ICFS2 Summative 2 Write Up

Section 1: Testing

Section 1.1: Testing strategy and methodology

Testing is key to be able to guarantee the functionality and performance of a program. When testing, it's important to perform these tests from two different perspectives, the first being the most important, that of a user, but the second being from the eyes of a developer. When testing from a user's angle, they are only ever interested in the functionality of the program and whether or not it fulfils their purpose, regardless of the underlying systems and code. This is more commonly known as black box testing, where the working parts are abstracted, and only the end result is important. On the contrary, developers will initialise tests for their code, rather than the whole functionality. This is known as white box testing, and an emphasis is made on code quality and testing individual components. Many languages have dedicated libraries or systems to perform these tests, and Python has a library called pytest, which enables automated testing. Automated testing refers to a set of tests that have predefined outputs, and can be run with ease and little time. Automated testing, is often integrated into the continuous development (CD) pipeline where each time a change is made to the program, the tests are rerun to ensure that existing functionality has not been broken in the process of development.

Whilst testing is considered to be the final stage of development, it can also be used to help a developer achieve all their requirements. This is known as test-driven development (TDD) where the program has a set of clear requirements, and from these requirements, a series of tests can be created, and the programmer works through the tests one by one, adding the functionality to the program and testing it once they've written the relevant code. The objective doesn't lie within writing all the code at once and conclusively testing, but rather an iterative process that produces high quality, minimal code that only achieves a passing outcome for an individual test before moving onto the next test. Although it wasn't a religiously followed TDD process, my development pathway resulted in the development of a supporting file called test_classes.py which was contributed to, whilst the construction of the main application. For me, it was a combined effort of understanding my first requirement, writing the code for it, then writing some code to test the success of the requirement. As such, in parallel, I created a series of tests using print statements to see the outcome, whilst remaining within the constraints of the requirements.

Following the whole development of the application, I moved onto more formalised testing, where I performed some white box testing through manually testing components on the front end such as buttons, filters and drop downs. For this, I moved through the application, created some tests with a given purpose, the identified what the expected outcome should be. After creating all the relevant tests, it was a process of executing these tests and recording my results, and ultimately deciding whether the test was passed or failed and the findings are displayed in the table within section 1.2.1. The next step was to test the underlying systems and the synergy between files and functions, to check if the created classes were functional. For this I went through all the functions that the classes provided and created unit tests to test each functionality. These tests can then be run using pytest, which returns a summary of all the passed/failed tests. The results can be found in section 1.2.2. Finally, I performed some alpha testing in the form of family, who worked through the program mimicking that of an actual user.

Alpha testing refers to an internal user testing the program to identify any issues that may arise. The reason behind alpha testing is that from a developer's perspective, it's easy to delude ourselves to think that everything works as it should, and that we know what the correct input is to get the correct output. By carrying out 4-eyes testing it means that another person who may think in a different way can identify different ways to interact with the systems, resulting in different outputs.

Section 1.2: Outcomes of application testing

Section 1.2.1: Outcome of manual tests

Test Number	Description/Purpose	Expected Result	Actual Result	Pass /Fail
1	Logging into the system with valid manager username/password	System authenticates user	User is authenticated	Pass
2	Logging in with incorrect credentials	System denies access	User is not authenticated and error message shows 'Invalid username or password'	Pass
3	Filtering the team's success rate by time period	No data for last 30 days but 3 calls in last 90 days, and all are successful	Filter works as intended and last 90 days shows a fully green circle with 3 calls. Other time periods except all time show that there is no available data	Pass
4	Performance Highlights correctly identifies top performers based on calls taken and success rate	Top performers are Logan, Ava and John. Worst performers are Jane, John and Ava. (Since there are not too many employees, John and Ava appear in both lists)	Correctly identifies top and worst performers.	Pass
5	View staff details, using the dropdown to view staff details for all staff members.	Data below should change to reflect each staff's details.	Outputted data matches excel data.	Pass
6	Add new staff member	New staff member entry should be found in csv file.	Streamlit shows a successful messaging saying that the new user has been added. Navigating to the csv file shows that the new staff has been added, however there is an opportunity for two new improvements. 1) Auto filling the staff ID with the next free unused number to prevent duplicates 2) Being able to see under view members the details of the newest member	Pass for intended purpose but room for improvement

7	Edit staff details	Change Jane Smith ID 102, target successful calls 10 -> 9	Success message flashes showing that details have been updated. CSV file shows update.	Pass
8	Remove staff	Remove 101 – John Doe, so record should not exist in CSV	Success messages pops up and CSV file has been updated	Pass
9	Log into staff account with valid credentials	User is authenticated and directed to dashboard	User is successfully authenticated and redirected	Pass
10	Accessing quick links at the top of the page for frequently used resources	Section should unfold to show 3 different link, all of which redirect to google.com for testing purposes.	Able to unfold section and click on all links	Pass
11	Performance metrics correctly shows pie chart with call success rate and table below shows all taken calls	See a pie chart with 50/50 split for user 102, since they only have 1 successful and 1 failed call. Line graph should show a comparison between team average and user average.	Whilst the pie chart is correctly shown, the line chart does not show the user's score even though it's evident in the legend.	Fail
12	Start workday simulating a user clocking in	Clicking the button should show a success message and have an option to simulate an incoming call for a dev environment like this	Correctly shows message and option to simulate incoming call	Pass
13	Simulate incoming call	Clicking on the button should start a dummy call and show the ID of the call in progress.	There is a message saying that a call is in progress and call_details.csv file has been updated with the record.	Pass
13	End workday	Clicking the button should end the user's workday and their status should be changed to Out of Office.	Success message shows time worked for, and time_elapsed shows how long the employee worked for, but the status has not been updated to Out of Office.	Fail
14	Log out button	Clicking the button should log out the current user and there shouldn't be any details left in the entry boxes.	User is successfully logged out and the credentials are not saved.	Pass

Section 1.2.2: Outcome of unit tests

```
============================= test session starts =============================
platform win32 -- Python 3.12.6, pytest-8.4.1, pluggy-1.6.0
rootdir: C:\Users\uarju\OneDrive\Documents\Uni Work\IFCS2\Summative_1
plugins: anyio-3.7.1, langsmith-0.3.45
collected 11 items

pytest_classes.py ...........

============================== 11 passed in 0.05s ==============================
```

```
== test session starts ==========================================================================================================

                                                                                                                        [100%]

=== 11 passed in 0.05s ==========================================================================================================
```

Used two screenshots for readability

Section 2: Learning

Whilst the previous course took a stance to teach you the fundamentals of writing good code, this sequel has enabled me to be able to write purposeful code stemming from the requirements.

The course began with Object-Oriented Programming, which is a skill that I already had an understanding of, but the course gave me a formal opportunity to learn how and why it works. I implemented the knowledge during Summative 1, where I created classes to be used during my EMS MVP.

Then the next section was an introduction to GUIs, which I have also had a lot of experience with in the past, where my first encounter with the tool led to the creation of a basketball scoring system. As such, portions of the content felt more like a recap of how to use the tool than any new knowledge.

Moving onto the next segment, unit testing was a skill that I taught myself, but didn't have any formal knowledge about it and this section gave me the theoretical perspective of the existing processes that I was already using, such as the Continuous Integration (CI) pipeline, which we met in the last course too, but through GitHub workflows and automated testing. Something new that I learnt in this portion of the course was regarding smoke tests, as it has never been something that I've used, as I've always started my tests under the assumption that the tests are working, however this promotes the robustness of the testing code's quality. As such, I have also implemented this new piece of learning into the unit tests above.

Whilst I have worked with visualisations in plotly and matplotlib, using seaborn was a new learning curve for me alongside using matplotlib visualisations within tkinter, which isn't something that I've experimented with before. Personally, the most interesting segment of this course was using NetworkX as it's a tool that I've touched on, but never really found the need for, nor the understanding to fuel any creativity with this library. However, combining it with Social Network Analysis (SNA) showed a whole variety of reasons to use NetworkX. Using this

introduction as a stepping stone, there's a lot more to learn about NetworkX and its domain of functionality and will be experimented with in my own time. As with many libraries in Python, hands on experience with the product will give me some more knowledge and motivation to understand the theory behind it, since on the face of it, graph theory and other mathematical structures can be hard to practically envision, but using tools like NetworkX will enable me to understand the purpose of these structures.

The final weeks of this course prioritised the production of large scale software, through the software development lifecycle (SDLC) and associated documentation, the least enjoyable portion of programming. The various SDLCs were taught during the A Level specification, so it wasn't unfamiliar territory for me but in most instances, like my own, it's rare to see the employment of a single SDLC that's religiously followed but rather certain characteristics picked and mixed to create a hybrid approach. Certain approaches will be better suited for certain projects based on the initial requirements, changeability of those requirements, response rate of customers amongst other factors.

As an AI-driven team who focus on operational efficiency, we originally started with an agile methodology but found that due to the slow response rates of business users who were our clients, it was hard to complete stories within the dedicated 4 week sprint timing and many projects and stories within those epics would roll over into the next. Now we have employed a Kanban board scrum approach, where we don't have fixed deadlines, and works flexibly to accommodate for the business' changing requirements and cooperation. Even documentation, while theoretically has a predetermined purpose and structure, it's rare to stick to a given template since it can differ from project to project, and as a team, we have different levels of detail and structure in our documentation, and are exploring methods to make it more uniform, and contain all the necessary data.

Comparing to my relatively short career, there are a lot of distinctions between what's taught as theory and what's used in practice. In one way, this can be seen as cutting corners, in another it can be seen as saving time, such as major documentation pieces, are often delegated to business analysts within the team, or they will perform the requirements piece alongside developers or even alone, the expertise they bring to the table about understanding the requirements and understanding what's technically feasible allows them to act as a bridge between the users and the developers and ensure that both parties have the same end goal. In teams dedicated to developing applications for end users such as the public to benefit from, there are often separated teams dedicated to each of the described functions when it comes to developing a large scale app. For instance, there is often a designer who focuses on UI/UX designing, a business analyst or someone who plays the role of translator, developers and even testers. Whilst theoretically, it's important to understand the stages that occur during the development of an app, for a day-to-day basis, each person is only concerned with their role in that wider team and will have sub-processes that fulfil their stage of the development. In cases like this, documentation becomes increasingly important since the application will move between many pairs of hands, and all must work from the same set of knowledge to get a consistent output that matches everyone's expectations.

Overall, I have learnt a lot from this course, and the benefit of an apprenticeship begins to shine as the content that we cover becomes increasingly apparent in our day-to-day lives and gives us opportunities in which to develop, for example, as a team, it's evident that we don't do enough testing of the programs we produce, leading to bugs later down the line, and this creates a chance for knowledge such as automated testing to be implemented. In fact, following last module's

introduction to CI/CD pipelines, I led a workshop regarding my learning, offering insights to others in the team on how this could be beneficial to them.