SDV502 Application Testing – Assessment 1

NUnit test cases

Tan Yi Xiong

[Yixiong-tan@live.nmit.ac.nz](mailto:Yixiong-tan@live.nmit.ac.nz)

Contents

[Introduction 3](#_Toc80178703)

[Arrange, Act and Assert 3](#_Toc80178704)

[[TestCase(…)] attribute 4](#_Toc80178705)

[Somethings to Note.. 4](#_Toc80178706)

[1. Adult\_Before\_5() 4](#_Toc80178707)

[2. Adult\_After\_5() 5](#_Toc80178708)

[3. Adult\_Tuesday() 5](#_Toc80178709)

[4. Child\_Under\_16() 6](#_Toc80178710)

[5. Senior() 6](#_Toc80178711)

[6. Student() 6](#_Toc80178712)

[7. Family\_Pass() 7](#_Toc80178713)

[8. Chick\_Flick\_Thursday() 7](#_Toc80178714)

[9. Kids\_Careers() 8](#_Toc80178715)

[Summary 8](#_Toc80178716)

# Introduction

Nelson State Cinema Ticket Prices

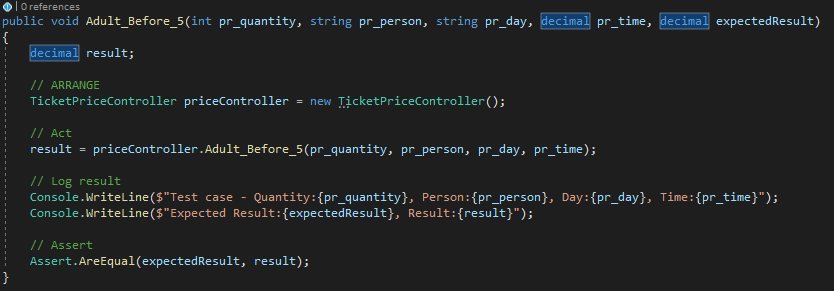


Using the Nelson State Cinema’s ticket prices, how can we build test cases to run unit tests to validate the functions in the cinema’s application? As a start, we were given 9 functions that were built in C# for the cinema’s app. The challenge given to us is to build test cases that can automate the testing in NUnit.

NUnit is an open-source unit testing framework that enable us to build test cases in C#. Using NUnit, I’ve built 5 cases each for the 9 functions which comes up to 45 test cases in total.

# Arrange, Act and Assert

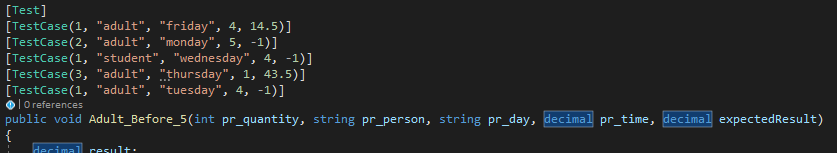
The pattern used to build test cases is the Arrange, Act and Assert (AAA) pattern. Here’s a sample taken from the code:



As we can see, the AAA pattern is used in this test function. We will see this in all of the test cases built in this challenge.

# [TestCase(…)] attribute

To make the code tidier, I’ve used the TestCase attribute to summarise similar test cases. Since there 9 functions to be tested, there will be 9 test functions and 5 test cases with the TestCase attribute. An example:



When the test is run, values in the TestCase will be used as parameters for the test function invoked.

Next, we will take a look at the 9 functions and each of their 5 test cases.

# Somethings to Note..

1. When expected result or result = -1, it can mean a few things:

* Any one of the inputs are Unacceptable
* Requirements not met for the function

1. I try to mix positive and negative tests in each of the 5 test cases created. So, you will notice that not all the test cases are all positive or all negative.

# Adult\_Before\_5()

Input: int pr\_quantity, string pr\_person, string pr\_day, decimal pr\_time, decimal expectedResult

Output: decimal result

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Equivalence Partitioning & Boundaries | | | | |
| Status | **quantity** | **person** | **day** | **time** |
| Acceptable | >0 | adult | Monday,  Wednesday,  Thursday,  Friday,  Saturday,  Sunday | <0500 |
| Unacceptable | <=0 | student, family,  senior,  child | Tuesday | >=0500 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Schedule | | | | | | |
| Use Case | **quantity** | **person** | **day** | **time** | **expected** | **Result** |
| 1. 1 adult, normal, before 5 | 1 | Adult | Friday | 4 | 14.5 | Pass |
| 1. 2 adults, normal, at 5 | 2 | Adult | Monday | 5 | -1 | Pass |
| 1. 1 student, normal, before 5 | 1 | Student | Wednesday | 4 | -1 | Pass |
| 1. 3 adults, normal, before 5 | 3 | Adult | Thursday | 1 | 43.5 | Pass |
| 1. 1 adult, Tuesday, before 5 | 1 | Adult | Tuesday | 4 | -1 | Pass |

# Adult\_After\_5()

Input: int pr\_quantity, string pr\_person, string pr\_day, decimal pr\_time, decimal expectedResult

Output: decimal result

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Equivalence Partitioning & Boundaries | | | | |
| Status | **quantity** | **person** | **day** | **time** |
| Acceptable | >0 | adult | Monday,  Wednesday,  Thursday,  Friday,  Saturday,  Sunday | >=5 |
| Unacceptable | <=0 | Student,  Family,  Senior,  child | Tuesday | <5 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Schedule | | | | | | | |
| Use Case | **quantity** | **person** | **day** | **time** | **expected** | **Result** |
| 1. 1 adult, normal, after 5 | 1 | Adult | Monday | 5.30 | 17.5 | Pass |
| 1. 2 adults, normal, after 5 | 2 | Adult | Wednesday | 5 | 35 | Pass |
| 1. 1 adult, Tuesday, after 5 | 1 | Adult | Tuesday | 8.30 | -1 | Pass |
| 1. 1 adult, normal, before 5 | 1 | Adult | Friday | 3.30 | -1 | Pass |
| 1. 10 adults, normal, after 5 | 10 | Adult | Saturday | 7 | 175 | Pass |

# Adult\_Tuesday()

Input: int pr\_quantity, string pr\_person, string pr\_day, decimal expectedResult

Output: decimal result

|  |  |  |  |
| --- | --- | --- | --- |
| Equivalence Partitioning & Boundaries | | | |
| Status | **quantity** | **person** | **day** |
| Acceptable | >0 | adult | Tuesday |
| Unacceptable | <=0 | Student,  Family,  Senior,  child | Monday,  Wednesday,  Thursday,  Friday,  Saturday,  Sunday |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Schedule | | | | | | |
| Use Case | **quantity** | **person** | **day** | **expected** | **Result** |
| 1. 1 adult, Tuesday | 1 | Adult | Tuesday | 13 | Pass |
| 1. 3 adults, Tuesday | 3 | Adult | Tuesday | 39 | Pass |
| 1. 1 adult, Wednesday | 1 | Adult | Wednesday | -1 | Pass |
| 1. 1 Senior, Tuesday | 1 | Senior | Tuesday | -1 | Pass |
| 1. 0 adult, Tuesday | 0 | Adult | Tuesday | -1 | Pass |

# Child\_Under\_16()

Input: int pr\_quantity, string pr\_person, decimal expectedResult

Output: decimal result

|  |  |  |
| --- | --- | --- |
| Equivalence Partitioning & Boundaries | | |
| Status | **quantity** | **person** |
| Acceptable | >0 | child |
| Unacceptable | <=0 | adult  Student,  Family,  Senior |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Schedule | | | | |
| Use Case | **quantity** | **Person** | **expected** | **Result** |
| 1. 1 child | 1 | Child | 12 | Pass |
| 1. 100 children | 100 | Child | 1200 | Pass |
| 1. 1 adult | 1 | Adult | -1 | Pass |
| 1. 1 student | 1 | Student | -1 | Pass |
| 1. 1 family | 1 | family | -1 | Pass |

# Senior()

Input: int pr\_quantity, string pr\_person, decimal expectedResult

Output: decimal result

|  |  |  |
| --- | --- | --- |
| Equivalence Partitioning & Boundaries | | |
| Status | **quantity** | **person** |
| Acceptable | >0 | Senior |
| Unacceptable | <=0 | Adult  Student,  Family,  Child |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Schedule | | | | |
| Use Case | **quantity** | **person** | **expected** | **Result** |
| 1. 1 senior | 1 | Senior | 12.5 | Pass |
| 1. 50 seniors | 50 | Senior | 625 | Pass |
| 1. 1 adult | 1 | Adult | -1 | Pass |
| 1. 1 student | 1 | Student | -1 | Pass |
| 1. 0 senior | 0 | Senior | -1 | Pass |

# Student()

Input: int pr\_quantity, string pr\_person, decimal expectedResult

Output: decimal result

|  |  |  |
| --- | --- | --- |
| Equivalence Partitioning & Boundaries | | |
| Status | **quantity** | **person** |
| Acceptable | >0 | Student |
| Unacceptable | <=0 | Adult  Family,  Senior,  Child |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Schedule | | | | |
| Use Case | **quantity** | **person** | **expected** | **Result** |
| 1. 1 student | 1 | student | 14 | Pass |
| 1. 25 students | 25 | Student | 350 | Pass |
| 1. 5 adults | 5 | Adult | -1 | Pass |
| 1. 100 children | 100 | Child | -1 | Pass |
| 1. 3 families | 3 | family | -1 | Pass |

# Family\_Pass()

Input: int pr\_quantity\_ticket, int pr\_quantity\_adult, int pr\_quantity\_child, decimal expectedResult

Output: decimal result

|  |  |  |  |
| --- | --- | --- | --- |
| Equivalence Partitioning & Boundaries | | | |
| Status | **Quantity\_ticket** | **Quantity\_adult** | **Quantity\_child** |
| Acceptable | >0 | >0 | >1 |
| Unacceptable | <=0 | <=0 | <2 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Schedule | | | | | |
| Use Case | **Quantity\_ticket** | **Quantity\_adult** | **Quantity\_child** | **expected** | **Result** |
| 1. 1 ticket, 2 adults, 2 children | 1 | 2 | 2 | 46 | Pass |
| 1. 1 ticket, 1 adult, 3 children | 1 | 1 | 3 | 46 | Pass |
| 1. 1 ticket, 1 adult, 1 child | 1 | 1 | 1 | -1 | Pass |
| 1. 1 ticket, 3 adults, 1 child | 1 | 3 | 1 | -1 | Pass |
| 1. 2 tickets, 4 adults, 4 children | 2 | 4 | 4 | -1 | Pass |

# Chick\_Flick\_Thursday()

Input: int pr\_quantity, string pr\_person, string pr\_day, decimal expectedResult

Output: decimal result

|  |  |  |  |
| --- | --- | --- | --- |
| Equivalence Partitioning & Boundaries | | | |
| Status | **quantity** | **person** | **day** |
| Acceptable | >0 | Adult | Thursday |
| Unacceptable | <=0 | Student,  Family,  Senior,  Child | Monday,  Tuesday,  Wednesday,  Friday,  Saturday,  Sunday |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Schedule | | | | | | |
| Use Case | **quantity** | **person** | **day** | **expected** | **Result** |
| 1. 1 adult, Thursday | 1 | Adult | Thursday | 21.5 | Pass |
| 1. 2 adults, Thursday | 2 | Adult | Thursday | 43 | Pass |
| 1. 1 adult, Wednesday | 1 | Adult | Wednesday | -1 | Pass |
| 1. 1 student, Thursday | 1 | Student | Thursday | -1 | Pass |
| 1. 1 child, Thursday | 1 | child | Thursday | -1 | Pass |

# Kids\_Careers()

Input: int pr\_quantity, string pr\_day, bool pr\_holiday, decimal expectedResult

Output: decimal result

|  |  |  |  |
| --- | --- | --- | --- |
| Equivalence Partitioning & Boundaries | | | |
| Status | **quantity** | **day** | **holiday** |
| Acceptable | >0 | Wednesday | False |
| Unacceptable | <=0 | Monday,  Tuesday,  Thursday,  Friday,  Saturday,  Sunday | True |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Schedule | | | | | | |
| Use Case | **quantity** | **day** | **holiday** | **expected** | **Result** |
| 1. 1 carer and 1 child, Wednesday, non-holiday | 1 | Wednesday | False | 12 | Pass |
| 1. 2 carer and 2 child, Wednesday, non-holiday | 2 | Wednesday | False | 24 | Pass |
| 1. 1 carer and 1 child, Sunday, non-holiday | 1 | Sunday | False | -1 | Pass |
| 1. 1 carer and 1 child, Wednesday, holiday | 1 | Wednesday | True | -1 | Pass |
| 1. 0 carer and 0 child, Tuesday, holiday | 0 | Tuesday | True | -1 | Pass |

# Summary

Testing is a big aspect of application development. Unit testing is an essential when building high quality applications. We can use unit testing to run automated testing for very low-level functions. On top of that, automated testing saves a lot of time when there are many test cases. And something came to my mind while building these test cases. If there are hundreds if not thousands of test cases, what would be the solution? Test cases stored in files? Or even databases?