Coursework Cover Sheet

Students should complete the input fields contained in this form and attach it in front of your formal assessment submission. All fields within this form are required. Please ensure that check boxes and radio buttons are appropriately selected. The last three questions are just for you to personally consider.

# Department and assessment information:

**School Name: School of Science**

**Assessment title:** Test-Driven Development of a Smart Building Controller Class

**Course Title:** BSc (Hons) Computing

**Module Title:** Software Development

**Module Code:** C02401

**Year of Study:** 2023

# Academic Misconduct / Plagiarism Declaration

By attaching this front cover sheet to my assessment I confirm and declare that **I am the sole author of this work**, except where otherwise acknowledged by appropriate referencing and citation, and that I have taken all reasonable skill and care to ensure that no other person has been able, or allowed, to copy this work in either paper or electronic form, and that prior to submission I have read, understood and followed the University regulations as outlined in the [Academic Integrity Policy and Procedure for Academic Misconduct](https://www.uclan.ac.uk/assets/student-contracts/2023-24/academic-integrity-policy-2324.pdf)

# Have you checked the following? This will help your assessment achievement.

I have applied the learning outcomes for this module

I have checked for Academic Integrity via Turn-it-in

I have followed the guidance in the Assessment Brief and have not used AI to boost my grade unfairly.

I have used references in accordance with instructions in the Assessment Brief

I have proofread my work for spelling, grammar and punctuation.

I have checked that the word count/size of this submissions accords with the guidance provided in the Assessment Brief.

# Well-being

We wish to support any student who is experiencing mitigating circumstances which prevents students from performing to the best of their ability when completing or submitting assignments. If you are experiencing such circumstances, then you may apply for Mitigating Circumstances**.** Wherever possible this must be done prior to handing in your assignment.

Do you need to apply for mitigating circumstances for this assignment No

Please refer to the [Mitigating Circumstances Policy](https://msuclanac.sharepoint.com/sites/CyprusStudentHub/SitePages/Mitigating-Circumstances.aspx)

# Questions you may wish to consider:

1. Have I allowed sufficient time to prepare this assessment? \_\_\_YES\_\_\_\_\_
2. Have I reflected on previous feedback and made improvements in accordance with advice? \_\_\_\_YES\_\_\_\_
3. What grade am I expecting? \_\_\_\_100\_\_\_\_

**Test-Driven Development (TDD) of a Smart Building Controller Class**

Table of Contents

[Department and assessment information: 1](#_Toc158327719)

[Academic Misconduct / Plagiarism Declaration 1](#_Toc158327720)

[Have you checked the following? This will help your assessment achievement. 1](#_Toc158327721)

[Well-being 2](#_Toc158327722)

[Questions you may wish to consider: 2](#_Toc158327723)

[1. Overview of TDD 4](#_Toc158327724)

[2. L1R1- First iteration of the RED-GREEN-REFACTOR cycle 5](#_Toc158327725)

[2.1 RED state 5](#_Toc158327726)

[2.1 GREEN state 5](#_Toc158327727)

[2.1 REFACTOR state 5](#_Toc158327728)

[3. L2R3 – Level 2 RED-GREEN-REFACTOR cycle 6](#_Toc158327729)

[3.1 RED state 6](#_Toc158327730)

[3.2 GREEN state 6](#_Toc158327731)

[3.3 REFACTOR state 6](#_Toc158327732)

[4. L3R3- Level 3 RED-GREEN-REFACTOR cycle 7](#_Toc158327733)

[4.1 RED state 7](#_Toc158327734)

[4.2 GREEN state 7](#_Toc158327735)

[4.3 REFACTOR state 7](#_Toc158327736)

[5. L4R1- Level 4 RED-GREEN-REFACTOR cycle 8](#_Toc158327737)

[5.1 RED state 8](#_Toc158327738)

[5.2 GREEN state 8](#_Toc158327739)

[5.3 REFACTOR state 8](#_Toc158327740)

[6. STD – Valid States and Valid Transitions 9](#_Toc158327741)

[7. Stubs and Mocks 9](#_Toc158327742)

[8. Lessons Learned and Conclusion 9](#_Toc158327743)

[References 9](#_Toc158327744)

# Overview of TDD

The strategy to develop the BuildingController class was to first analyse the supposed structure of the project, in other words, where the system will be operating, what is its primary purpose, desired functionality and security considerations. Additionally, the functional requirements were analysed. After the initial analysis, the main class and its dependencies were outlines, i.e. all the members, all the methods and their signatures were coded, so that when developing the initial tests there would not be a necessity to create tests for existence/non-existence of these methods. Following that a standard test-driven development was used: first, develop the test for the future implementation, and then implement the actual code itself. In this development the red-green-refactor cycle was as follows: complete all of the tests and functionality and be in the green state and after that – refactor. This approach presents several benefits: ensures that test will definitely be written since making them after the developments usually seems redundant; forces the developer to think ahead and form the intended behaviour of code rather than the code itself; provides motivation for completing steps after steps - especially when the tests are small and fast to make (Larman, 2004). The challenges, however, are fairly obvious – this style of coding is unfamiliar and thus the process is much slower; there must be approximately twice as much code written, if not more; lastly, bugs can also be introduced in tests themselves. In relation to the Smart Building project specifically, one of the most prominent challenges was to make small tests, thus eradicating possible errors in them. This was especially due to the fact that most of the functionality of the class is clumped within one of the methods (‘SetCurrentState’). To combat this, the code inside the method was split into several blocks each for a separate case of its usage (for ‘fire alarm’, ‘fire drill’ and the rest). This allowed to comment out and test separate blocks of code.

Literature lists several skills related to TDD that increase employability: extended discipline in coding standards, better software design and ability do produce maintainable code (Janzen, 2005). One could argue that any job title which is related to programming can benefit from test-driven development and skills that it requires. However, there are more suitable roles that practically require TDD, for example, Test Automation engineer, QA engineer, DevOps, software architect, release manager and so on (Rajani, 2017).

|  |  |  |
| --- | --- | --- |
| Level | Number of Tests | Number of Test Cases |
| Level 1 | 4 | 8 |
| Level 2 | 4 | 26 |
| Level 3 | 4 | 2 |
| Level 4 | 3 | 5 |
| Totals: | 15 | 41 |

# L1R1- First iteration of the RED-GREEN-REFACTOR cycle

## 2.1 RED state

The test for this requirement has to be combined with the second requirement (L1R2). This is because the constructor in R1, sets the buildingID, but there is no way to test if the constructor has worked correctly, since the building ID is private and there is no getter method yet. Combining it with the second requirement allows to call the constructor and then check that it set the id correctly with a GetBuildingID method. For this test only the requirements were consulted with (appendix B).

[Test]

//L1R1\_L1R2

public void L1R1\_L1R2\_Constructor\_SetBuildingID()

{

//Arrange

string newBuildingID = "test";

var controller = new BuildingController(newBuildingID);

//Act

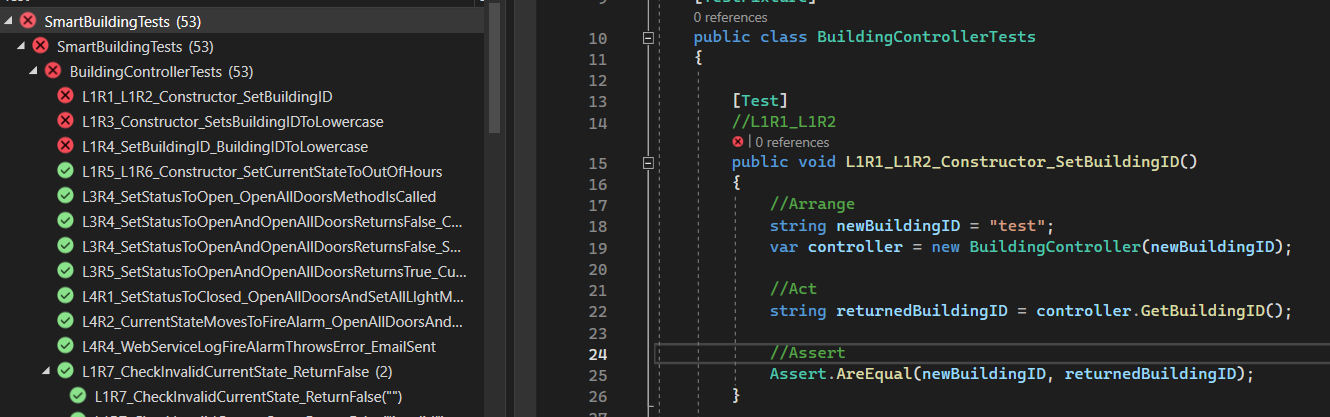
string returnedBuildingID = controller.GetBuildingID();

//Assert

Assert.AreEqual(newBuildingID, returnedBuildingID);

}

The test is failing because there is no constructor, or a getter method in the class.



## 

## 2.1 GREEN state

To go to the green state, the necessary code was added to the class.

Before:

**public BuildingController(string buildingID)**

**{**

**}**

**public string GetBuildingID()**

**{**

**return "";**

**}**

After:

**public BuildingController(string buildingID)**

**{**

**this.buildingID = buildingID.ToLower();**

**this.currentState = "out of hours";**

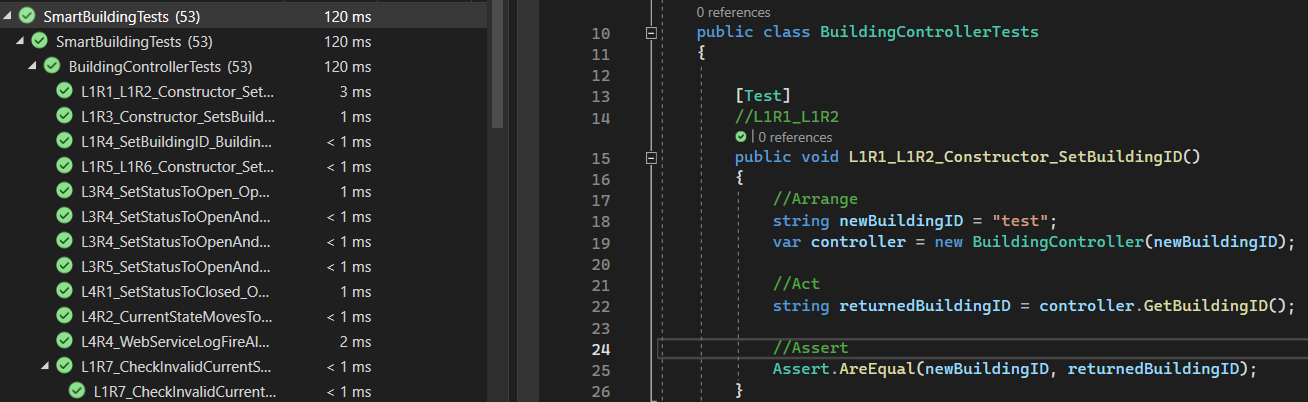
**}**

**public string GetBuildingID()**

**{**

**return buildingID;**

**}**



## 2.1 REFACTOR state

During refactor state nothing was done. This is due to the fact that there is nothing to be done.

# L2R3 – Level 2 RED-GREEN-REFACTOR cycle

## 3.1 RED state

To test this requirement, it was broken into two parts:

The first is when the constructor sets the state to lowercase when initializing and, second part, when it throws an exception when it is a wrong state. Only the requirements were consulted for this.

First test:

**[TestCase("out of hoUrs")]**

**[TestCase("clOsed")]**

**[TestCase("opeN")]**

**//L2R3**

**public void L2R3\_SetCurrentStateToUpperCase\_SetsToLowerCase(string testState)**

**{**

**//Arrange**

**var controller = new BuildingController(buildingID: "testID", startState: testState);**

**//Act**

**string setState = controller.GetCurrentState();**

**//Assert**

**Assert.AreEqual(testState.ToLower(), setState);**

**}**

Second test:

**[TestCase("fire drill")]**

**[TestCase("fire alarm")]**

**//L2R3**

**public void L2R3\_SetCurrentStateToWrongState\_ThrowException(string testState)**

**{**

**string correctMessage = "Argument Exception: BuildingController can only be initialised to the following states 'open', 'closed', 'out of hours'";**

**//Arrange & Act & Assert**

**Assert.Throws(Is.TypeOf<ArgumentException>()**

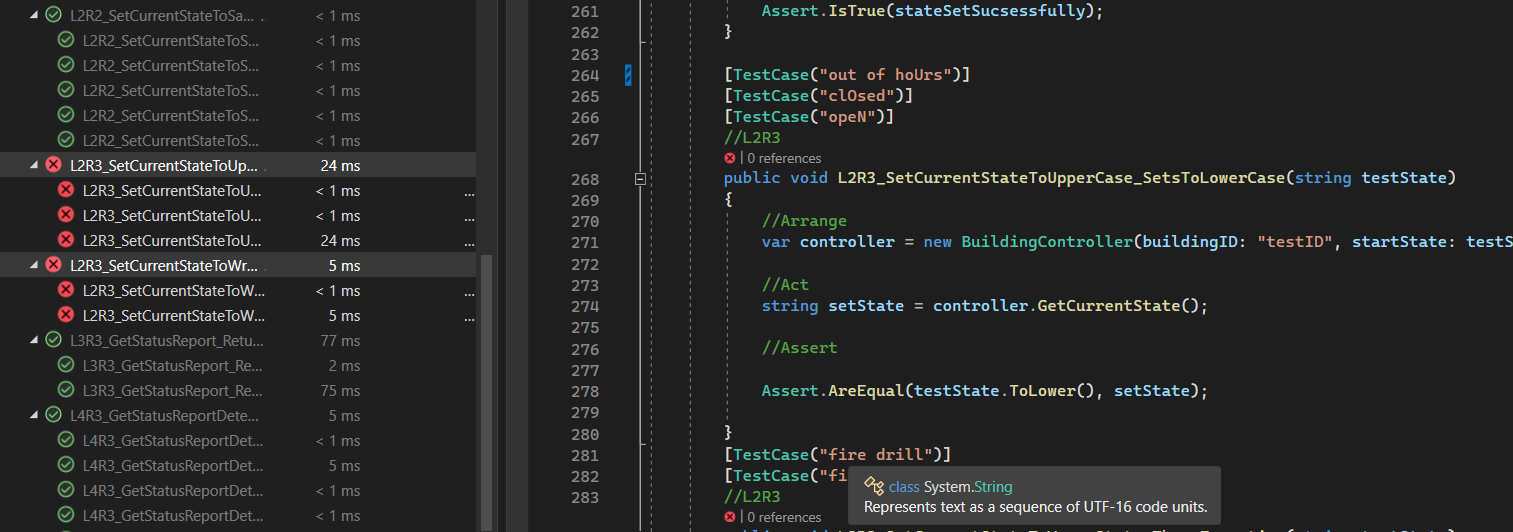
**.And.Message.EqualTo(correctMessage),**

**() => new BuildingController(buildingID: "testID", startState: testState)**

**);**

**}**

This is in the red state because there is yet no functionality for these tests.



## 3.2 GREEN state

To go to the green state, the necessary code was added to the class.

Before:

**public BuildingController(string buildingID, string startState)**

**{**

**}**

After:

**public BuildingController(string buildingID, string startState)**

**{**

**this.buildingID = buildingID.ToLower();**

**startState = startState.ToLower();**

**if (validStates.Contains(startState))**

**{**

**this.currentState = startState;**

**}**

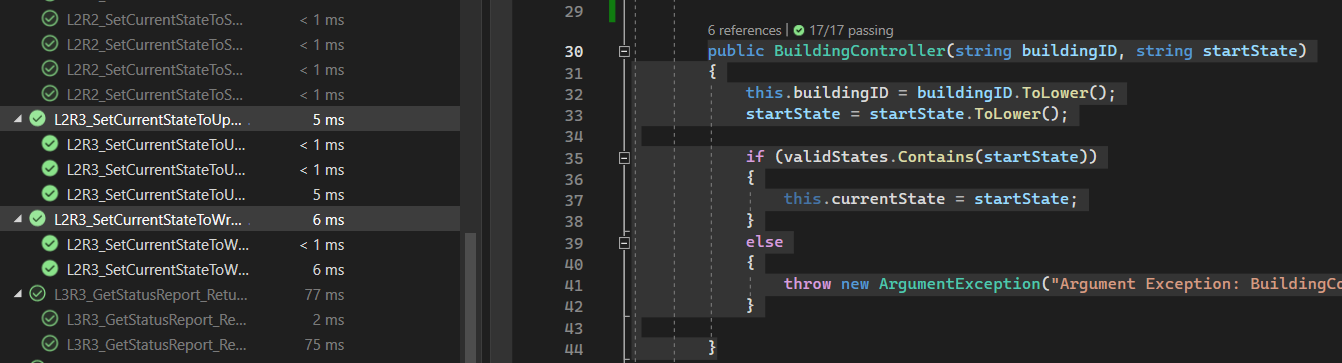
**else**

**{**

**throw new ArgumentException("Argument Exception: BuildingController can only be initialised to the following states 'open', 'closed', 'out of hours'");**

**}**

**}**

****

## 3.3 REFACTOR state

During refactor state nothing was done. This is due to the fact that there is nothing to be done.

# L3R3- Level 3 RED-GREEN-REFACTOR cycle

## 4.1 RED state

To fulfil this requirement, first the code for the test was created and then the functional code was created. Only the requirements part of the brief was consulted with.

**public void L3R3\_GetStatusReport\_ReturnsCorrectString(string testString1, string testString2, string testString3 )**

**{**

**//Arrange**

**ILightManager lightManager = Substitute.For<ILightManager>();**

**lightManager.GetStatus().Returns(testString1);**

**IDoorManager doorManager = Substitute.For<IDoorManager>();**

**doorManager.GetStatus().Returns(testString2);**

**IFireAlarmManager fireAlarmManager = Substitute.For<IFireAlarmManager>();**

**fireAlarmManager.GetStatus().Returns(testString3);**

**IWebService webService = Substitute.For<IWebService>();**

**IEmailService emailService = Substitute.For<IEmailService>();**

**var controller = new BuildingController("testID", lightManager, fireAlarmManager, doorManager, webService, emailService );**

**//Act**

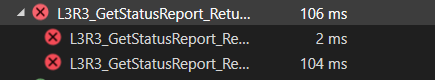
**var report = controller.GetCurrentReport();**

**//Asset**

**Assert.AreEqual (testString1 + testString2 + testString3, report);**

**}**

The test is in the red state because there is no functional code for it yet.



## 4.2 GREEN state

To go to the green state, the necessary code was added to the class.

Before:

**public string GetCurrentReport()**

**{**

**return "";**

**}**

After:

**public string GetCurrentReport()**

**{**

**if (doorManager != null && lightManager != null && fireAlarmManager != null && webService != null)**

**{**

**string faultyLights = lightManager.GetStatus().Contains("FAULT") ? "Lights," : "";**

**string faultyDoors = doorManager.GetStatus().Contains("FAULT") ? "Doors," : "";**

**string faultyFireAlarm = fireAlarmManager.GetStatus().Contains("FAULT") ? "FireAlarm," : "";**

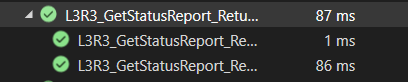
**webService.LogEngineerRequired(faultyLights + faultyDoors + faultyFireAlarm);**

**return lightManager.GetStatus() + doorManager.GetStatus() + fireAlarmManager.GetStatus();**

**}**

**return "";**

**}**

****

## 4.3 REFACTOR state

During refactor stage a check for null for dependencies was added.

Before :

**public string GetCurrentReport()**

**{**

**return lightManager.GetStatus() + doorManager.GetStatus() + fireAlarmManager.GetStatus();**

**}**

After:

**public string GetCurrentReport()**

**{**

**if (doorManager != null && lightManager != null && fireAlarmManager != null && webService != null)**

**{**

**string faultyLights = lightManager.GetStatus().Contains("FAULT") ? "Lights," : "";**

**string faultyDoors = doorManager.GetStatus().Contains("FAULT") ? "Doors," : "";**

**string faultyFireAlarm = fireAlarmManager.GetStatus().Contains("FAULT") ? "FireAlarm," : "";**

**webService.LogEngineerRequired(faultyLights + faultyDoors + faultyFireAlarm);**

**return lightManager.GetStatus() + doorManager.GetStatus() + fireAlarmManager.GetStatus();**

**}**

**return "";**

**}**

# L4R1- Level 4 RED-GREEN-REFACTOR cycle

## 5.1 RED state

To fulfil this requirement, first the code for the test was created and then the functional code was created. The requirements and Class Diagram part of the brief was consulted with.

**[Test]**

**//L4R1**

**public void L4R1\_SetStatusToClosed\_OpenAllDoorsAndSetAllLIghtMethodsAreCalled()**

**{**

**//Arrange**

**ILightManager lightManager = Substitute.For<ILightManager>();**

**IDoorManager doorManager = Substitute.For<IDoorManager>();**

**IFireAlarmManager fireAlarmManager = Substitute.For<IFireAlarmManager>();**

**IWebService webService = Substitute.For<IWebService>();**

**IEmailService emailService = Substitute.For<IEmailService>();**

**var controller = new BuildingController("testID", lightManager, fireAlarmManager, doorManager, webService, emailService);**

**//Act**

**controller.SetCurrentState("closed");**

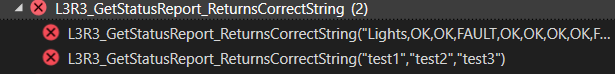
**//Assert**

**doorManager.Received().LockAllDoors();**

**lightManager.Received().SetAllLights(false);**

**}**

The test is in the red state because there is no functional code for it yet.

****

## 5.2 GREEN state

To go to the green state, the necessary code was added to the class.

Before:

**public string GetCurrentReport()**

**{**

**return "";**

**}**

After:

**public string GetCurrentReport()**

**{**

**if (doorManager != null && lightManager != null && fireAlarmManager != null && webService != null)**

**{**

**string faultyLights = lightManager.GetStatus().Contains("FAULT") ? "Lights," : "";**

**string faultyDoors = doorManager.GetStatus().Contains("FAULT") ? "Doors," : "";**

**string faultyFireAlarm = fireAlarmManager.GetStatus().Contains("FAULT") ? "FireAlarm," : "";**

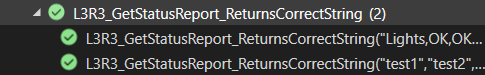
**webService.LogEngineerRequired(faultyLights + faultyDoors + faultyFireAlarm);**

**return lightManager.GetStatus() + doorManager.GetStatus() + fireAlarmManager.GetStatus();**

**}**

**return "";**

**}**



## 5.3 REFACTOR state

During refactor stage an additional if condition was added to check for nulls.

Before:

**public string GetCurrentReport()**

**{**

**return lightManager.GetStatus() + doorManager.GetStatus() + fireAlarmManager.GetStatus();**

**}**

After:

**public string GetCurrentReport()**

**{**

**if (doorManager != null && lightManager != null && fireAlarmManager != null && webService != null)**

**{**

**return lightManager.GetStatus() + doorManager.GetStatus() + fireAlarmManager.GetStatus();**

**}**

**return "";**

**}**

# STD – Valid States and Valid Transitions

**if (!validInitialStates.Contains(state))**

**{**

**return false;**

**}**

**//returning to previous state from drill or alarm**

**if (currentState == "fire drill" || currentState == "fire alarm")**

**{**

**if(state == previousState || state == currentState)**

**{**

**currentState = state;**

**return true;**

**}**

**return false;**

**}**

**//set fire alarm**

**if (state == validInitialStates[4])**

**{**

**lightManager?.SetAllLights(true);**

**doorManager?.LockAllDoors();**

**fireAlarmManager?.SetAlarm(true);**

**try**

**{**

**webService?.LogFireAlarm("fire alarm");**

**}**

**catch (Exception e)**

**{**

**emailService?.SendMail("smartbuilding@uclan.ac.uk", "failed to log alarm", e.Message);**

**}**

**previousState = currentState;**

**currentState = state;**

**return true;**

**}**

**//set fire drill**

**if(state == validInitialStates[0])**

**{**

**previousState = currentState;**

**currentState = state;**

**return true;**

**}**

**//set regular state, i.e. close, open, out of hours**

**if (Math.Abs(Array.IndexOf(validInitialStates, state) - Array.IndexOf(validInitialStates, currentState)) <= 1)**

**{**

**if (doorManager != null && lightManager != null)**

**{**

**//State sets to open, but the doors fail to open**

**if (state == validInitialStates[1] && !doorManager.OpenAllDoors())**

**{**

**return false;**

**}**

**//open state**

**if (state == validInitialStates[3])**

**{**

**lightManager.SetAllLights(false);**

**doorManager.LockAllDoors();**

**return true;**

**}**

**}**

**currentState = state;**

**return true;**

**}**

**return false;**

This algorithm acts based on the type of state: first on ‘irregular’ states – fire drill, fire alarm – to determine what to do when states move to these states. Then it determines if the building returns to the correct previous state by comparing it to the saved state. Finally, it checks the correctness of the ‘regular’ states. It does so by checking the index of the new state compared to the index of the current state – if it is close enough. This also allow to extend the states laterally without much effort.

The STD were translated into tests which tested first the irregular behaviour – random strings, invalid states, etc. – and then tested for correct order of states. For example, that setting ‘closed’ when having ‘open’ state, and setting incorrect historic states.

# Stubs and Mocks

The purpose of a stub is to fake information that needs to go into the object under test from its dependency. A mock object is needed to test the object under testing is sending information into its dependency. For example, for the Building Controller class it needs to receive an exception from the email service and it needs to be faked. As for mock object, the fact that the object’s method is called is tested.

Stub:

**IWebService webService = Substitute.For<IWebService>();**

**IEmailService emailService = Substitute.For<IEmailService>();**

**webService.When(x => x.LogFireAlarm(Arg.Any<string>()))**

**.Do(x => { throw new Exception("fake exception"); });**

Mock:

**IDoorManager doorManager = Substitute.For<IDoorManager>();**

**doorManager.Received().OpenAllDoors();**

# Lessons Learned and Conclusion

While working on this project it was very beneficial to think ahead of creating the code. This helped immediately make design choices that definitely eliminated potential errors. IN the future it would be more beneficial to stick to the requirements more, for example, because an empty constructor was made, it allowed for multiple hidden failures of the tests, which resulted in big redesign of both tests and code. Additionally, in retrospect, an ADT for states could prove to be more manageable rather than a few nested ‘if statements’.

# 

# References

Janzen, D. and Saiedian H. (2005) *Test-Driven Development: Concepts, Taxonomy, and Future Direction*. IEEE Computer Society.

Larman, C. (2004) *Agile and Iterative Development: A Manager's Guide.* Addison-Wesley Professional.

Rajani, R. (2017) *Testing Practitioner Handbook.* Birmingham: Packt Publishing Ltd.