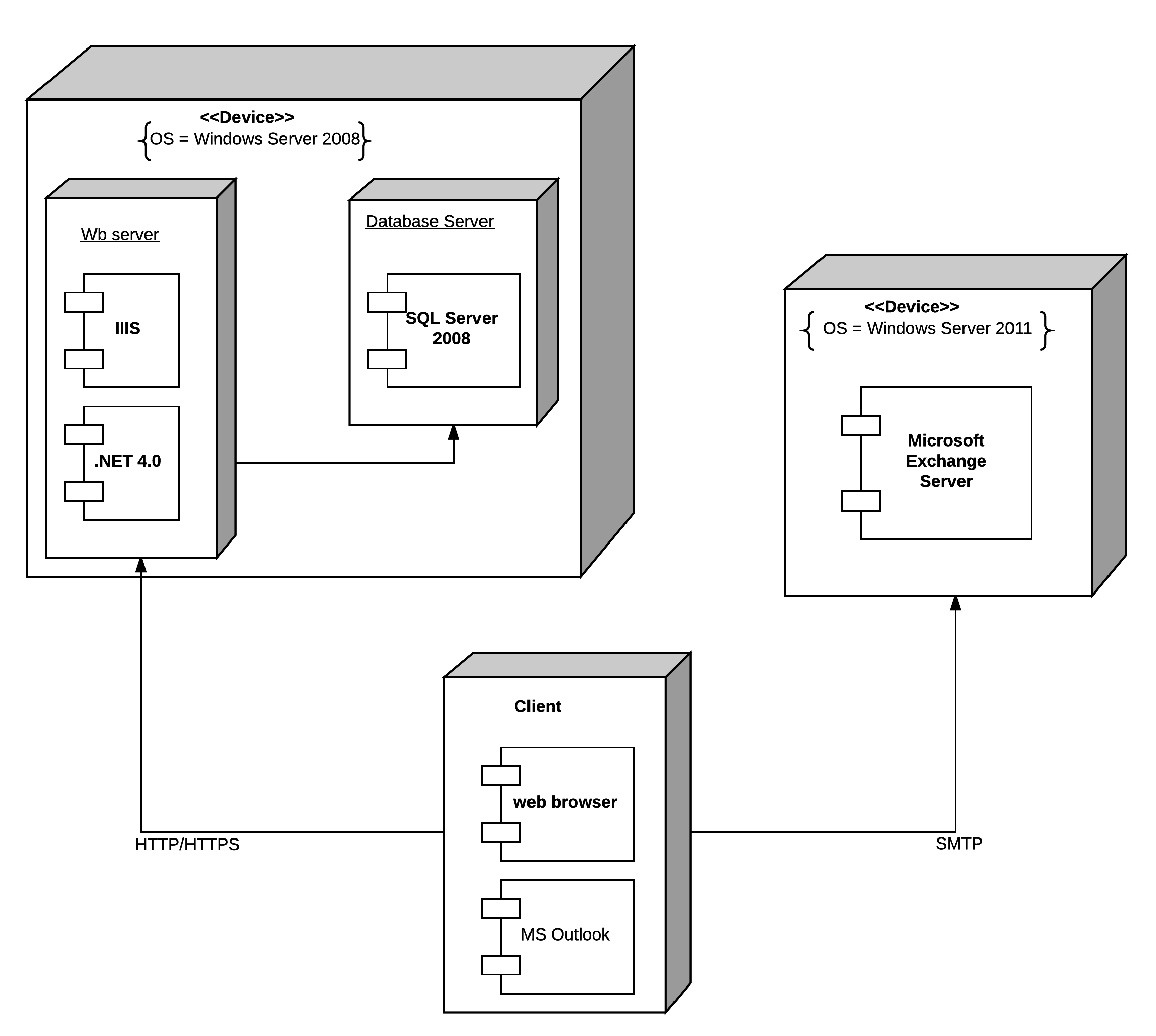


Figure 6: Deployment Diagram for Event Registration System

# Part C: System Development Methodologies

## Choosing the methodology

For this team and project, I would choose to follow a structured approach such as waterfall development. With waterfall development, you can fully take advantage of knowing all of the system requirements prior to beginning development. With a military system, you also want to make sure that everything is as per specification, so following a waterfall development gives the team lots of opportunity to make sure than anything very specific that isn’t in the pre-existing documentation is noted inside the documents. Following a waterfall development is also probably easier for the team, since they aren’t familiar with any particular approach. Teaching the basics of waterfall development to a development team is much easier than teaching about rapid application development or agile methodologies. It’s a very simple approach, and it is effective.

Nothing in the specifications asks for a time or money constraint either, so even if waterfall development takes a little bit longer due to more requirements for extensive documentation, that doesn’t seem to be much of an issue with this system. The organization building the system is also an external organization, so having extensive documentation makes the system more maintainable once it’s been passed off to the military. All of the black-box requirements have been specified, but all of the technical documentation still needs to be written, so handing off all the technical documentation to the military will be better than handing them a system that they know nothing of the inner workings.