Science and Technology

<u>BSc Computer Science (Systems Engineering)</u>

Software Engineering Management and Development

CST 2550

Group Coursework Report

Middlesex University

Report Layout

The report will delve into the following aspects of the development of the project:

- Introduction
- Design
- Testing
- Conclusion

Introduction

About the project

The project is for a driving lesson booking system. The project has been made using WinForms for the front end and C#.Net version 8.0 with connection to an SQL database for the backend. The project has been developed over the course of 3 sprints, using an Agile methodology, which has nowadays gained significant traction in various sectors, beyond its origins in software development. (**Dong et al, 2024**).

Parts of the SOLID design principles have been used, such as Single Responsibility Principle, which help the team in creating a highly maintainable and scalable system. (**H. Singh et al, 2015**)

Development team

The team that developed the project consists of

- Sanish Ramlal – **Team leader and Scrum master**
- Ridwan Ghamy - **Developer**
- Paul Kamani - **Developer**
- Julien Doolub - **Tester**

Design

The data structure

The data structure selected to store data for this project is the **Hash Table**.

It is a data structure that is stores large amounts of data and provides a constant amortized access time. (**D. Liu et al, 2014**)

This particular data structure was chosen due to its outstanding performance, as even small performance gains compared to other structures can lead to enormous amounts of time and memory saving. For instance, if no collisions happen, a hash table is a flat structure and inserting/deleting/retrieving data could be done in constant time (O(1)). (**van Djik, no date**)

A prime number has been chosen for the size of the hash table as large prime numbers usually produces good separation among input and results in less numbers of collusions (**Bhullar et al, 2016**).

Analysis of key algorithms

The key algorithms used in the program have been analyzed in order to evaluate their time complexity, as shown in the figures below.

```
Method Insert(key, value):      // Avg: O(1), Worst: O(n)
    index ← GetBucketIndex(key)

    if buckets[index] is null:
        buckets[index] ← new linked list

    for each pair in buckets[index]:
        if pair.key == key:
            remove pair from buckets[index]
            break

    append (key, value) to buckets[index]
    size ← size + 1

    if size / capacity > LoadFactor:
        call Resize()
```

The method to add an element to the hash table has an average performance of O(1) and a worst-case scenario performance of O(n), indicating a consistently rapid performance for this operation.

```
Method Delete(key):             // Avg: O(1), Worst: O(n)
    index ← GetBucketIndex(key)

    if buckets[index] is not null:
        for each pair in buckets[index]:
            if pair.key == key:
                remove pair from buckets[index]
                size ← size - 1
                return

    throw "KeyNotFound"
```

The method to remove an element from the hash table has an average performance of O(1) and a worst-case scenario performance of O(n), indicating a consistently rapid performance for this operation.

```
Method Search(key):          // Avg: O(1), Worst: O(n)
  index ← GetBucketIndex(key)

  if buckets[index] is not null:
    for each pair in buckets[index]:
      if pair.key == key:
        return pair.value

  throw "KeyNotFound"
```

The method to search an element in the hash table has an average performance of O(1) and a worst-case scenario performance of O(n), indicating a consistently rapid performance for this operation.

Testing

Testing method used

The various key program functions and algorithms have been tested using **Unit Testing** via MSTest as unit testing provides an effective means to test individual software components for boundary value behavior and ensure that all code has been exercised adequately. (**Ellims et al, 2008**). Some functions have been tested using red-green refactoring.

Table of key test cases

Student functions

Test Name	Description	Inputs (Simulated)	Expected Output	Notes
AddUser	Adds a new student based on user input	Multiple console inputs: Name, Email, DOB, Phone, etc.	Student with email SampleUser@example.com exists in DB	Simulates user input using Console.SetIn
DeleteUser	Deletes a student by email	Input: Fakeuser@example.com	Student no longer exists in DB	
SearchUser	Searches a user and prints data to console	Input: searchme@example.com	Console contains searchme@ex	Output captured using Console.SetOut

			ample.com and Searchyy	
DisplayUser	Displays all users to console	No input	Console contains both seeded emails	Verifies correct DB read/output

Instructor functions

Test Method	Purpose / Description	Input (Console/Input Setup)	Expected Output / Assertion	Notes
AddEntity	Tests adding a new instructor to the DB	Console input for: FirstName, LastName, Email, Password, DOB, Phone, Address	Instructor with email eloise.fox@tutorsden.com should exist in DB	Uses StringReader to simulate user input
DeleteUser	Tests deletion of instructor by email	Console input: quokka@sunnyisle.com	Instructor with that email should be null (deleted)	Tests side-effect of InstructorOperations.DeleteUser()
SearchUser	Tests displaying data of a specific instructor	Console input: quokka@sunnyisle.com	Output should contain quokka@sunnyisle.com and Mighty	Uses StringWriter to capture console output
ListAllInstructorEmail	Tests listing all instructor emails	None	Output should include quokka@sunnyisle.com	Assumes initial data has 1 instructor
ViewInstructorLessons	Tests viewing lesson info for a given instructor email	Input to method: "quokka@sunnyisle.com"	Output should contain learna@trymail.com, quokka@sunnyisle.com, and Manual	Cross-checks instructor, student, and car data across related tables

Car functions

Test Method	Purpose	Input	Expected Output	Notes
AddCar	Test adding a new car to the system	“Toyota”, “Automatic”, “AB12 XYZ”	Car is added with correct values; registration normalized	Tests user input simulation and DB persistence

DeleteCar	Test deletion of a car by registration number	"CD34 EFG"	Car is removed from DB	Checks that car is no longer found in DB after deletion
SearchCar	Test searching for a car by registration number	"EF56 HIJ"	Console output contains car's registration and make	Checks for string presence in simulated console output
DisplayCar	Test display of all cars	N/A (two cars are added)	Console output lists both cars with correct make	Validates multi-car display formatting
TransmissionChecker_ShouldReturnTrueForValidTransmission	Validates transmission format acceptance	"Automatic", "Manual"	Returns true	"Manual" is assumed valid (note spelling vs. "Manual")
TransmissionChecker_ShouldReturnFalseForInvalidTransmission	Rejects invalid transmission strings	"Auto", "manuals"	Returns false	Edge case validation
CarRegistrationChecker_ValidInput_ShouldReturnTrue	Validates correct registration number formats	"AB12 XYZ", "CD34EFG"	Returns true	Tests two acceptable formats
CarRegistrationChecker_InvalidInput_ShouldReturnFalse	Ensures invalid formats are rejected	"1234 AB", "ABCD123", ""	Returns false	Includes empty string as test case
IsUnique_ShouldReturnFalseIfRegistrationExists	Tests uniqueness checker when reg. exists in DB	"XY99 ZZZ" (already in DB)	Returns false	Ensures method checks DB correctly
IsUnique_ShouldReturnTrueIfRegistrationIsNew	Tests uniqueness checker for a new registration	"ZZ88 AAA" (not in DB)	Returns true	Validates check doesn't false-positive
EnterCarReg_ShouldReturnValidInput	Test for correctly reading a valid car registration input	User inputs "AB12 XYZ"	Method returns "AB12 XYZ"	Simulates input/output, confirms correct return value

Lesson functions

Test Method	Purpose	Input	Expected Output	Notes

AddLesson	Tests creation of a new lesson	Student email, Instructor email, Lesson date, Car reg: "AB12 XYZ"	Lesson is added; correct student, instructor, car, and date	Relies on Console.SetIn to simulate user input; verifies DB save and relationships
SearchDate	Tests searching lessons by a specific date	Lesson date: 2025/04/20	Console output includes student and instructor email	Checks if lesson info is printed for that date
Test_EnterDate_ValidDate	Tests valid date input handling	"2025/04/20"	Returns DateOnly(2025, 04, 20)	Direct valid input to date entry function
Test_EnterDate_InvalidThenValidDate	Tests retry mechanism on invalid date input	"wrongdate", "", then "2025/04/20"	Returns DateOnly(2025, 04, 20) after retrying input	Tests methode's resilience to user input error
Test_CheckInstructorEmailExistence_True	Checks if a real instructor email exists in DB	"quokka@sunnyisle.com"	Returns true	Confirms system can recognize a registered instructor email
Test_CheckInstructorEmailExistence_False	Checks if non-existent instructor email is rejected	"fake@sunnyisle.com"	Returns false	Verifies email existence logic works for negative cases

Conclusion

Summary

The Driving Lesson Booking System was successfully developed using C#.NET 8.0 for the backend, integrated with an SQL database. The Agile methodology enabled iterative development and continuous feedback. The Hash Table was chosen as the primary data structure due to its efficient constant-time operations and memory savings, especially with a prime-sized table to minimize collisions. To ensure software reliability and correctness, unit testing was conducted using MSTest, effectively validating core functionalities and boundary behaviours. Overall, the project demonstrates a robust, efficient, and scalable solution underpinned by sound software engineering principles.

Limitations and reflection

- **Time:** Time was one of the main limiting factors for this project. This was mainly due to the fact that the team was working on another coursework in parallel which was given higher priority as it had an earlier deadline. This caused the team to give little focus on this coursework. The problem was further aggravated by the fact that the deadline was preponed by 4 days.

- **Knowledge:** Another limitation of this coursework is that initially the team had very little knowledge of the tools to use in the project. This has caused progress to come at a slower pace as the team had to both learn about the tools and implement them at the same time.
- **No Real Users for Feedback:** Given that the team is comprised of members which are all well versed in programming, when testing, the feedback is given more from a developer perspective, rather than an end-user perspective, possibly leading to the creation of unintuitive or user-unfriendly interfaces.
- **Vague/Unclear requirements:** Some of the requirements for this coursework were quite unclear. This caused the team to lack a clear direction in which to develop the project and caused the team to rely to much on the lab tutor's feedback to make changes.

Changes in approach for the future

- Plan in advance how much time the project and each task is going to take in order to better manage time.
- If there are other projects or coursework running in parallel and with similar deadlines, focus on each one in the same way, instead of heavily prioritizing one or the other.
- Research in advance and in great detail about the tools needed for the project. This will reduce the amount of time spent on research during the coursework.
- Have one or more normal persons test the program in order to get feedback from an end-user perspective.
- Try to understand the requirements fully and ask questions to the lab tutor when in doubt from the very beginning instead of during the project.

References

- D. Liu, Z. Cui, S. Xu and H. Liu, "**An empirical study on the performance of hash table,**" *2014 IEEE/ACIS 13th International Conference on Computer and Information Science (ICIS)*, Taiyuan, China, 2014, pp. 477-484, doi: 10.1109/ICIS.2014.6912180. (2014) *Available at: https://ieeexplore.ieee.org/document/6912180*
- Dong, H., Dacre, N., Baxter, D., & Ceylan, S. (2024). **What is Agile Project Management? Developing a New Definition Following a Systematic Literature Review. Project Management Journal, 55(6), 668-688**. (2024) *Available at: https://journals.sagepub.com/doi/full/10.1177/87569728241254095*
- Ellims, M., Bridges, J. & Ince, D.C. **The Economics of Unit Testing.** ***Empir Software Eng*** 11, 5–31 (2006). *Available at: https://link.springer.com/article/10.1007/s10664-006-5964-9*
- H. Singh, S.Hassan, "**Effect of SOLID Design Principles on Quality of Software: An Empirical Assessment**" *International Journal of Scientific & Engineering Research, Volume 6, Issue 4, April-2015 1321 ISSN 2229-5518* (2015) *Available at: https://d1wqtxts1xzle7.cloudfront.net/51951126/Effect_of_SOLID_design_principles_on_quality_of_software_An_empirical_assessment-libre.pdf?1488192514=&response-content-disposition=inline%3B+filename%3DEffect_of_SOLID_design_principles_on_qua.pdf&Expires=1744746243&Signature=Byj2qNwqFoAJ0t6HUtTwrw2MfUz6t3Kiwk9wHu-cP1LixhieeR17Zx0iH9U6bWURu2qwtdAjH7HCBbMzvVKSRUWVGSlh1NMvjwph4tImTfO-yW9mjibZ29fMdHheTcckWjLqY1KAEBTji3i1OWHF6hnVDpW-LNa9HsnJ0xHhd-AjjZpKEccIrFslt8L0xaoFCllj7AVZm9VHd~OInBixb0cAmYncNhFda3WYn5qbnwr6A0PxXF4owtzq8vR2tp6bIU8WEhIX-jPVz2YPk37LLchLE9LPHnKOfvd7wMa~3t-1kWHMmCL75PbZtmIAtXzmSc76EbWkIrTYEOjy6eIGNw__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA*
- R. K. Bhullar, L. Pawar, V. Kumar and Anjali, "**A novel prime numbers based hashing technique for minimizing collisions**," *2016 2nd International Conference on Next Generation Computing Technologies (NGCT)*, Dehradun, India, 2016, pp. 522-527, doi: 10.1109/NGCT.2016.7877471 (2016) *Available at: https://ieeexplore.ieee.org/abstract/document/7877471*
- T. van Djik **"Analysing and Improving Hash Table Performance"** (No date) *Available at: https://www.tvandijk.nl/pdf/bscthesis.pdf*