Project Work - Week 3

This is the third week of a repeated task where I attempt to improve week by week, showing skills I learn, the code I create and documenting them effectively.

I will provide the topic discussions, just like the previous 2 weeks, as follows:

- Provide a descriptive summary of the task I chose.
- Provide snippets of the code I created with a commentary to show good coding practices.
- A descriptive summary of the test code I wrote.
- A reflective summary of any changes requested during code review, along with fixes if required.
- A descriptive summary of the issues I found with the code I was asked to review.
- The final step will be a reflection with additional content pertaining to improvements and so on.

Descriptive Summary

This week was different from previous weeks, as this week I worked on two tasks. The reason for this, is that the first task I took, I had a lot of the code ready for it. This was unintentional, and after completion I wasn't satisfied so after confirming with my team, I took another task.

The first task was to simply remove an employees access from the system. However, with my work in previous weeks, I already had methods in place to altar employees. Instead of deleting an employee, instead, they would have their status changed.

The second task was related to assigning resources to operations on request. This was done by creating a separate table in the database to hold equipment, then a piece of equipment could be assigned to an operation.

Code Commentary

The first method this week is as follows:

```
/// <summary>
/// This task will change the users status
/// </summary>
/// <param name="item"></param>
/// <returns></returns>
public async Task<int> ChangeStatusAsync(TeamMemberModel employee)
{
    await Init();

    employee.access = false;

    return await Database.UpdateAsync(employee);
}
```

This first method will take in an employee, which is an instance of "TeamMemberModel", then it will alter that users status, updates this change in the database and returns an int.

This method follows all good coding principles, with examples of KISS, as it is a simple method with no code that's difficult to understand. It follows YAGNI, as there is no unused code. It follows DRY as there is no repeated code, it has doxygen suitable comments which explain what the code does in a simple and informative manner. It follows the single responsibility principle as it only does one thing. These are a few examples, however, the code in question appears to follow all coding principles.

The second method this week is as follows:

```
/// <summary>
/// Assigns a piece of equipment to a team
/// </summary>
/// <param name="equipmentId"></param>
/// <param name="teamId"></param>
/// <returns></returns>
public async Task<bool> AssignEquipmentToTeam(int equipmentId, int teamId)
{
    var equipment = await database.Table<Equipment>().Where(i => i.ID ==
equipmentId).FirstOrDefaultAsync();
    if (equipment != null)
    {
        equipment.AssignedTeamId = teamId;
        await database.UpdateAsync(equipment);
        return true;
    }
    return false;
}
```

This method is more complex than the previous method, however, it still follows all coding standards.

This will assign a piece of equipment, designated by its ID, to a team which is also designated by its ID. When it is called, it attempts to get the equipment item from the database. If the equipment item is found, it will then be assigned to the team specified previously. If the equipment is not found it will return a boolean value of false.

This code, even with its increased complexity, still follows all good coding principles. Examples include YAGNI, as theres no unused code, KISS, as its simple and well explained, DRY as it doesn't have repeated code, the single responsibility principle as it only does the task that its supposed to.

Testing

The test I will be discussing is as follows:

```
/// <summary>
/// Changes a users status then asserts that it was done successfully
/// </summary>
[Fact]
```

```
public void ChangeAndCheckUserStatus()
{
    var teamMemberDB = new TeamMemberDB();

    String ID = "1";

    teamMemberDB.ChangeUserStatus(ID, false);

    bool result = teamMemberDB.CheckUserAccess(ID);

    Assert.False(result);
}
```

This test was created to check the status of a user after its status is changed. It does this by altering a dummy member with the ID 1, then checking that the status has changed successfully.

This test is very simplistic, however, it is vital that the method of altering a users status is working for this weeks task, thus, it is essential to test.

However, this code again follows good coding conventions, with a good explanation in the doxygen comments, well named variables and it is simple and easy to understand.

Code Review Results

This week I saw that my team were trying harder with their tasks. I saw many good and well written pieces of code.

The issue I have found is that I haven't been able to find any obvious issues, which of course, leaves this section of my portfolio lacking. The only exception is with comments. People forget to comment properly, or sometimes they may use variables with less than ideal names. But that of course isn't very interesting.

The same can be said about myself this week. However, this could be in part due to the small tweaks I had to make overall to the existing codebase on altering a members status. This of course would mean they wouldn't have much to look at.

However, we are all learning, and there are most likely many issues with unknown coding smells and the like that we haven't been able to spot. But, I know that my ability to find these issues is improving.

Reflection

This week I have attempted to improve in several areas. Of course, the obvious being with code quality. However, I have as mentioned in previous entries, been working on creating mocks. This went better this week, but the issue I believe I am having is with the way this application is built. It doesn't follow MVVM, which is the second thing Ive been looking into this week. Due to this, I find that mocking is proving to be bothersome. I have plans to create my own program, following MVVM to attempt to implement a mock in the future.

The third thing I have been working on this week is lambda expressions. This could be very basic knowledge, however, as someone with no previous experience with C#, this isn't something I've ever done. The concept isn't easy for me, but I have tried none the less, and have implemented it into this weeks work.

As it could be gathered from my previous writings, my intention with this isn't to simply do the coursework, but it is to improve as a programmer overall. Thus looking into things which aren't expected has become somewhat of a habit.

As these weeks go on, I can see a huge improvement overall, but I still am encountering a lot of code I don't understand. A lot of this has to do with the way C# differs from the languages I have previously done.

I find my coding has improved a lot due to following good coding practices. Most of which are becoming second nature now. Like the single responsibility principle or the major principles like YAGNI, KISS and so on. This is of course great news as that is the aim of the course.

However, in saying this, I can't help but feel that I have a mountain to climb. I find new features and pieces of code every day that I don't fully understand. Lambda is one. It can be something incredibly simple but immediately turned up to a 10 where it somehow contains 4 lines of code.

One issue which is also hard for me is working within a team. I understand the need for this, and welcome it, however, having numerous people implementing code in their own unique ways also creates its own set of problems. As we are all at different levels of understanding, one person can have very easy and basic code while the other has code written in what appears to be another language.