APPRENTICESHIP PORTFOLIO

Arron Dunne

Table of Contents

Introduction 2

Project 1 – Machine Learning Test Input Generator 2

- Project Summary 2
- Case Study 3
- Requirements 4
- Design - Analysis 4
- Design - Architecture 9
- Logic 10
- Problem Solving 14
- Source Control – Interpret and Follow 15
- Business Environment 17
- Agile 17
- Project Conclusion 18

Project 2 – RBLS 18

- Project Summary 18
- Analysis 18
- Requirements 19
- Design 20
- Waterfall 20
- Requirements Standards - Interpret & Follow 20
- Business Environment 21
- Project Conclusion 21

Project 3 – Database 22

- Project Summary 22
- Data 22
- Logic 23
- Test 27
- Project Conclusion 28

Project 4 – Dashboard 28

- Project Summary 28
- User Interface 29
- Deployment 31
- Maths 32
- Project Conclusion 32

Business Environment 32

- Security 32
- Health & Safety 33
- General Data Protection Regulation 33

Professional Development 34

Introduction

I joined Capgemini Engineering (previously Altran UK, referred to as the company) on the Software Engineering Apprenticeship Scheme in September 2020. During my time on the course, I have worked on a variety of projects in a range of different roles. My experience has taught me important competencies required for Software Engineering and how to use them in a business environment. In this portfolio I will discuss the projects I worked on, the responsibilities I had on them, the skills I learnt from each one and how I applied these skills to my work.

I am based in the Bath office of Capgemini Engineering, which is the High Integrity Engineering Centre (HIEC). The HIEC specialises in safety critical software, such as Air Traffic Control tools, Railway Safety Systems and Military Defence Equipment software. Because of the risks involved with these applications, safety is given the highest priority and the tools and processes used within the HIEC aim to ensure software solutions are safe and reliable.

When I joined the Apprenticeship Scheme, I had recently graduated from the University of Bath with a MEng degree is Aerospace Engineering. I was able to bring a lot of skills from university to my new role as a software engineer, including problem solving, mathematics and design methods. This apprenticeship helped me to improve these skills as well as learn new ones.

Project 1 – Machine Learning Test Input Generator

Project Summary

My first project on my Apprenticeship was on the HICLASS research project. HICLASS is a large research and development project across multiple companies, aimed at improving software methodologies, tools, and techniques for the aerospace sector in order to reduce costs and improve performance. I was part of a small team of 3-4 people focussed on methods to improve the performance and efficiency of software testing.

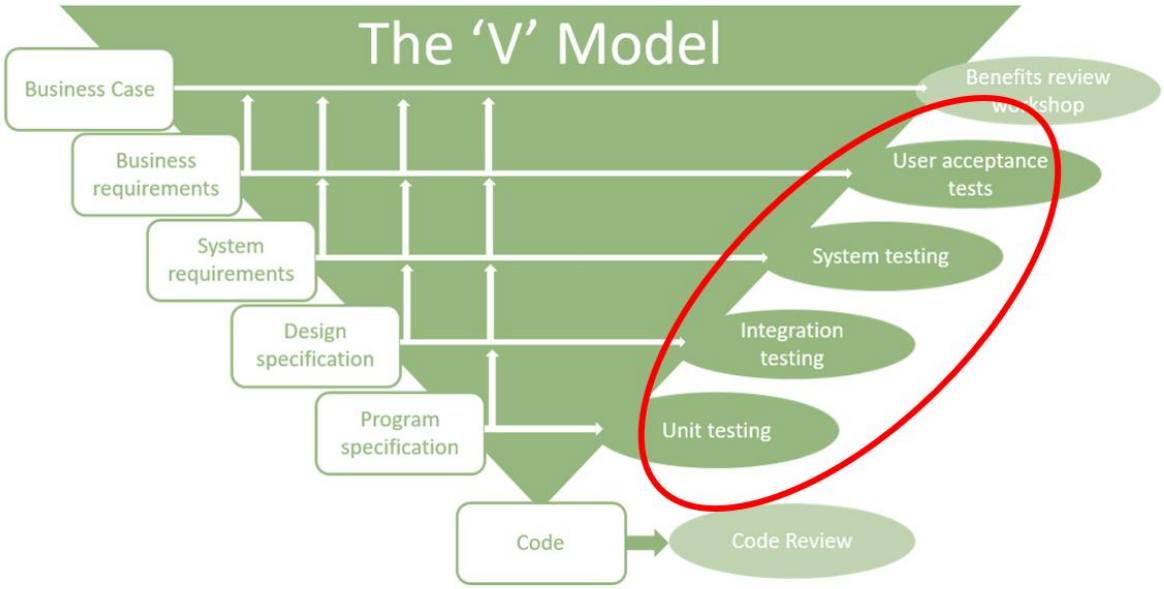


Figure 1: The V-Model of Software Development (Testing shown in red bubble)

Software testing is one of costliest and most time-consuming stages in the Software Development Lifecycle, especially in the safety critical industry. Systems being deployed in this sector need to be extensively tested to ensure safety targets are achieved. There are many phases of testing which each

corollate to a different phase of requirements and development, as shown in the V-Model (Figure 1). The different phases on the right-hand side include unit testing, integration testing and system testing. System testing was the focus of this project because it is normally the most complex testing required as all components of the system are tested together and it takes the longest to complete as a result. Reducing the time taken and cost of system testing was the objective given by the stakeholders of the project. These stakeholders were members of the HICLASS project and the Team Leader of the HICLASS team at the company, Thomas Wilson.

To make an improvement to system testing, I first had to understand the current situation. The system testing methods used at the HIEC mainly fell into 2 categories:

- **Manual Testing:** Tests created by Testing Engineers with knowledge of the system. This allows specific functions/scripts and code segments of interest to be targeted during testing. However, this is a slow process and requires a lot of time from the Test Engineer, as well as the domain knowledge. Therefore, it is costly.
- **Random Testing:** Tests generated automatically using scripts and a defined input space. This allows a lot of tests to be quickly created and only requires human input to create the test input generation process, so little domain knowledge is needed. On the other hand, this method is very poor at covering the entire system and will rarely generate tests that cover interesting areas of code with many specific prerequisites, which are often the most useful tests.

My task was to research and prototype an automatic test input generation tool, which used cutting edge Machine Learning (ML) technologies. This was split into two deliverables:

1. **Case Study** outlining the feasibility of this tool, based on existing technologies in the aerospace or similar industries, and
2. **Prototype Tool** with some experiments on software testing using it, to compare with other testing methods.

The aim was to analyse the feasibility, performance and difficulties of an approach and assess whether it could be used in future projects to reduce testing costs. Theoretically, this approach could combine the speed and automation of random testing with the complexity of manual testing. The machine learning could identify interesting and valuable tests without the need for a Test Engineers time, which would save time and money.

Case Study

For the case study, I had to produce a report which analysed machine learning technologies used in the software engineering industry. I needed to assess which technologies could be potentially used for the prototype tool I was developing, as well as learn what the cutting edge was for machine learning in the technology industry. I had a timescale of 2 months to complete this work and produce the final report.

I found other case studies which had been done on similar areas of machine learning and used these as a starting point. They identified some of the most widely used machine learning technologies; neural networks, evolutionary algorithms (EAs), fuzzing and generative adversarial networks (GANs). I researched these technologies further by looking through the references of these reports and searching for online material about these technologies. I completed some training and tutorials on them as well to get an idea of how intuitive they were to get started with and use. This also helped me assess how promising they were for use in software testing.

I concluded that the most promising technology for our application were evolutionary algorithms, and that they would be a feasible technique for software testing, and could potentially improve the efficiency and reduce the cost of software testing at the company. This is because they do not rely on training data (the output data for a given set of inputs on a system which is used to improve the machine learning algorithms), which neural networks need. They are also one of the most intuitive concepts and therefore engineers could quickly understand how they work and make improvements to them if desired. EAs are also a widely used technique and so there is a lot of frameworks, research, and documentation around them. They have also been tailored for many different applications, including software testing.

I completed the case study and submitted the report to the HICLASS project board within the deadline. I received positive feedback from other external members of HICLASS who agreed with the conclusion I had made that EAs were the best suited machine learning approach to test input generation and was a feasible technology to develop further.

Requirements

The second phase of the project was to design and develop a prototype machine learning test input generator and evaluate it to see if it was a feasible tool to deploy into Capgemini's software testing toolset. I began by writing out requirements that the tool would need to achieve to be a valuable method of software testing. In order to identify what these requirements should be, I spoke with other Software Engineers who had a mixture of experience with software testing. This helped me identify the important characteristics that are needed for a software testing tool, as well as areas where the ML tool could be an improvement on the existing methods. The requirements were split into two sections:

- **ML Core Requirements:** The requirements the machine learning algorithm must achieve in order to generate tests which were an improvement over current methods
- **Interface Requirements:** The requirements that the tool must achieve in order to be able to properly integrate with the rest of the architecture and automatically generate and execute tests on a system under test (SUT)

I captured these requires in a table and assigned them a priority of 'Low=1', 'Medium=2' or 'High=3'. A priority of 'High' was a requirement that was fundamental to the success of the tool. A priority of 'Medium' was a requirement that would improve the performance of the tool considerably but was not necessary for its operation. And a priority of 'Low' was something the tool did not specifically need to operate well but would be a nice feature to have and offered small improvements. The requirements for the ML Core are shown in Table 1. My Technical Lead reviewed the requirements I had specified and agreed that they were correct.

Design - Analysis

Once the requirements stage had been complete. I began planning the design and development of the tool. As mentioned before, there are a range of tools and frameworks for EAs that have been made by others. This meant that I had three different options for developing my prototype.

- Off the shelf Tool – A pre-existing Test Input Generator tool for software testing
- Machine Learning framework – A library of Machine Learning functions which could be adapted to make a Test Input Generator tool
- Completely custom-made tool – No external tools or libraries used

The off-the-shelf tool would be the most cost effective as it would not require much work to get it working. On the other hand, building my own tool from scratch would take the most time and therefore be the costliest. Based on this, I first completed an evaluation of the off-the-shelf tools. I found lots of automated software testing tools that use machine learning to some degree, however I needed to evaluate whether they could be used for our application. There were key features that many of the tools failed to meet and could therefore be quickly eliminated. I discarded all tools that were:

- Not using machine learning
- Not applicable for system testing
- Not open source

Using machine-learning and system testing was a necessity for the project and was the purpose of this research, therefore anything not complying would not be useable. The requirement for an open-source tool was necessary because of the cost constraints of the business and project. Also, open-source software allows different users to modify the application to better suit their needs, and users are not locked into using the tool only how the developers intended it. This evaluation left me with three potential candidates for off-the-shelf tools: Evomaster, AFL and Sapienz.

These three tools used machine learning for software testing and had the code publicly available. They therefore needed a more thorough examination to determine if they should be used for the prototype. They were assessed by scoring each tool against the ML Core requirements and comparing these scores. This evaluation was done with the technical lead, Thomas Wilson, and another member of the team, David Good. For each requirement the tools were analysed specifically for that aspect and scored, with some rationale. A score of 0 was given where the tool failed to meet the requirement. A score of 1 was given where the tool partially fulfilled the requirement. And a score of 2 was given where the tool fully fulfilled the requirement.

Table 1: ML Core Requirements and Off-the-shelf tools evaluation

Requirement	Priority	AFL	Evomaster	Sapienz	Notes
The EA tool shall be adaptable to different sources of feedback or an appropriate algorithm should be selected based upon available feedback	3	0	0	0	**All** - they are tailored for their specific objective; modification would be needed to meet this requirement.
The EA tool shall be able to optimise for 1 or more objectives	3	1	1	2	**AFL** - a modified version of branch coverage (there is test case minimisation but this is a separate function) **Evomaster** - maximises code coverage (or API endpoints coverage for black box) and fault detection **Sapienz** - Maximising fault detection & coverage, minimising test case length

Requirement	Priority	AFL	Evomaster	Sapienz	Notes
The EA tool shall produce data that covers input space defined by constraints	3	1	2	1	**AFL** - If left for long enough, should eventually cover all the input space **Evomaster** - Covers all endpoints of an OpenAPI schema **Sapienz** - Only Android interactions permitted, doesn't optimise for input space coverage
The EA tool shall be able to develop tests that cover the input space without any system coverage feedback	3	0	2	0	**AFL** – Seems to rely on system coverage feedback **Evomaster** - Covers all endpoints if an OpenAPI schema is provided **Sapienz** - Doesn't appear to support this
The EA tool shall be able to generate tests that efficiently reach test coverage criteria	3	1	2	2	All tools seem to do this (AFL relies on random fuzzing however so is less efficient).
The EA tool shall be agnostic to source of tests being evaluated (i.e. inputs to population fitness etc. should be valid from either automatically generated tests and from other sources)	3	2	0	1	**AFL** - Uses an initial seed of test cases **Evomaster** - Doesn't appear to allow user-defined tests **Sapienz** - Motif core captures interesting test sequences, but otherwise no support for user-defined tests
The inputs to the EA tool shall be known and effort to manage predictable	3	?	?	?	We would need to evaluate this based on experience using the tools because there is not enough in literature to reliably assess it.
The methods of test generation shall be resilient to changes in the test artefacts (ICD's, spec, SUT)	3	1	0	?	**AFL** - May be able to use previous tests **Evomaster** - Previous test cannot be used on a new interface

Requirement	Priority	AFL	Evomaster	Sapienz	Notes
The EA tool shall not require information about the implementation.	3	1	2	1	**AFL** - on-the-fly instrumentation of black-box binaries. **Evomaster** - Black-box using the OpenAPI schema **Sapienz** - Black-box uses a non-invasive activity level 'skin-coverage' metric
The EA tool shall be able to use presence of faults to avoid generating tests that hit existing faults.	3	0	0	0	**AFL** - Random fuzzing doesn't take this into account **Evomaster** - Failing tests: the tests generated should all pass, and not fail, even when they detect a fault. In those cases, comments/test-names would point out that a test is revealing a possible fault, while still passing. **Sapienz** - Only focussed on coverage and not avoiding faults
The EA tool shall accept constraints	3	?	1	1	**Evomaster** - Constraints are provided through the OpenAPI schema **Sapienz** - Constraints are provided through the Android package
The EA tool shall be able to quickly learn from existing test data	3	1	0	1	**AFL** - initial test cases are required at the start **Sapienz** - Motif genes are used to capture interest events which unlock coverage
The EA tool objective shall be editable by an external entity	3	0	0	0	**All** - All tools have a fixed objective.

Requirement	Priority	AFL	Evomaster	Sapienz	Notes
The EA tool should be able to provide a minimum length test for a given fault.	2	2	0	2	**AFL** - Test case minimisation is provided as a separate function **Sapienz** - Test case minimisation is part of the GA
The EA tool should be able to generate a regression test suite (i.e. create a suite of tests that has examples of coverage where there were/are faults)	2	?	2	?	**Evomaster** - Self-contained tests: the generated tests do start/stop the application, binding to an ephemeral port. This means that the generated tests can be used for regression testing
The EA tool may be able to identify where gaps in coverage are	1	1	1	1	**All** - coverage (branch/code etc) is used as a metric but is not identifying gaps
The EA tool may be able to save state such that it can be restored to point in time	1	2	0	?	**AFL** - All test cases can be saved in the queue and accessed **Evomaster** - Test cases are saved at the end only
The EA tool may be able to merge "understanding" with data from a different run of the tool. Ideally should be parallelisable.	1	2	?	?	**AFL** - multiple runs can be done in parallel, with the information pooled. Information is not shared between parallel runs however.
Weighted minimum score		36%	38%	35%	
Weighted maximum score		42%	41%	48%	

Based on this evaluation, Evomaster and AFL are poor candidates for the full requirements. Sapienz has a large uncertainty with the highest maximum score. However, the score is still low for the needs and therefore was also not a good candidate. I therefore concluded that none of the off-the-shelf tools were fit for purpose.

Next, I moved onto looking at machine learning frameworks. Frameworks offer methods, classes, and scripts for machine learning, without packaging them up in a specific application. This meant that that were more customisable compared with off the shelf tools, but at the cost of a need for engineers to integrate the different parts properly and fill in any of the missing aspect which the framework didn't provide. There are a wide range of possible frameworks, some of the most promising were:

- DEAP
- TensorFlow

- PyTorch

As part of the research into using machine learning for software testing, I concluded that an evolutionary algorithm would be the most effective to use. This is because it uses a trial-and-error type approach with feedback from running tests on the system to improve itself. The benefit is that no prior knowledge of the system, other than the input space, is needed to set the EA up. I decided to use DEAP because it was written in Python, open source, well supported and documented, and specialised in EAs. I chose to use Python because it is a company recommended language and is used extensively in the machine learning field. The companies recommended IDE for Python is PyCharm, so this is what I used for development.

Design - Architecture

Using my research from the case study on machine learning and the requirements, I created the architecture of the EA. The architecture was split into two aspects, the machine learning architecture, and the interface architecture, similarly to how the requirements were split.

I designed the overall tool architecture so that it could interface with any external systems. This architecture had been used for other software testing applications, and the main the difference was having a EA core instead of other methods of testing. The test harness is a technique that my colleagues had experience working with.

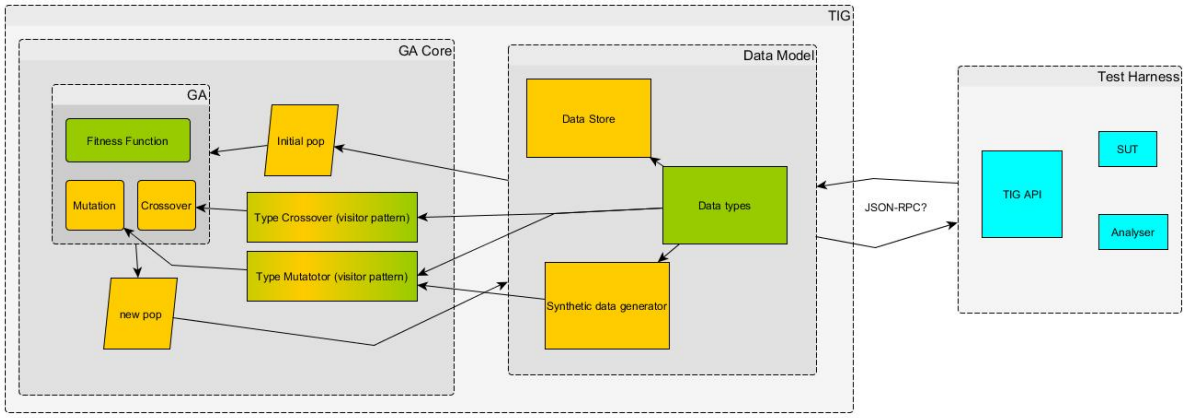


Figure 2: Overall tool architecture

Next, I designed the ML architecture. As concluded earlier, an EA was selected for this task so the architecture was based on EA architectures researched as part of the case study. The architecture is shown in Figure 3. The EA used an iterative process to optimise the software tests. Data is shown in circles, and functions are shown in rectangles. Once the configuration had been set, which is the first step, the main iterative EA loop begins: evaluate fitness using fitness function, select next population, modify individuals (cross-over and mutation) and evaluate fitness of new individuals. This loop continues until a pre-determined stopping criteria had been met.

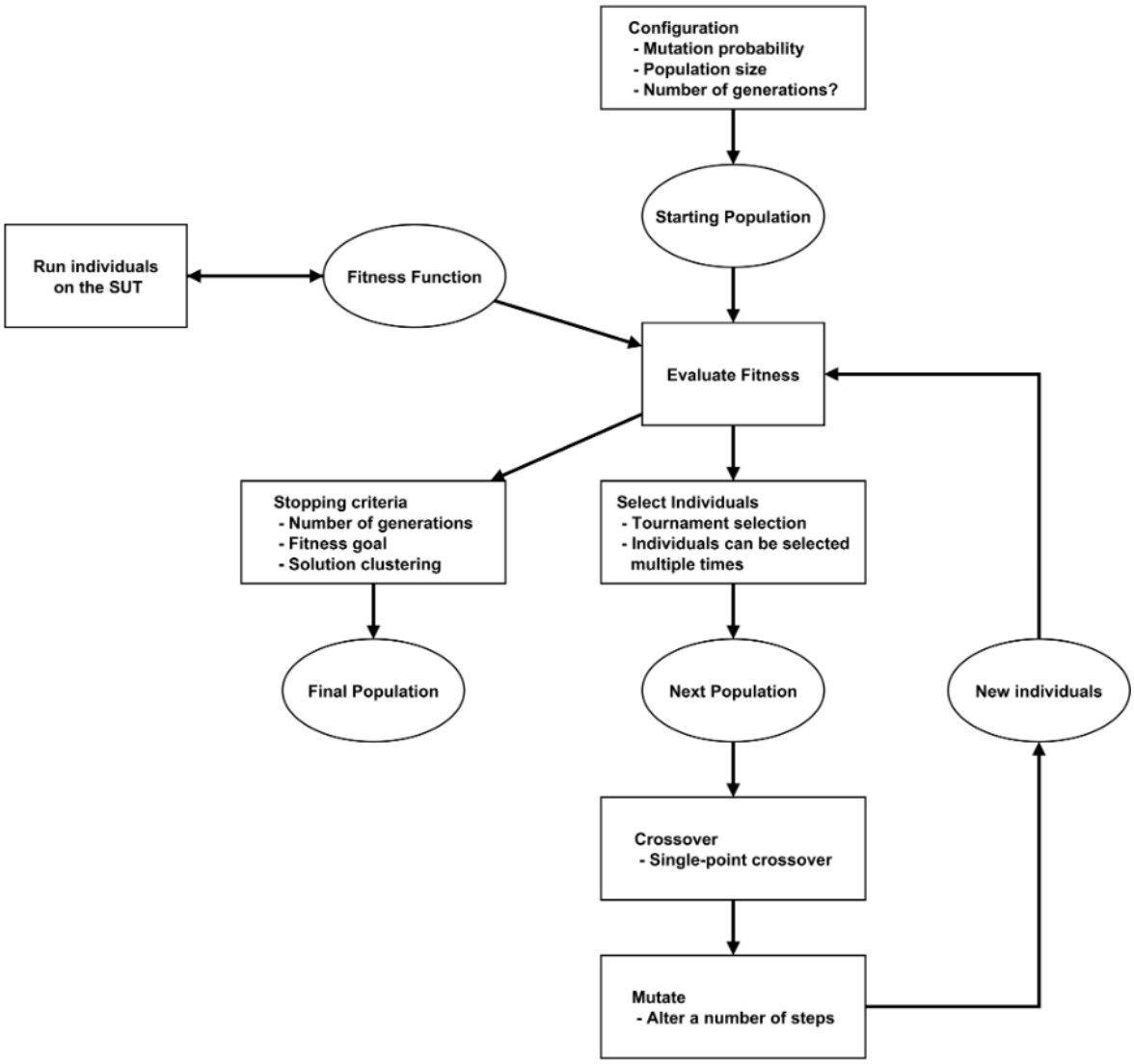


Figure 3: Architecture for Machine Learning Test Input generation tool

Logic

My next task was to create the Machine Learning core using the DEAP framework. Because the main project aim was to establish the feasibility of a Machine Learning tool for Test Input Generation, a representative example of Test Input Generation was used to spare time creating a test harness to a real-life system, which would have taken considerable time and not contributed to the evaluation of the tool itself. Therefore, I created a 2D 'Turtle-Game' to evaluate the tool. The 'Turtle-Game' contained a grid of stationary boxes, shown to the user. The turtle was a point that began in the middle and could move around the area using rotation and forwards steps. The boxes were analogous to verification conditions in a system, and the turtle represented the test case, which executed particular areas of the system. The visual feedback to the user was very insightful and allowed the user to quickly determine how the tool was performing. The turtle, being the test cases, was controlled by the evolutionary algorithm. Figure 4 shows a single turtle with multiple steps and the red boxes.

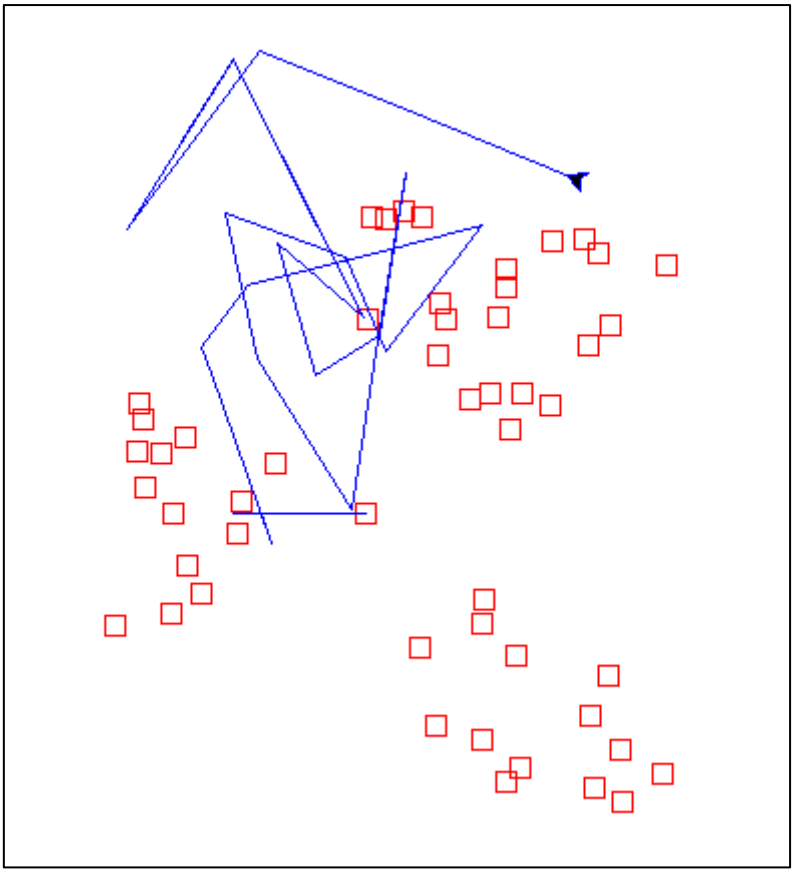


Figure 4: Turtle Game

The main functions of the Evolutionary Algorithm were generation, evaluation, and reproduction (also called mutation and crossover). DEAP provided functions for modification, selection, and the overall EA iterative loop. The most notable missing component was the fitness function, and the population generation processes.

The fitness function in an EA is used to evaluate how good an individual is in relation to some pre-defined objectives. A score is assigned to each individual in the evaluation stage, and this score is then used in determining the likelihood of an individual remaining in the population or being discarded. The overall EA goal is to produce individuals with greater and greater fitness values as time goes on, by keeping higher fitness value individuals in the population, modifying them in the hopes of increases their fitness even further, and removing individuals with a lower fitness value.

The fitness function was a function that was called for each individual once per generation. To determine the fitness, the induvial was run on the system, and the metrics which were returned were analysed in order to give a score. DEAP handled the calling of the fitness function, but I had to write the contents of this function, shown in Figure 5.

In this fitness function, the fitness score was based on the desired characteristics or metrics of a test case. That is, the number of box-to-box connections a test makes in its run. This represents a test case executing code in one areas of the system space, and then in another area afterwards. Test cases which move through the system are deemed better test cases and more likely to find faults, so the higher the number of box-to-box connections, the better the test case and therefore the higher its fitness score should be.

As shown in Figure 5, the function is declared and appropriately named **`fitness`** (line 13). Then I added a multi-line comment to provide information of what the function is doing (line 14). This means that other developers can use this function and understand how it operates in the scope of the whole tool.

At the beginning, a timer is started using the python in-build time function **`time.time()`**, which returns the current system time and assigning this value to the variable **`start`** (line 25). The time object was imported into the script (line 10). This is not a part of the fitness value. Instead, it is a metric I used to compare the time taken for different fitness functions to run. This meant I could analyse the speeds of different versions of the fitness functions and could analyse and improve the efficiency.

Inside the fitness function, the test case is executed on the system (line 31). This is done using the **`run()`** functions and passing in all the necessary parameters. The parameters passed in are a part of the testcase variable which is passed in as an input to the function. The run function returns information about the test cases run on the SUT, each test case run is also shown on the GUI like in Figure 4. The returned data is captured in the variable **`boxes_entered`**, **`dist_traveled`**, **`dist_from_origin`**, **`average_pos`** and **`no_interesting_steps`**. After the run function has been executed (which make take a varying degree of time depending on the length and complexity of the tests cases and the system on which they are being tested), the data can be analysed to determine the fitness value to assign to this individual.

The parameters used for analysing fitness scores are initialized to **`[]`** or **`0`** (line 39), so that any values from the previous execution of this function are not used. The **`test_sequence`** variable is an array of arrays (2-dimensional array). The outer array is for each **`test_step`**, and the inner array is the identifier of each box which was hit on that test step. To calculate the number of box-to-box connections, which was equal to the score of a test case, I first created a **`for`** loop (line 44) which iterated through each test step in the test sequence. Inside the loop, I firstly did a check whether the **`test_step`** length is greater than one using an **`if`** statement (line 45). If the number of boxes hit in a test step was greater than 1, then there was at least 1 box-to-box connection, and the fitness is therefore not zero. Otherwise, there was not a box-to-box connection and the **`if`** statement would return false, and it moves onto the next test step. If there is at least 1 box-to-box connection, it goes into the code inside the **`if`** statement. There may be 1 or more box to box connections, so I wrote another **`for`** loop that iterated a variable **`i`** for each entry in **`test_step`**. For each entry, I assigned the two box values that made up a single hit, into the variable **`box_to_box_hits`**. Once each test step had been looped though, the total number of box-to-box connections was calculated as the length of the **`box_to_box_hits`** variable.

The fitness value was then returned (line 82) back to DEAP framework which used this in the optimisation of the test cases.

```python
"""
This file is designed to simulate a method that has a feedback of the box to box hits made
"""
from TIG.MLCore.EA.simpleGa import run_ga, setup_genetic_operators
from turtle_test.input_definitions.turtle_schema import test_case_schema
from turtle_test.harness.run_boxes_test import run
from turtle_test.harness.run_boxes_test import draw_boxes
from deap import tools
import math
import time


def fitness(test_case):
    """
    Fitness function that returns the fitness value as the number of box_to_box connections a test case makes.
    Unique connections are not rewarded differently in this function.
    It will print the following for useful comparison with other fitness functions:
        - The overall fitness
        - The number of box to box hits (in this case, this is the same as the overall fitness)
        - The number of test steps
        - The time taken for the fitness function to run
    """

    # Start the timer
    start = time.time()

    # Rough boundaries of the cluster of boxes
    region_of_interest = [-150, -150, 150, 150]

    # Run Test
    boxes_entered, dist_traveled, dist_from_origin, average_pos, no_interesting_steps = \
        run(test_case, box_coordinates, region_of_interest)

    # Evaluate Fitness and Coverage
    test_sequence = [x for x in boxes_entered]
    no_of_steps = len(test_case.data["steps"])

    # Set parameters to zero for a new test
    box_to_box_hits = []
    this_run_unique = []
    num_of_hits = 0

    # Step through each test and if there is a box to box connection for that step, add it to the box_to_box_hits list
    for test_step in test_sequence:
        if len(test_step) > 1: # If the length is more than 1, then at least 2 boxes were hit
            for i in range(1, len(test_step)): # Step through the test step boxes hit in order
                box_to_box_hit = [test_step[i-1], test_step[i]] # Assign the box to box hit as a list of the two boxes
                box_to_box_hits.append(box_to_box_hit) # Add the hit to a list of all hits for this test case
    num_of_hits = len(box_to_box_hits) # The number of hits is the length of the list

    # Check for unique box_to_box_hits, add any unique hit to the list this_run_unique
    if len(box_to_box_hits) > 0:
        for hit in box_to_box_hits: # Loop through each box to box hit for the test case
            count = 0 # Reset the counter
            if unique_hits != []:
                for tc in unique_hits: # Loop through all the hits that have been recorded in the unique_hits list
                    if len(tc) == 1 and tc[0] == hit:
                        count += 1 # If there is a match, increment the counter
                    elif len(tc) > 1: # If the length of tc is more than 1, count the instances of hit
                        count += tc.count(hit) # Increment the counter by the count
            elif hit not in this_run_unique: # If unique_hits is emppty, add the hit to the this_run_unique list
                this_run_unique.append(hit)

            if count == 0 and hit not in this_run_unique:
                this_run_unique.append(hit) # If count is zero, this is a unique hit and add to this_run_unique

        unique_hits.append(this_run_unique) # Add all unique hits for this test case to the global unique_hits list

    if this_run_unique == []: unique_hits.append([]) # If there were no unique hits, append an empty list

    # Assign the fitness value as the number of box to box hits
    general_fitness = num_of_hits

    # End the timer
    end = time.time()

    # Print these parameters in this order to compare with other methods
    print("general_fitness", general_fitness)
    print("box_to_box_hits", len(box_to_box_hits))
    print("steps", no_of_steps)
    print("time", end - start)
    return (general_fitness,)

```

Figure 5: Fitness function written in PyCharm IDE

I also used print statements (lines 78 to 81) to print useful information to the console during the run. This allowed me to extract fitness, step length and time taken for each test case and analyse them.

Problem Solving

In order for this function to work as intended, all the imports, function inputs and retuned data must be in the correct format to integrate with the rest of the tooling. Otherwise, an error would occur during runtime. Also, the calculated fitness had to be correct or else the EA would be optimising for incorrect data and therefore would never produce the desired, optimised test case.

When I made the first iteration of this function and ran the script, I received the error shown in Figure 6. This error states that there is an assertion error somewhere in the tool. This means that the type of object that PyCharm was expecting is not the same as the actual object type it was given.

```
Traceback (most recent call last):
  File "C:/Users/adunne/PycharmProjects/test-input-generator/turtle_test/fitness_box_to_box.py", line 163, in <module>
    final_population = run_ga(pop_size, generations, cross_probability, mutation_probability, hof=hof)
  File "C:\Users\adunne\PycharmProjects\test-input-generator\TIG\MLCore\EA\simpleGa.py", line 70, in run_ga
    final_population = algorithms.eaSimple(p, toolbox, cross_prob, mut_prob, generations, halloffame=hof, verbose=False)
  File "C:\Python37-32\lib\site-packages\deap\algorithms.py", line 151, in eaSimple
    for ind, fit in zip(invalid_ind, fitnesses):
  File "C:/Users/adunne/PycharmProjects/test-input-generator/turtle_test/fitness_box_to_box.py", line 32, in fitness
    run(test_case, box_coordinates, region_of_interest)
  File "C:\Users\adunne\PycharmProjects\test-input-generator\turtle_test\harness\run_boxes_test.py", line 68, in run
    test_steps = load(test)["steps"]
  File "C:\Users\adunne\PycharmProjects\test-input-generator\turtle_test\harness\run_boxes_test.py", line 204, in load
    assert isinstance(test, dict)
AssertionError
```

Figure 6: Error Message

To solve this problem, I used the PyCharm debugger tool. The error message shows that the error occurs on line 32 of my Fitness function script. This is the where the test case is run using the run function. I placed a breakpoint on this line and ran the script in debug mode instead of running the code normally. Debugger mode allows the script to be paused at breakpoints and errors, as well as stepping through the script line by line, which gives the user an insight into what is going wrong.

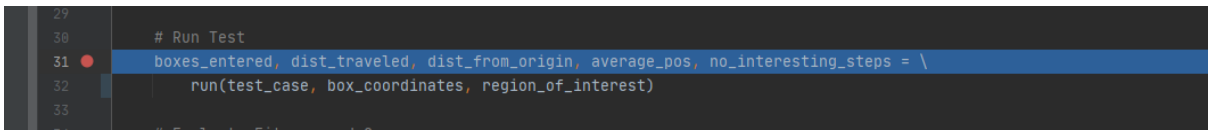


Figure 7: Running paused at a breakpoint using debugger mode

When the script had been paused at this point, which I knew was just before the error occurred from the error messages, I used the step into button. This took me to the beginning of the run function. I then pressed the resume button which ran the script up until the error occurred, where it paused again. Figure 8 shows the point where the assertion error occurs, it is an error indicated by the symbol on the line number.

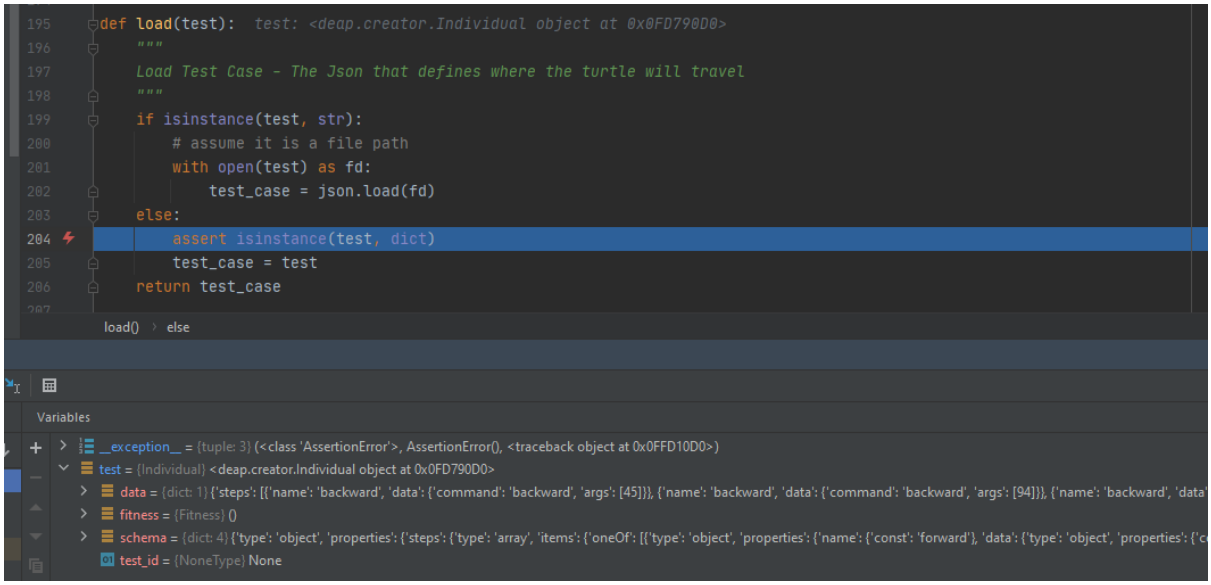


Figure 8: Running paused at the location of error in debugger mode

The **`assert isinstance()`** is used to ensure that the object being used in the load function is of the correct type as expected. This is used so if there is a mismatch, the error is clearly identified and the script stops, instead of being hidden whilst the script keeps running, which could potentially still run but with inaccurate data, which would be undetected and so is dangerous. The **`assert isinstance`** on line 204 is comparing that the parameter **`test`**, is of the type **`dict`**. In this case the parameter **`test`** is not a **`dict`** and so the assertion error is thrown. I can also tell this is the error point from the error messages (figure 6) which shows the assertion error occurs on line 204 in the **`load`** function, which is the same as shown in figure 8. Using the variables debugger window in Figure 8, I can see that the test parameter is a **`deap.creator.individual`** object. This is an object used by the DEAP framework, which I can attach data to. This is not the type of object I want however. The variable menu in Figure 8 also shows the data attribute attached to the test parameter is the dictionary of test steps that I actually want to load.

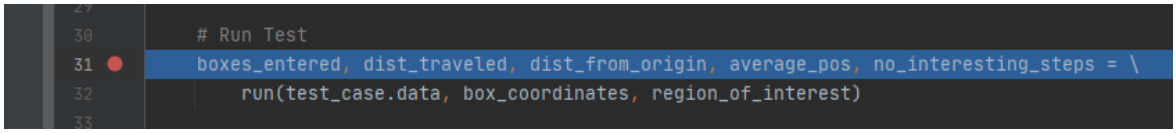


Figure 9: run function with the parameters corrected

Therefore, to fix this issue, I went back to the fitness function file and instead of passing the **`test_case`** into the run function, I swapped this to **`test_case.data`**. This meant the right data was passed in and when I ran this script, it worked as intended with no errors.

Source Control – Interpret and Follow

On this project, I used Jira, Bitbucket and Git to manage and save my work as it progressed. All three of these tools are used extensively within Capgemini. Therefore, there was a lot of company documentation on how to properly use them. Jira is an Atlassian tool that is used for Agile project management. Work is divided into 'tickets' which are then assigned to me, along with some details about the task and the expected timescale. The tickets and timescales were discussed and assigned during fortnightly scrum meetings with the whole team. I found the Jira UI to be very intuitive and it made seeing and organising my work easy. Figure 10 shows the Jira backlog, which shows tickets for a project in a table, along with the priority, assignee and category of the ticket.

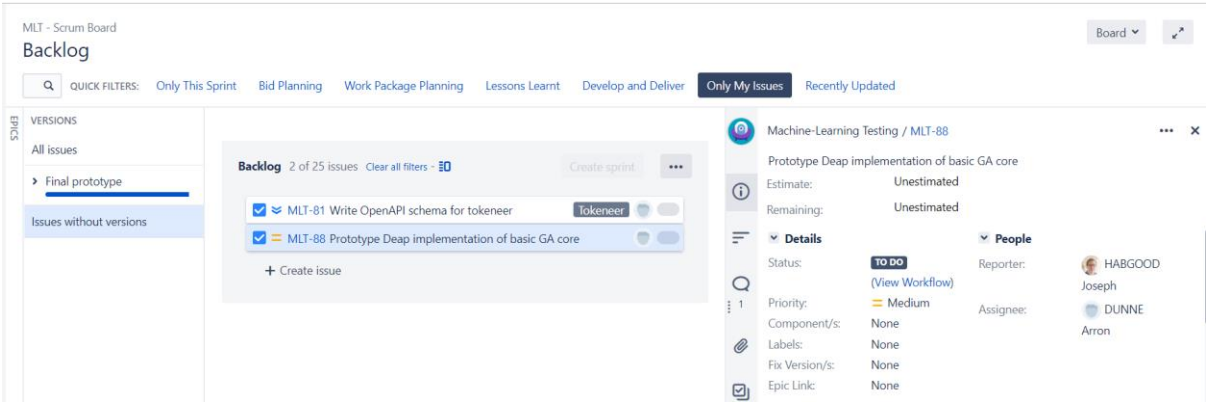


Figure 10: Jira backlog

BitBucket is a Git-Based code repository that I used to commit and save all my work to. BitBucket is also an Atlassian tool and so integrates very well with Jira. This meant I could create a git branch for tickets in Jira that then appeared in BitBucket ready to pull from. Also, pushing work to BitBucket updated the Jira ticket with the most recent commit messages and notified the Team Lead if there was a pull request. This integration made managing my work very smooth and I liked using these tools a lot. Bitbucket, like Jira, had a very intuitive UI with all the necessary functions were just a few clicks away for me.

When a ticket was completed and ready to be merged with the main code repository, the code had to be reviewed by a colleague before it can be merged into the develop branch. The company has a code review document that outlines how code reviews must be done. This is to ensure quality and consistency in work being added to the main code and is effective at finding mistakes. I had to read the code review process document and follow it for each code review I completed.

When my work was being reviewed, it would create a pull request on BitBucket and then notify the reviewer that the code was ready to review. The reviewer then left comments in Bitbucket which were either comments identifying areas that needed rework, or general comments. Then, I had to address all the comments which highlighted rework and commit the updated work. Figure 11 shows a comment on one of my pull requests that needed rework, and the comment left by myself outlining how I had addressed this comment. Once all the comments had been closed, the review was complete and with me and the reviewer agreeing that the work was complete, it could be merged.

Figure 11: BitBucket review comment

I was also asked to review work for my colleagues on their tickets. This was also outlined in the company review document. I had to ensure their code was formatted properly, with sufficient commenting. And I had to ensure that the code did what was intended and met the aims of the original

Jira ticket. Also, I could leave general comments which outlined potential efficiency improvements or future work, but these did not prevent the review from being completed. Once my colleague had addressed the rework, I would review it again to make sure all my previous comments had been sufficiently addressed before approving the work for merging.

The Company Git standards explained how Git must be used for all my work. It states that commit messages must be written in a certain format, with the task and Jira ticket that is being worked on included. If these are not provided the push will fail. I therefore had to adhere to this standard properly for my work to be added to the code repositories.

Business Environment

To meet budget requirements for this project, I had to meet all the deadlines and only use freely available software. The project budget allowed me and the other team member to work on this task for a specific timescale, and by the end of it the deliverable must be completed. This is what I did as discussed early. I therefore ensured that all budget requirements were met.

Agile

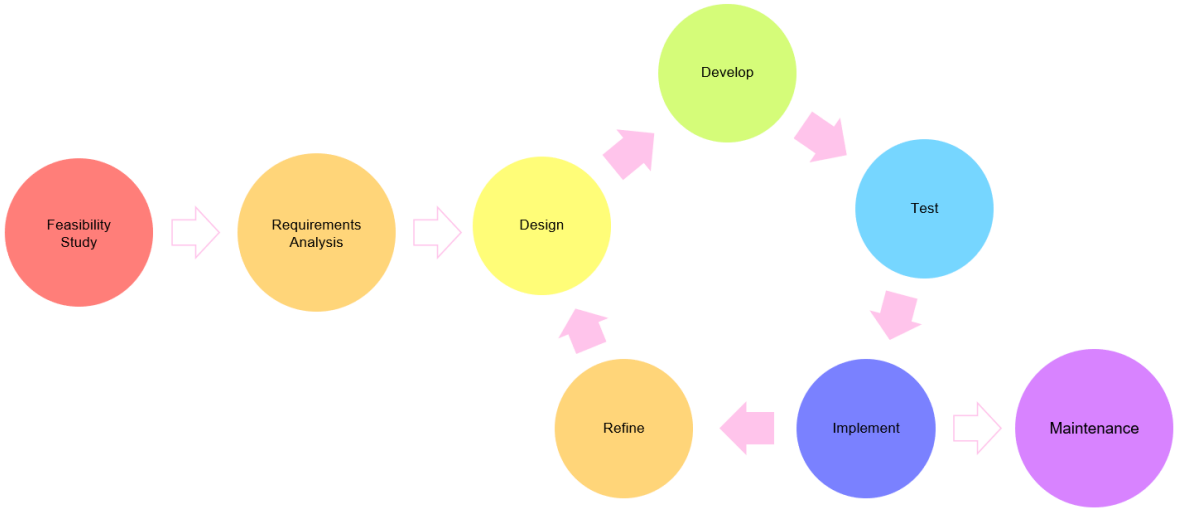


Figure 12: Agile workflow

I used an Agile development methodology for this project. Agile is an iterative approach where the development follows a loop of being developed, tested and then refined multiple times before the final version is released, as shown in Figure 12. Agile was a good choice for this project because it is very flexible and adaptable to new requirements from the stakeholder. Because I was developing a prototype, the requirements were likely to change as I discovered new techniques I could implement or found blockers which prevented me from implementing some aspects. Using Agile, it was easy and straightforward to change the requirements so new ideas could be incorporated and blockers could be removed. Agile also promotes continuous delivery, which was ideal for showing the stakeholders the progress I had made over time. I was working on a small team so we could communicate regularly and effectively to make the Agile workflow work.

The main disadvantage of Agile is that it can be a steep learning curve and requires someone who knows how to use agile properly for it to work. This was my first time using agile and I had a Team Leader who showed me how to properly work using it.

Project Conclusion

As my first project at the company, I learnt a lot of skills both technically and commercially. I experienced numerous stages of the software development lifecycle: requirements, design, and development. I also learnt how the company operates and how to use the source control and project planning tools, as well as company coding standards, and I applied these in a commercial application on this project. This was my first experience working on an Agile project, and I learnt the principles of Agile and the range of different job roles; scrum master, technical lead, project manager. The work I completed on this project was well received by the stakeholders both in the company and in the wider HICLASS community, which I was pleased with.

Project 2 – RBLS

Project Summary

My second project was on a project called RBLS (Radio-Based Limited Supervision). RBLS was a safety system for trains in the UK, which prevented trains from over speeding or passing red signals, among other functions. The purpose was to protect the train and reduce the dangers of driver error on the railway. In addition, the project was also a cheaper and more readily available solution for train safety in comparison to the more complex ETCS (European Train Control System) trains, which were more expensive and not expected to be fully integrated on UK railways for many years.

I joined the project in the requirements stage of the lifecycle, following the approval for it in the feasibility stage by various bodies. I was a part of the requirements ream which was led by James Nevell as Project Manager and Alan Newton as Technical Lead for my team. The team had a total of 6 people who were all requirements' engineers. My responsibilities were to elicit requirements for the system that would allow it to meet all its safety, operational and commercial goals and be as beneficial as possible to the stakeholders. The main stakeholder was Network Rail, and Capgemini worked closely with rail industry experts from different companies who helped in discussions of the project.

Analysis

RBLS is a large and complex system, with many different operational uses and functions. As part of the team, I needed to find a way to make sure that we captured all the necessary requirements for all these different scenarios at this early stage of the project. Capturing the requirements as early as possible reduces the cost of implementing them compared with adding them later in the development cycle. Therefore, it was an important factor in meeting the cost and business targets for the whole project.

The requirements were elicited by analysing the operational behaviour that was desired from the system and use this to write formal requirements. To analyse the different operational scenarios the system would be operating in, Use Cases were created. Use Cases were used as stories for the system in operation. A single Use Case captured a particular event that the system would be operating in. Some examples of the Use Cases used are:

- Emergency Brake being applied due to over-speeding
- Starting up the system
- Overriding the system so the driver can pass a red signal

Each Use Case consisted of a title, a description, and pre and post conditions. The pre and post conditions described the state of the system before and after the Use Case and showed the state change that was required. I helped to write the description of the Use Case by speaking with industry experts about what the desired behaviour would be based on the stakeholder needs and desires.

After the Use Case had been created one of my tasks was to break it down into Steps. The Steps captured functions and changes of the system at a lower level. They were used to break down the Use Case into a series of ordered steps, which describes exactly what the system would do through the use case. Each Step captured a single function/change in order to make the sequence of events as clear as possible.

To create the steps, I spoke with the specialists from the company with expertise in human factors, safety and other important disciplines, as well as experts from external Rail Companies to discuss and decide what the steps should be. These were very important meeting and contributed to the design and architecture of the system. Use Case captured the goal of the system, but the Steps captured the behaviour of the system, and therefore what functions and features it needed to achieve. Figure 13 shows the steps for the Start of Mission Use Case.

Steps

Step ID	Step
STP-0552	(Context) Driver Switches\Enters Cab
STP-0553	(Context) Proceed Signal is displayed to Driver
STP-0554	(Context) Driver executes Key On (Open Desk)
STP-0467	(Product) The LS-OBS starts-up and performs start-up health check [SBIT] (including relevant Security Artefact checking)
STP-0640	(System) The RBLS System (Onboard) starts-up and performs start-up health check [SBIT] (including Security Artefact and Configuration checking)
STP-0411	(Product) The LS-OBS logs the events internally
STP-0391	(Product) The LS-TsP logs data internally
STP-0412	(System) The RBLS System logs events internally
STP-0413	(Product) The LS-OBS sends a logging message to the Data Recorder within 5 seconds of a triggering event (SS 027 4.1.1.4)
STP-0414	(System) The RBLS System (Onboard) sends a logging message to the Data Recorder within 5 seconds of a triggering event (SS 027 4.1.1.4)
STP-0550	(Product) LS-OBS periodically monitors to detect faults [CBIT]
STP-0551	(System) RBLS System (Onboard) periodically monitors to detect faults [CBIT]

Figure 13: Steps for Start of Mission Use Case

The Steps were split into three categories:

- **Context Steps**: Steps which were not done by the system but by the environment interfaced with the system
- **Product Steps**: Steps that were done by a particular product/component of the system
- **System Steps**: Steps done by the whole system

Requirements

The Steps were then used to derive the requirements. For each Step in a Use Case, I would analyse it and discuss with the Team, what were the requirements that are needed in order to perform the step. It was important to focus on capturing the goals of each step. I wrote the requirements to describe what the System needed to achieve in a particular step, and not exactly how the System should do it. This was the principle behind REVEAL and allows flexibility for optimisation in the Design Stage.

REVEAL is an approach used by the company to identify requirements early and ensure there are no conflicts between requirements. It emphasises having correct and clear requirements that capture a specific need of the system. This method cuts down costs, because finding conflicts later in the process is more expensive to fix.

Another important concept of REVEAL is to completely define the system boundary, which is how the system interfaces with the environment. The system boundary tells us in what ways the system can

change the environment, and how the environment can change the system. Once this is known, we can capture the behaviour over this interface.

Domain is also an important part of REVEAL, as this helps make assumptions about the system and identify gaps. On RBLS, I spoke with external professional in the rail industry to establish domain knowledge. I worked with people specialised in train and railway infrastructure, signalling and ETCS (European Train Control System) on a regular basis.

Once I had written the requirements, they were reviewed by the Technical Lead, and the Stakeholder. If there was anything that needed changing, they left a comment, and I would discuss with them whether the changes were needed and what exactly should be changed. This communication with the stakeholder gave me lots of good experience with dealing with clients and compromising between their needs and what was feasible for the project.

Design

As part of the project, the System Architecture was created. The Architecture explains how different components of the system interfaced which each other, and how the overall system interfaced with the environment. This was then used as a starting point for the design work. In order to design the system correctly, the Architecture had to include all the interfaces on the system. Part of my work was to check the architecture fit in with the requirements I was creating from the Use Cases analysis.

One of the Use Cases was the logging of information to a digital log. The steps for this Use Case were developed and the requitements defined an interface between the system and the on-board log, which is outside the system boundary. However, when I checked the Architecture to ensure this interface was properly defined, I found the Data Recorder was missing. I notified the Technical Lead of the project and discussed the need for this component on the architecture because the system has a direct interface with it. I also explained how I had arrived at the requirements for this interface using the Use Case for logging data.

I updated the architecture to include this component as being interfaced with the system, and therefore the requirements I had generated matched the architecture. I then notified everyone on the project the changes to the Architecture to ensure everyone was working with the latest version.

Waterfall

The project followed a waterfall methodology, which is common for large scale infrastructure projects. This is because there is not the flexibility to continuously iterate from requirements to testing, as there are large costs and time involved with each stage. This makes Agile an unsuitable method. As mentioned in the summary, I joined the project in the requirement stage, following the feasibility.

Working in a waterwall manner meant that this was the only time requirements would be created on the whole project. It was therefore essential for the correct performance of the system, as well as meeting all business targets, that the correct requirements were made that covered the entirety of the project.

Requirements Standards - Interpret & Follow

The aims of the RBLS project I worked on was to capture the needs of the system and derive requirements. Because of the large scope of the projects, which had a large team working on it, consistency was vital. To maintain consistency, the project had Management Plans for the different areas of work including requirements. These plans laid out the way in which work should be done to ensure quality and efficiency. These helped the project stay in budget and meet deadlines.

As part of the Requirements Team, I had to follow the 'Requirements Management Plan' carefully. This plan provided an outline of the methodology used to capture requirements, how requirements should be structures and how requirements should be linked back to Use Cases and Business Benefits to show the value gained for each requirement.

When I initially joined the project, I read through the 'Requirements Management Plan' and took down notes on the key topics. One of the main standards was that all requirements must be written in EARS format. EARS format is a widely used format for requirement and states that requirements must follow one of the five fundamental patterns. These patterns are:

- Ubiquitous: Always active, not driven from an event or input
    - The system Shall …
- State-Driven: Active throughout the time a defined state remains true
    - While [state is active] the system Shall …
- Event Driven: A response when an event is detected at the system boundary
    - When [trigger] the system Shall …
- Optional Feature: Applies when an optional feature is present
    - Where [feature] the system Shall…
- Unwanted Behaviour: Covers situations that are undesirable
    - If [trigger] then the system Shall …

Business Environment

This was a large and complicated project and lots of different companies were partners on it. Therefore, there were lots of business considerations throughout to ensure budgets, deadlines and quality was achieved.

In the middle of the project, the number of Use Cases completed was behind the target for the time. The management team decided that to reduce the time taken for each Use Case, the requirements team should be split into two. I went from a team of 6 onto a team of 3. To maintain the same level of quality I had to take on some extra responsibilities. The main one I took on was taking a more active part in the focus meetings and dealing more with experts to decide what functionality the system should have.

I had to manage my time more effectively, and due to my extra responsibilities, had to reduce the time taken on other tasks. I spoke with my Team Leader about which tasks I needed to prioritise and managed my time according to these. I was also required to maintain good communication with other requirements team, as the work would occasionally overlap between us. I had to give regular updates to the other team on what work I had done, what I was working on, and what I planned to work on next. This prevented both teams working on the same Use Case, and duplicating efforts, which was a main risk of working in separate teams.

After splitting into separate teams, I helped to successfully reduce the time taken for Use Cases to be completed. The project caught back up to the target number of Use Cases with a month and met the deadline by the end. I also prevented any work being duplicated and the two teams worked very efficiently in parallel.

Project Conclusion

This project was very client facing and so I had a lot of commercial experience working with external experts. I learnt a lot about the requirements stage of the lifecycle, as well as how the decisions made about requirements effect other stages of the lifecycle, such as design and development. I learnt about

the REVEAL technique used by the company for gathering requirements and the EARS format that is used. This was also the first Waterfall project I worked on I so got to experience a different methodology, which was great experience, and I learnt a lot about the differences between Agile and Waterfall and the different projects they are suited for. As I worked with the stakeholders a lot on this project, I got regular feedback about the work I had done, which was mostly positive. And any issues they saw I could address quickly so they were happy with my work.

Project 3 – Database

Project Summary

Following my work on RBLS I went back to the HICLASS project. This time I was in a different team and had a different project. I worked on a tool which automatically generated tests based on a specification of the system. This tool was still in the early stages of development. The Team Leader was Thomas Wilson and the main stakeholders were the members on the HICLASS project.

The task I was given was to improve the memory efficiency of this tool by utilising databases to store information instead of local memory. To do this I would convert one of the data objects, **`TestConditions`**, which is stored locally, and save the data in a database instead. This would free up local memory. Then I had to write methods that fetched the necessary data from the database and return it, according to an interface class, which laid out the required methods. Implementing the interface meant that the new implementation would work with the existing architecture. The benefit of using a database would be to reduce the amount of data that is stored locally and reduce the risk of the tool failing due to insufficient memory.

Data

To begin with, I identified all the data contained within the **`TestConditions`** object and what could be stored in a database. Then I designed a Database schema based on whether each object had a single attribute or many of them. Figure 14 shows the Database schema I designed. It was approved by the technical lead and contained all the necessary data for it to work properly.

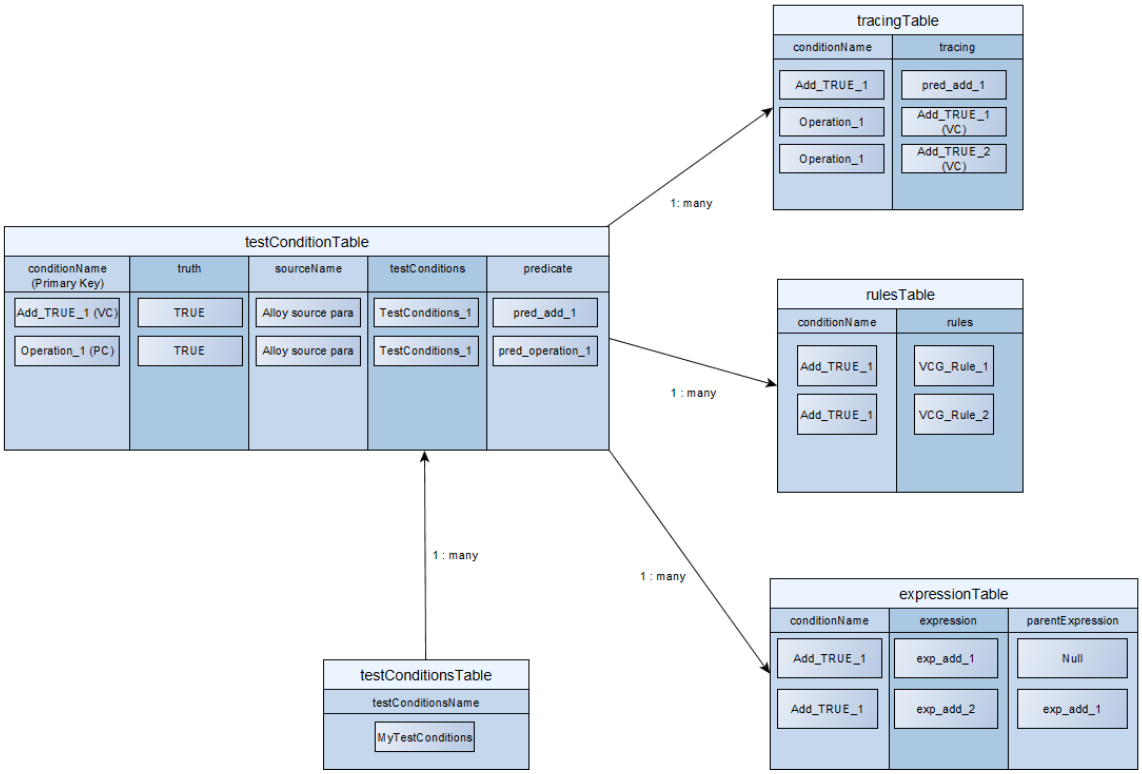


Figure 14: Database schema

Logic

Once the design was complete, I began writing the code for this new implementation. I created two new classes named **`DatabaseTestConditions`** and **`DatabaseTestCondition`** which implemented the interfaces **`TestConditions`** and **`TestCondition`** respectively.

```java
/**
 * This class takes a {@link TestConditions} object as input and transfers the data into a database
 * which is then read using SQLite statements when the interface methods are called. The database is
 * stored in the output directory at /conds_db/condition.db
 */
public class DatabaseTestConditions implements TestConditions {

  private final String databasePath;
  private Connection conn = null;
  private String name;
  private List<DatabaseTestCondition> dbTcList;

  public DatabaseTestConditions(TestConditions tc, String name) {
    this.name = name;
    new File(Settings.OUTPUT_FOLDER + "/conds_db").mkdir();
    databasePath = Settings.OUTPUT_FOLDER + "/conds_db/conditions.db";

    // Return early if we couldn't make the connection as subsequent statements will only cause
    // exceptions
    if (!connect()) {
      Logger.error("Unable to read database: " + databasePath);
    }

    try (Statement statement = conn.createStatement()) {
      // Create the tables (if they don't already exist)
      statement.execute("CREATE TABLE IF NOT EXISTS testCondition ("
          + " testConditionName text NOT NULL,"
          + " truth text,"
          + " sourceName text NOT NULL,"
          + " testConditions text NOT NULL,"
          + " predicate blob NOT NULL);");

      statement.execute("CREATE TABLE IF NOT EXISTS tracing ("
          + " testConditionName text NOT NULL,"
          + " tracing text NOT NULL);");

      statement.execute("CREATE TABLE IF NOT EXISTS rules ("
          + " testConditionName text NOT NULL,"
          + " rule text NOT NULL);");

      statement.execute("CREATE TABLE IF NOT EXISTS expression ("
          + " testConditionName text NOT NULL,"
          + " expressionId integer NOT NULL,"
          + " expression blob NOT NULL,"
          + " parentExpressionId integer);");

      statement.execute("CREATE TABLE IF NOT EXISTS testConditions ("
          + " testConditionsName text NOT NULL);");

      // Clear the Database
      statement.execute("DELETE FROM testCondition;");
      statement.execute("DELETE FROM tracing;");
      statement.execute("DELETE FROM rules;");
      statement.execute("DELETE FROM testConditions;");
      statement.execute("DELETE FROM expression;");

    } catch (SQLException e) {
      Logger.error(e, "Error initialising test input database");
    }
```

Figure 15: Database Test Conditions class

Figure 15 shows the **`DatabaseTestConditions`** class. The constructor is also shown, starting on line 39. It implements the **`TestConditions`** Interface (line 32) which is necessary to allow it to integrate properly with the architecture. There is a database path and a database connection stored locally in the variables **`databasePath`** (line 34) and **`conn`** (line 35) respectively. These variables were needed to execute commands, such as storing and retrieving data, from the database. A name for the object and a list of **`DatabaseTestCondition`** objects is also stored locally because these are needed for reference in the tool.

The java SQL library is used for working with the database. This was used because it has good integration with Java and eclipse and has been used in the company before, so I was able to discuss and learn how to use it from others. I was able to use the methods from the SQL Library by importing it at shown in Figure 16.

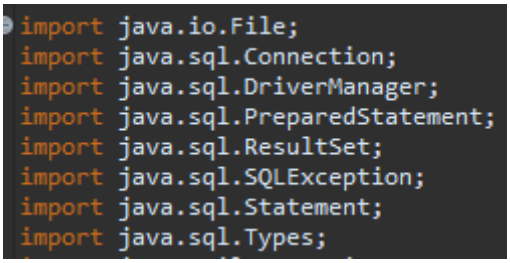


Figure 16: Java Import Statements

In the constructor of the **`DatabaseTestConditions`** class, the database file is created (line 41) and then a connection is made (line 46). Failure to connect will result in an error message being logged (line 47). After that the database is populated with data. All the database SQL statements are formatted in a try, catch statement. This is so that if the connection to the database is broken at any time, and the SQL statements fail to execute, there will be an error message logged and this will help with debugging. The catch section catches **`SQLExceptions`** which are part of the Java SQL library, imported at the beginning of the class (Figure 16). Also in the constructor, if the connection is successful the Database tables are created. This is done using SQL statements using the **`CREATE`** keyword inside the java command **`statement.execute`**, which is how the **`java.sql`** library is used. The tables are created as designed in the schema. I wrote the **`CREATE`** statements using **`IF NOT EXISTS`**. This means that if the table already exists, it will not be created a second time, and improves the efficiency of the class and also prevents errors with creating multiple of the same table. At the end of the constructor (line 77), I execute a **`DELETE`** SQL command which will clear the database from any previous entries from previous runs as these could cause errors.

```
    // Get the condition map of the passed in test conditions, which will be used for adding
    // data to the database
    Map<String, Map<String, TestCondition>> condMap = tc.getCondMap();

    // Loop through each predicate in the condition map
    for (String k : condMap.keySet()) {
      Map<String, TestCondition> inner = condMap.get(k);
      Set<String> tcNames = inner.keySet();

      Iterator<String> i = tcNames.iterator();
      // Iterate through each test condition
      while (i.hasNext()) {

        String tcName = i.next();
        TestCondition tcObject = inner.get(tcName);
        // Convert data into DatabaseTestCondition objects
        DatabaseTestCondition dbtc = new DatabaseTestCondition(tcName, conn);
        // Add data to the database
        recordTestCondition(tcName, tcObject.getTruth().toString(),
            tc.getSourceParaIdForPredWithName(tcName), this.name, tc.getPredWithName(tcName));
        recordTracing(tcName, tcObject.getTracing());
        recordRules(tcName, tcObject.getRules());

        // Get the parent expressions and add them to database with a null parent expression
        List<Expression> rootExpressions = tcObject.getRootExpressions();
        for (Expression ex : rootExpressions) {
          recordExpression(tcName, ex, null);
        }

        Map<Expression, List<Expression>> childrenParentMap = tcObject.getChildren();
        // Get the child expressions and add then to the database with the correct parent value
        for (Expression parent : childrenParentMap.keySet()) {
          List<Expression> children = childrenParentMap.get(parent);
          for (Expression child : children) {
            recordExpression(tcName, child, parent);
          }
        }

        // Save the Database test condition in the array for future reference
        dbTcList.add(dbtc);
      }

    }
    // Add the test conditions name to the database
    recordTestConditions(this.name);

  }
```

Figure 17: Storing data into the database

```
  /**
   * Add test condition data to the database.
   */
  public void recordTestCondition(String tcName, String truth, String sourceName,
      String testConditions, Pred pred) {
    final String insertSql =
        "INSERT INTO testCondition (testConditionName, truth, sourceName, testConditions, predicate)"
            + "VALUES (? ,?, ?, ?, ?);";
    try (PreparedStatement insertStatement = conn.prepareStatement(insertSql)) {
      insertStatement.setString(1, tcName);
      insertStatement.setString(2, truth);
      insertStatement.setString(3, sourceName);
      insertStatement.setString(4, testConditions);
      insertStatement.setBytes(5, SerialisationUtils.serialise(pred));
      insertStatement.execute();
    } catch (SQLException e) {
      Logger.error(e, "Failed to write to database");
    }
  }
```

Figure 18: Inserting data into the database

Next the data to be stored in the database (which is stored with the **`condMap`** variable) is looped through, and the relevant data is passed into a function which records the data in the database as shown in Figure 17 and 18. The records functions for all the tables follow similar processes so I have

only outlined one here (Figure 18). The Java garbage collector helps delete the data once it has been used and allows the memory needed to be reduced.

The information to be put in the database is passed in as a string. Then, A SQL statement is created in string format. The **`VALUES`** keyword is used instead of string concatenation to prevent the risk of SQL injection which could have occurred without this.

Then the statement is executed on the database, and the **`VALUES`** are set the corresponding inputs. The SQL statement is enclosed in a try catch branch in case of any errors with the database, and the error message helps identify what went wrong.

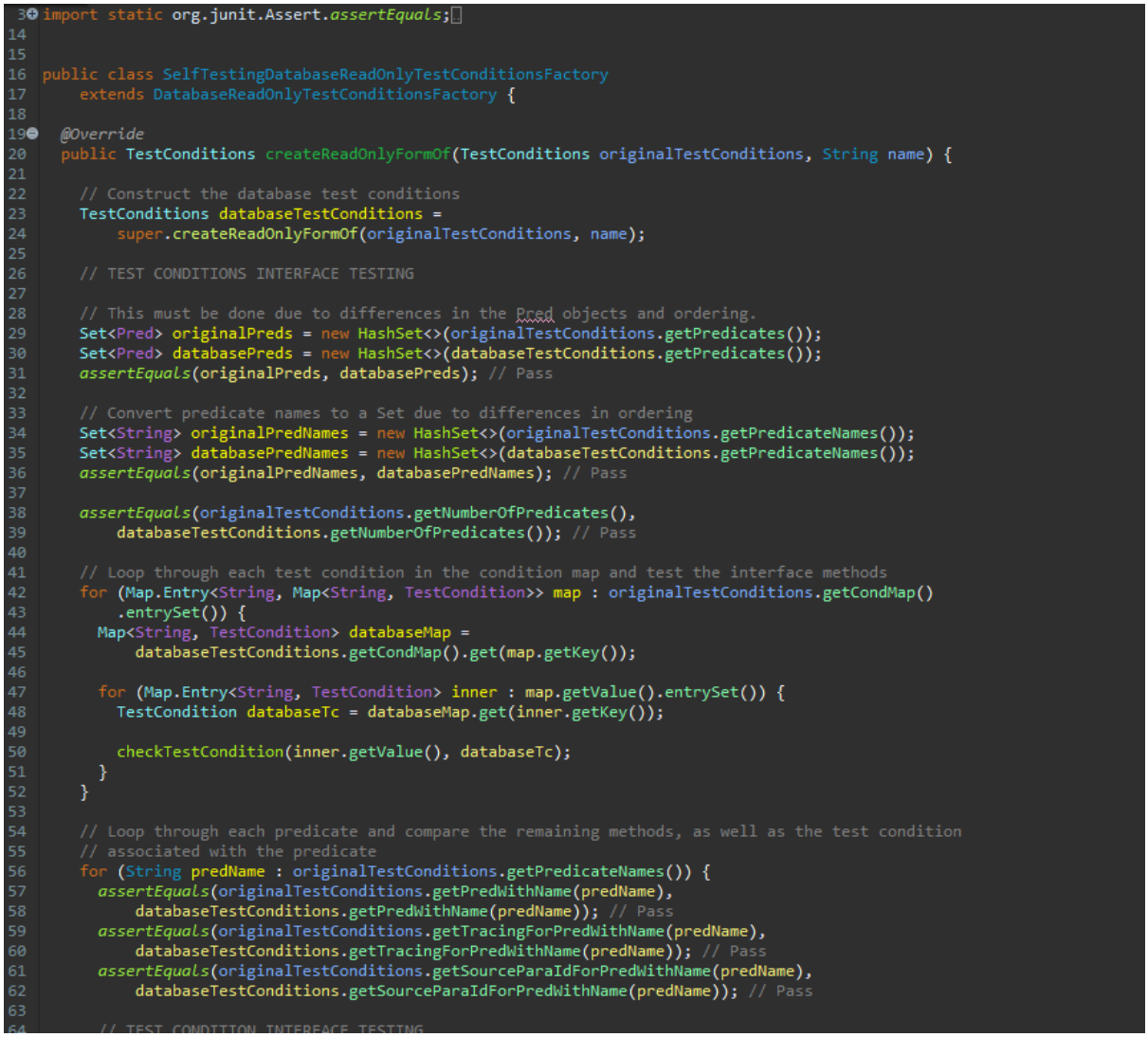
```
268  /**
269   * Select all predicates stored in the database for this test conditions object and return it as a
270   * list of predicates.
271   */
272  @Override
273  public List<Pred> getPredicates() {
274    List<Pred> predsList = new ArrayList<Pred>();
275
276    final String selectSql = "SELECT predicate FROM testCondition WHERE testConditions = ?;";
277
278    try (PreparedStatement selectStatement = conn.prepareStatement(selectSql)) {
279      selectStatement.setString(1, this.name);
280      ResultSet rs = selectStatement.executeQuery();
281      while (rs.next()) {
282        byte[] predBytes = rs.getBytes(1);
283        predsList.add((Pred) SerialisationUtils.deserialise(predBytes));
284      }
285    } catch (SQLException e) {
286      Logger.error(e, "Error reading from database");
287    }
288    return predsList;
289  }
```

Figure 19: Retrieving data from the database

Retrieving data from the database follows a similar structure, as shown in Figure 19. The SQL statement is constructed and tried. For this example, the test condition value is stored in the class so can be directly accessed using this.name. For other cases, the values that is being selected is passed into the function as an input, like the record. A result set is then sent back from the database. This result set is looped through, and each entry is added to the local variable **`predsList`**. This list contains the list of objects and is returned. The necessary format for the returned results were in the type **`Pred`**, but the database returns them in for format of string. Therefore, to turn them into the desired format, they are cast to a **`Pred`** object (line 283) before being added to the **`predsList`** variable.

Test

To ensure that my implementation was correct, I wrote a Junit test script which compared the results from interface methods for the original implementation and the database implementation. This is shown in Figure 20. For each Interface method, I compared the results of the original implementation (which did not use a database but stored all the data locally instead) and the database implementation I had created. For each test, the interface method is called on the original and database test conditions and the I used the **`assertEquals`** function to ensure they were the same. If the tests failed, they would be flagged in the IDE.



```
import static org.junit.Assert.assertEquals;


public class SelfTestingDatabaseReadOnlyTestConditionsFactory
    extends DatabaseReadOnlyTestConditionsFactory {

  @Override
  public TestConditions createReadOnlyFormOf(TestConditions originalTestConditions, String name) {

    // Construct the database test conditions
    TestConditions databaseTestConditions =
        super.createReadOnlyFormOf(originalTestConditions, name);

    // TEST CONDITIONS INTERFACE TESTING

    // This must be done due to differences in the Pred objects and ordering.
    Set<Pred> originalPreds = new HashSet<>(originalTestConditions.getPredicates());
    Set<Pred> databasePreds = new HashSet<>(databaseTestConditions.getPredicates());
    assertEquals(originalPreds, databasePreds); // Pass

    // Convert predicate names to a Set due to differences in ordering
    Set<String> originalPredNames = new HashSet<>(originalTestConditions.getPredicateNames());
    Set<String> databasePredNames = new HashSet<>(databaseTestConditions.getPredicateNames());
    assertEquals(originalPredNames, databasePredNames); // Pass

    assertEquals(originalTestConditions.getNumberOfPredicates(),
        databaseTestConditions.getNumberOfPredicates()); // Pass

    // Loop through each test condition in the condition map and test the interface methods
    for (Map.Entry<String, Map<String, TestCondition>> map : originalTestConditions.getCondMap()
        .entrySet()) {
      Map<String, TestCondition> databaseMap =
          databaseTestConditions.getCondMap().get(map.getKey());

      for (Map.Entry<String, TestCondition> inner : map.getValue().entrySet()) {
        TestCondition databaseTc = databaseMap.get(inner.getKey());

        checkTestCondition(inner.getValue(), databaseTc);
      }
    }

    // Loop through each predicate and compare the remaining methods, as well as the test condition
    // associated with the predicate
    for (String predName : originalTestConditions.getPredicateNames()) {
      assertEquals(originalTestConditions.getPredWithName(predName),
          databaseTestConditions.getPredWithName(predName)); // Pass
      assertEquals(originalTestConditions.getTracingForPredWithName(predName),
          databaseTestConditions.getTracingForPredWithName(predName)); // Pass
      assertEquals(originalTestConditions.getSourceParaIdForPredWithName(predName),
          databaseTestConditions.getSourceParaIdForPredWithName(predName)); // Pass

      // TEST CONDITION INTERFACE TESTING
```

Figure 20: Junit Test Script

Project Conclusion

During this project, I learn how to create a database and execute CRUD commands on it. This included, creating table, adding data to table, reading data from tables and deleting the contents in a table. I also designed a database schema and then implemented this schema in Eclipse using the Java SQL library. I then tested this using Java Junit tests. This taught me how to perform tests using interface methods and run Junit tests on my code. I learnt a lot of coding skills and improved my Java and SQL knowledge. My implementation successfully reduced the local memory whilst the system was running, which reduced the risk of the system having insufficient memory to run. The stakeholder of the project was pleased with my implementation and the quality of my work.

Project 4 – Dashboard

Project Summary

I continued the HICLASS team throughout the end of 2021, working on improving the tools. Another useful part of the tools is that there is a HTML plugin which, when using, produces several webpages with information about the tool as it processes verification conditions (VCs) and path conditions (PCs) for testing. This provides a user-friendly and intuitive way to analyse the processing going on, without needing any in-depth knowledge of the tool or reading from console outputs. This makes It very useful

for engineers who will be using the tools for testing but did not work on the development of the tool. I was tasked with adding a new webpage which contains more information about the process, to give as much detail to users as possible.

User Interface

To begin with, I spoke with software engineers on the team on what data they would be most interested in about the tools. We concluded that having information about the progress of VC processing over time would be very beneficial.

I decided that this information would be most easily viewed in a graph of number processed over time. As well as having separate series for the different status. I used Plotly for this task because it was recommended by other members on the team who had used it, and after looking at the documentation, it was designed for producing graphs in HTML, so was a perfect fit for this task.

The HICLASS tools are written in Java, in the Eclipse IDE. I used the existing HTML framework as a starting point and created a new class **`DashboardHTML`**. This class has a static method of **`generateDashboardHTML`**, which is called from the plugin manager every 10 VCs. When the method is called, the data for the status of VCs and PCs is passed in, and then looped through to count the number of each status which had been processed at that point in time. Using static data, I implemented a list which stored the number processed for each time step.

I used the Plotly **`plot`** function to produce the graphs, shown in Figure 21. This is embedded into the HTML body, and the Plotly library is included in the HTML head. The Plotly **`plot`** function takes three arguments:

- Plot: This is the div that the graph should be drawn on
- Data: The data which is shown in the plot (data lines, series names, marker, and line styling)
- Layout: The formatting of the plot (titles, axis labels, grid)

```
// VC plot
html.append("<div id=\"vc_plot\" style=\"width:1000px;height:250px;\">");
html.append("<script>\r\n" +
    "    vcPlot = document.getElementById('vc_plot');\r\n" +
    "    Plotly.newPlot( vcPlot, " + vcPlotData + ", " + vcPlotLayout + ");\r\n" +
    "</script>\n");
html.append("</div>");
```

Figure 21: Plotly embedded in HTML

Once this was done, I showed the dashboard page to members in the team to ask for feedback. People found the data useful and shown in a clear manner. They gave me a suggestion that being able to see the time taken for each PC to be processed would be an additional useful piece of information which would help with improving performance and debugging.

To do this new graph, I stored the system time when the PC began processing and when the PC had finished being processed, I found the difference between the current system time and the system time at the start, which gave me the time elapsed to process that PC. The timing data is plotted on a separate graph. Similarly, to the other graphs, different status is shown in different colours to provide additional information to the user. An average line was also calculated.

Figure 22: Dashboard HTML User Interface

This is the seconds iteration of the dashboard page and contains the three plots suggested as well as a metrics table which provides the time taken overall and the current progress as a percentage. The Plotly graphs have many additional features such as being able to download the graphs as png files, zoom in and pan on the graphs as well as hide or show series by clicking them on the legend. I showed this to my team members, and they were pleased with the amount of data which could be seen in them, as well as the simplicity of the page.

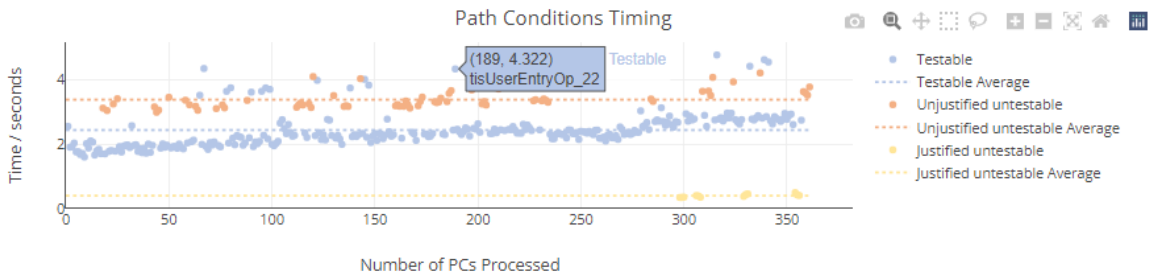


Figure 23: Interactive Plotly graph

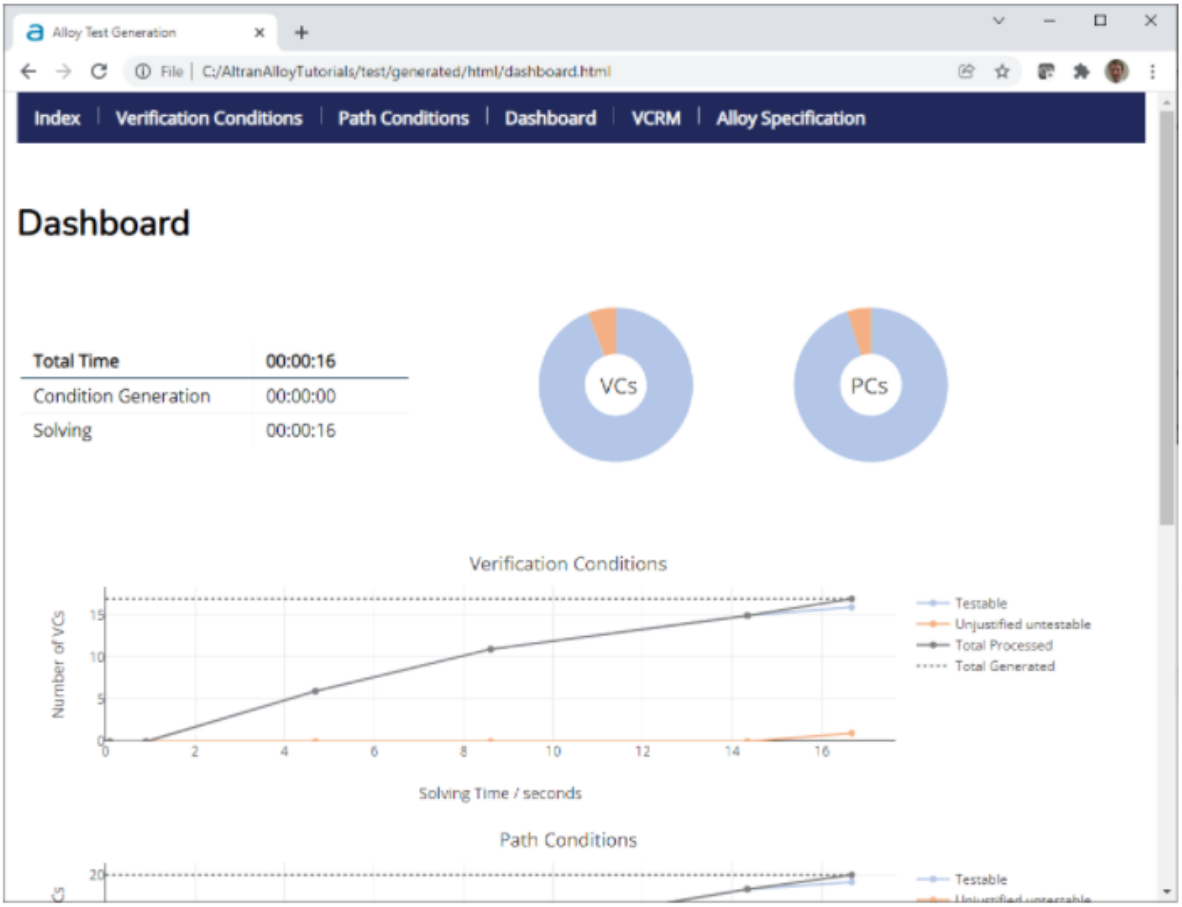
Deployment

Once I had completed the work. I created a pull-request on Jira for it to be reviewed and merged into the main repository on Bitbucket. Once the work had been merged, I created some user documents. This was so that users who had no knowledge of the dashboard page could learn how to use it and what information it shows. The user documents outlined the contents of the Dashboard page I had created and some information on how to use the Plotly interactive functions. Figure 24 shows the documentation.

Dashboard page

The Dashboard page presents a range of metrics relating to the test generation process.

The start of the page is:



This shows:

1. The total time taken for test generation, and a breakdown between the stages.
2. Pie charts showing the current proportion of statuses for VCs and PCs.
3. A graph of VCs processed against solving time.
   - The x-axis shows the length of time elapsed (which will change from, seconds, minutes or hours depending on how long the run is). The y-axis shows the number of VCs which have been processed. Separate lines are drawn for different statuses as well as the total number processed (black solid line) and the total number generated (black dashed line).

Figure 24: Dashboard User Documentation

Maths

One of the features that was requested by the team was to show the average time taken to solve for each PC status type, on the same plot as the individual PC processing time. I therefore had to calculate the average time using the individual time data I had stored.

```
// calculate average
Double count = 0d;
for (String point : yData) {
  count += Double.valueOf(point);
}
Double average = count / yData.size();

statusTiming.put("xData", xData);
statusTiming.put("yData", yData);
statusTiming.put("names", predNames);
statusTiming.put("average", Arrays.asList("" + average));
```

Figure 25: Calculating average

Figure 25 shows the code I wrote to calculate the average in Java. To calculate the average time, first, a counter is set to zero. Then I iterate through each data point and add the time value to the count parameter. After summing all the times for this status, I divide the count by the number of data points. This gives me the mean time to solve each PC for a particular status. I then add this data into the **`statusTiming`** variable, which gets plotting onto the graph. The average data gives a clear indication of which status are quicker of slower compared to others and is a valuable piece of information for the user.

Project Conclusion

On this project I learnt how to design and make a user interface which showed the performance of a system. I spoke with users on what the most useful information would be to include on the UI, and then created a design based on this. Next, I created the UI using HTML and the Plotly library. I showed the UI to the users who suggested adding some other useful information, and I added these suggestions to a second version of the UI. The UI was interactive and user friendly and showed a lot of useful information for both users of the tool and developers of the tool. The stakeholder, who was responsible for the tool, was happy with my addition to the HTML output. I also learnt how to write user documentation and release this to users.

Business Environment

As an employee of the company, I followed the policies and regulations of the business environment that I was working in. The Employee Handbook, which was given to me when I started, contains the company policies for Security, Health & Safety, GDPR, care of company property, personal appearance and more. I had to learn these policies and follow them.

Security

The safety critical nature of work that the company does means that some information must remain confidential and could cause damage to the company and clients if it were to be released to the outside world. This is especially true for projects which may contain government information that can only be viewed by people with the correct UK Government security clearance. Because of these factors, the company has a very strict and thorough security policy to prevent any security related issues occurring. During my induction, I was given training about the 4 areas of security at the company:

- **Staff Security** – ensure all employees have the correct security clearance for their work
- **Building Security** – ensure only authorised people can enter the companies' premises
- **Information Security** – ensure company information is not leaked to the outside world
- **IT Security** – ensure company hardware is secure from physical or virtual threats

I also learnt how to follow these security aspects in my job role. For staff security, this was primarily done during the recruitment phase. After I had agreed my contract with the company, but before I had started my first day of work, I had to complete a pre-employment check. I provided the company with my personal information so they could verify that I was not a malicious person or a threat to the company. The company operates security related projects on a need-to-know basis. Therefore, when I started, I agreed to the basic level of security in the office, which does not allow me access to higher levels of protected data. As part of the security policy, I must not enter higher security areas.

Building security is enforced with the use of staff badges, among other aspects. I was given a staff badge which gave my access to the office through the locked doors. The company's policy is that all members of staff must have their badges visible when inside the company premises, to show that I am allowed the access the building. I must also be vigilant when entering and exiting the office, to prevent people following me through the doors without scanning their security badge.

As a software-based company, all employees have a laptop which can connect to the internet. This makes it a potential target for hacking attempts or stealing data from. My laptop is protected by Anti-Virus protection, which I must keep updated and on a secure network. It is also password protected, and it is my role to protect and regularly change this password. Another part of the security policy is to never allow devices to be plugged into my laptops USB ports. This is something I must adhere to myself but also be vigilant if I see others doing it. If I leave my laptop unattended, I must lock it first to prevent anyone else accessing it

Health & Safety

The health and safety of staff at work is an important aspect of the business environment. The health and safety policy is also outlined in the employee handbook and I participated in a health and safety induction when I first joined.

I was taught how to ensure fire safety. This involved knowing the location of fire exits and emergency meeting points, which must be used in case of a fire alarm. Another part of fire safety is that all electronic devices must be tested to ensure they are fused and don't pose a fire threat when plugged in. I had to comply with this and ensure any new electronics I got were safety tested first before using them.

Since the nature of my work is computer based, one of the main health concerns is from body and eye strains. I took a training course on the health threats from this and how to prevent them. For prolonged sitting, I leant how to sit with correct posture and ergonomically set up my mouse and keyboard to prevent excess strain on my body. I also learnt some stretches than can be done every now and again to reduce the risk of stiffness. To protect my eyes from excess strain due to looking at the screen I learnt how to properly align my screen and the importance of taking regular breaks to look off into the discant to allow the eyes to relax.

General Data Protection Regulation

The company must ensure that all personal information is handled appropriately and securely. This is governed under the Data Protection Act 2018 [REF], which companies in the UK must adhere to. It is used to govern how the company uses, processes, and stores personal data. As part of the company,

I must understand this regulation and follow to it as well. To learn about General Data Protection Regulation (GDPR) I did a training course which taught me the principles of GDPR as well as my responsibilities. I learnt that personal data must be:

- Processed lawfully, fairly and in a transparent manner,
- Collected for a legitimate reason,
- Limited to what is necessary,
- Accurate,
- Kept in a form which identifies data subjects that no longer need personal data stored,
- Processed securely, and
- The controller shall be responsible for compliance.

These are the 7 principles of GDPR. Since I was not a data controller, and would not be handling personal data, my responsibility was to prevent the breaching of this legislation. I must not access anyone's personal data without the correct authorisation from the data controller. Secondly, if I found personal data accidently, which was not stored securely, or seemed to breach any other of the key principles, I must notify the data controller immediately to resolve the issue.

Professional Development

Participating in group activities inside or outside the working environment that can assist with the development of interpersonal skills

I attended virtual meet-and-greets over Teams. This gave me the chance to meet new joiners like myself. The calls had people from different projects and roles. I discussed the work I was doing, and how I had found my time since joining the company. I asked questions about the projects and work other people were doing, which gave me some useful insights into other areas of the business. I also asked about hobbies and interests and used the meetings to make good connections with my colleagues. I really enjoyed these mini-sessions and it helped my build connections with others in the company.

Undertaking pro bono (unpaid) activities that can help to develop professional skills or offer additional insight into, or understanding of, their working role

During my Apprenticeship, I completed the Inspiring Digital Enterprise Award (iDEA) in my personal time. The course taught me how to work in the digital industry by developing my digital, enterprise and employability skills. The course had numerous challenges and talks for which I worked through. I learnt a wide range of skills useful for digital industries, including cloud computing, cyber security, social media, and use of the internet. I also learnt more widely useful skills for any role including teamwork, leadership and problem solving. Finally, I learn how to use creative tools for my career, including, web design, animations, graphic design, and virtual reality. These skills are extremely transferable and applicable to my job and the course has helped my develop in my role.

Undertaking learning in subjects relevant to, but not directly related to, their role (e.g. foreign language courses, mentoring skills, cultural awareness and diversity training), perhaps through self-study or evening classes

I completed diversity and unconscious bias training during my Apprenticeship. These courses outlined the protected attributes of employees and taught me identify and report any discrimination, which is a very important part of working for the company. This was very useful training to improve my skills and teach me the importance of diversity in the company.

Gaining basic knowledge of the employing organisation, its business, structure, culture, products/services, operations and terminology

I have completed training courses and videos teaching me the values and culture in the company. It taught me how to work with other people effectively and make sure the companies' values were upheld. I also learnt, through training courses what must be avoided at the company; including bribery and bullying. This enabled me to follow the companies values and be a good representative to the company wherever I worked.

Gaining knowledge of IT activities in the employing organisation external to their function

I attended a workshop on Docker. Docker is a tool used at the company. I learnt the uses and advantages of using Docker and how to setup a Docker container and perform basic operations on it. I learnt about how Docker is used for company code repositories and continuous integration tools. I found this very interesting, and it gave me a new perspective on the companies IT activities, and how this interesting tool is used in the company and why it is used.

Exploring a topic that is not part of their normal responsibilities, and presenting findings to colleagues and/or management

I attended a company workshop of LEAN and how it can be used. MY colleagues asked me to complete a series of training material on LEAN and provide recommendations for the best training for other colleagues to try, and what would be most useful to show at the workshop. I spent a couple of days going through various LEAN training on the companies' training libraries. I make notes on how I enjoyed the training, the experience and how much I taught me about the subject. I then presented my recommendations for which training people should do and how this could be integrated into the workshop. The training I had recommended was sent out to everyone attending the workshop and provided a good starting point for people to learn about LEAN.

Attending meetings, seminars and workshops organised by a professional body and reading published material such as journals and web content

I attended a careers fair organised by the University of Bath. This was to help recruit new people into the company. I talked to students from the university and discussed the role I had as a software engineer, the type of work I did at the company and what I did as part of the apprenticeship. I answered any additional questions they had about my role, or the company in general. The event went very well, and I spoke to lots of different people. It was good networking experience for me, and the event was ran smoothly by the University.

Undertaking learning and practice in the techniques of team and collaborative working. Gaining an understanding of the underlying concepts

During some of the training sessions I attended, I participated in teams to complete different tasks. One of these tasks was to create a small game using the React framework. I was in a team of 4. I divided the work up into chunks and was assigned a piece of work to do with the logic of the game. Once I had completed some of the work, I discussed it with other team members who suggested changes and I was able to ask them any questions I had to see if they could help. I did the same for other people's work, offering my feedback and providing assistance whenever I could. I found communicating with the team very easy and we worked very well together and completed the game to a good standard.

Undertaking learning and practice in oral and written communications, including report writing and presentations

I presented a meeting on Machine Learning technologies I had been researching, and wrote a report on the subject. The meeting had around 20 attendees. I firstly gave a presentation on the technologies I had researched and how I thought they could be useful to the company. Then I opened up the meeting to questions and discussions. The meeting lasted around an hour, and it was a very enjoyable experience which helped improve my public speaking skills as well as teaching others about the work I had done.