

Part 1

by Rhysian Jones

FILE	12010_RHYSIAN_JONES_PART_1_159704_1401989844.DOCX		
TIME SUBMITTED	10-JAN-2019 02:42PM (UTC+0000)	WORD COUNT	2036
SUBMISSION ID	98702403	CHARACTER COUNT	11792

PROGRAMMING AND SOFTWARE TESTING

ASSIGNMENT 1

Rhysian Jones – 1809710

DVLA brief outline:

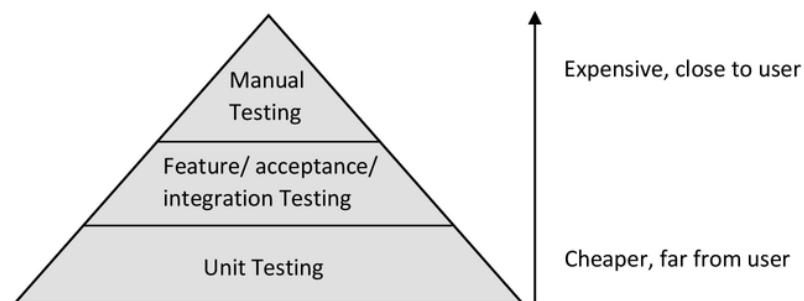
The overall purpose of the teams at the DVLA is to design, create and implement software which has been ordered which will aid in the everyday workings of the DVLA. The teams can work on related or entirely separate projects. For example, one team might be working on creating a new database in order to upgrade from legacy systems while another team might be using java and **JSON to create a web form** for applying for a license and a third team might be creating the front end for that web form using ruby on rails.

Each team will have developers and testers but developers are also testers to a degree. Quality is everyone's responsibility. The DVLA uses a test driven development version called acceptance test driven development. This means that the acceptance test is the thing that drives the whole process so the end result for the end users is the focus.

Additionally, as a government agency, all DVLA web forms and online information must adhere to the guidelines shown on the gov.uk design page^[1]. This means that in order for a front end application to pass the acceptance test, it must fall in line with the gov.uk guidelines.

DVLA specifics:

The DVLA uses an automation test pyramid as shown below:



Unit testing is performed by the development team. It tests the code for each feature as soon as it is completed. The DVLA unit tests at compile and again at branch merge. Unit tests are coded, automated tests.

Integration testing is performed by the development team. It checks that the functions work together and that the structure of the system is correct. Integration tests are coded, automated tests.

Feature testing is a type of acceptance testing which deals with testing each feature.

Acceptance testing is performed by **BAs**. It involves confirming that the product aligns with the user requirements from a user's point of view. It doesn't involve the code at all and is only concerned with functionalities.

Manual testing is performed by a select group of volunteers from the public. The purpose of manual testing is to see how the intended user will use the software. In the DVLA, this is done in the UX lab where members of the public use the software and researchers can track what they do and where their attention is held. Afterwards, the users answer questions about their experience.

As the diagram shows, unit testing is the cheapest and easiest type of testing and is carried out the most. Unit testing catches bugs right at the beginning of the process, saving money and effort in the long run.

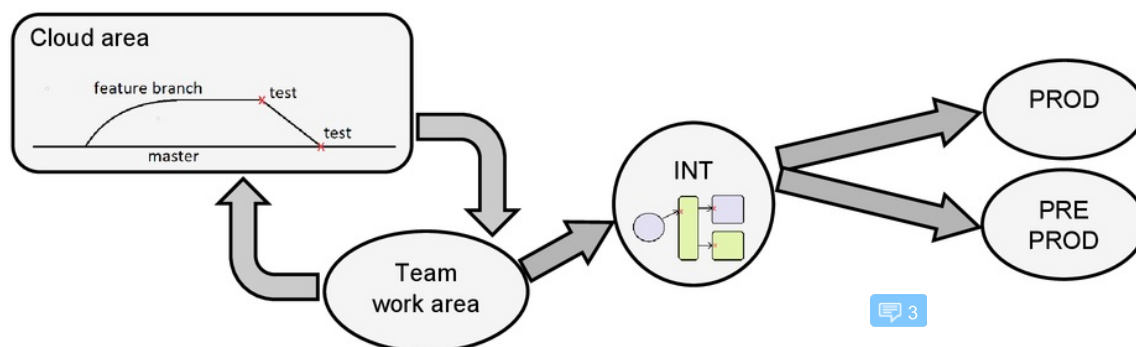
The process of acceptance test driven development is illustrated by the diagram below:



This is similar to the basic test driven development (fail, pass, refactor) but in this case, the process focusses on the acceptance testing. The fail, pass refactor process still occurs for each new feature but the whole system is also tested at the run stage and refactored based on the outcome of acceptance testing.

DVLA testing:

The process of software development at the DVLA can be illustrated by the following diagram:



Each team has its own work area where they are free to set up testing environments as needed. At the start of a new sprint, the teams will write the unit tests that the code must pass usually alongside the code being written but ideally, before coding starts. Junit is used for java code and rspec is used for ruby on rails. The unit tests will be maintained for the lifetime of the system. All documentation and code is static tested by the team to ensure that there is no ambiguity. The acceptance testing is done by the dedicated testers in each squad and is facilitated by the use of 'three amigos' where the developer, tester and business analyst will have a discussion to ensure that everyone understands exactly what is required. The work the teams do is committed to the project repository via a feature branch which is named after the jira ticket of the feature. The new code is unit tested on commit (the first test on the feature branch in the diagram). If this test is passed, the branch can be merged. At merge (the second test) the code is put through regression testing. This can take many hours for a full test so usually, only a partial test is carried out to encourage frequent testing. If this test is passed, the branch is merged with the master.

The cloud area holds all of the repositories for all of the projects. This is where the git repository is held. The team work area is where the team does its work.

The int layer is where integration happens. This is where the whole project is integration tested and the points of integration (red Xs in int layer of diagram) are tested for every function. The DVLA uses a mock live environment for testing. Jenkins is the software behind running the continuous integration testing but Drone is the software that actually does the tests. The integration tests are written in Cucumber.

After integration testing, the project can move into the production and pre-production areas. These areas should be the same. Pre-production is a staging environment used for testing the software as it would happen in a live environment. This is where a large number of tests are carried out: load testing entails running the system at its expected peak load and analysing the resulting metrics from the test, stress testing involves forcing the system to fail to see how it fails (the ideal situation being graceful degradation rather than total failure), soak testing involves running the system for a long time to try to catch any memory leaks or similar problems and performance testing entails using Apache Jmeter to simulate many users and high load and seeing if the desired tests are passed or failed[3].

All of this occurs in three major development stages:

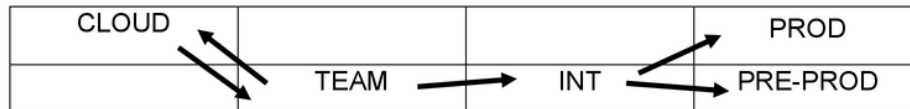
Dev → Test → Live

The dev stage is for developing new code. This could be considered to be the left side of the diagram ending after the int layer. The test stage is for testing the whole project as it would appear in live before going live. This is the pre-production layer. Live is the release of the project. In a traditional software development methodology, this would be the end point but as the DVLA

practices a variation on Scrum, the project can be continually added to for as long as needed, adding new functionalities with every release[4].

Conclusion:

To conclude, the DVLA uses acceptance test driven development in a system which can be described by the outline:



With each area of the system and stage in development containing numerous tests.

References:

- [1] "Home – GOV.UK Design System." [Online]. Available: <https://design-system.service.gov.uk/>. [Accessed: 17-Dec-2018].
- [2] "Weblet Importer." [Online]. Available: <https://reqtest.com/wp-content/uploads/2015/04/atdd.jpg>. [Accessed: 17-Dec-2018].
- [3] "Welcome to Software Testing Fundamentals (STF) ! - Software Testing Fundamentals." [Online]. Available: <http://softwaretestingfundamentals.com/>. [Accessed: 18-Dec-2018].
- [4] "Dev/Test/Live." [Online]. Available: <https://pantheon.io/agencies/development-workflow/dev-test-live-workflow>. [Accessed: 18-Dec-2018].

Bibliography (Code Part 1):

```
package Main;

import java.util.*;
import java.lang.*;

public class AssignmentTask1
{
    public static Scanner input = new Scanner(System.in);
    public static void main(String[] args)
    {
        //VARIABLES:
        int choice;
        double[] arrA;
        double[] arrB;

        //INPUT ARRAYS:
        System.out.println("Welcome. This program will take two arrays of doubles and compare
them as you see fit.");
        arrA = callArray();
        arrB = callArray();

        //SWITCH TO CALL FUNCTIONS:
        do {
            //INTRO + INPUT ARRAYS:
            System.out.println("Please select an option:");
            System.out.println("Select 1 to display your numbers");
            System.out.println("Select 2 to display your minimum and maximum values");
            System.out.println("Select 3 to display the average values of your numbers");
            System.out.println("Select 4 to check for an overlap");
            System.out.println("Select 5 to check for intersection");
            System.out.println("Select 6 to display set difference");
            System.out.println("Select 7 to quit the program");
            System.out.println("Select 8 to reset and reenter your numbers");

            choice = Integer.parseInt(input.nextLine());

            switch (choice)
            {
                case 1: //DISPLAY NUMBERS:

                    displayArray(arrA, arrB);
                    break;
```

```

        case 2: //MIN AND MAX:

            minMax(arrA, arrB);
            break;

        case 3: //AVERAGE:

            displayAv(arrA, arrB);
            break;

        case 4: //OVERLAP:

            overlap(arrA, arrB);
            break;

        case 5: //INTERSECTION:

            intersect(arrA, arrB);
            break;

        case 6: //SET DIFFERENCE:

            setDiff(arrA, arrB);
            break;

        case 8: //RESET NUMBERS:

            arrA = callArray();
            arrB = callArray();
            break;
    }
    } while (choice != 7);
    input.close();
}
//-----END MAIN

//-----FUNCTION CALL ARRAY
public static double[] callArray()
{
    //FUNCTION VARIABLES:
    int X=0;

```

```

//TRY CATCH FOR EXCEPTIONS (NON-INTEGER):
boolean done1 = false;
while (!done1)
{
    try
    {
        System.out.println("Please input your array length:");
        X = Integer.parseInt(input.nextLine());
        done1 = true;
    }
    catch (NegativeArraySizeException e)
    {
        System.out.println("Error found. Out of bounds.");
        done1 = false;
    }
    catch (Exception e)
    {
        System.out.println("Error found. Non-integer.");
        done1 = false;
    }
}
double[] userArray = new double[X];
System.out.println("Please input your array numbers:");
//TRY CATCH FOR EXCEPTIONS (NON-DOUBLES):
boolean done2 = false;
while (!done2)
{
    try
    {
        //OUTER LOOP INPUTS NUMBERS:
        for (int i=0; i<X; i++)
        {
            System.out.println("Number " + (i+1) + ":");
            userArray[i] = Double.parseDouble(input.nextLine());
            //INNER LOOP REJECTS DUPES:
            for (int j=0; j<i; j++)
            {
                if (userArray[i]==userArray[j])
                {
                    i--;
                }
            }
        }
        done2 = true;
    }
}

```

```

        }
        catch (Exception e)
        {
            System.out.println("Error found. Non-double.");
            done2 = false;
        }
    }
    return userArray;
}
//-----END FUNCTION CALL ARRAY

//-----FUNCTION DISPLAY ARRAY
public static void displayArray(double[] arrA, double[] arrB)
{
    System.out.println("Array A: " + Arrays.toString(arrA));
    System.out.println("Array B: " + Arrays.toString(arrB));
    System.out.println();
}
//-----END FUNCTION DISPLAY ARRAY

//-----FUNCTION MINMAX
public static void minMax(double[] arrA, double[] arrB)
{
    //FUNCTION VARIABLES:
    double maxA=-1000;
    double maxB=-1000;
    double minA=1000;
    double minB=1000;

    //MAX ARRAY A:
    for (int i=0; i<arrA.length; i++)
    {
        if (arrA[i] > maxA)
        {
            maxA = arrA[i];
        }
    }
    //MAX ARRAY B:
    for (int i=0; i<arrB.length; i++)
    {
        if (arrB[i] > maxB)
        {

```

```

        maxB = arrB[i];
    }

}

//MIN ARRAY A:
for (int i=0; i<arrA.length; i++)
{
    if (arrA[i] < minA)
    {
        minA = arrA[i];
    }
}

//MIN ARRAY B:
for (int i=0; i<arrB.length; i++)
{
    if (arrB[i] < minB)
    {
        minB = arrB[i];
    }
}

System.out.println("Maximum value of A: " + maxA);
System.out.println("Minimum value of A: " + minA);
System.out.println("Maximum value of B: " + maxB);
System.out.println("Minimum value of B: " + minB);
System.out.println();
}

//-----END FUNCTION MINMAX

//-----FUNCTION DISPLAY AVERAGE
public static void displayAv(double[] arrA, double[] arrB)
{
    //FUNCTION VARIABLES:
    double sum=0;
    double meanA, meanB;

    //ARRAY A SUM:
    for (int i=0; i<arrA.length; i++)
    {
        sum += arrA[i];
    }
    //ARRAY A MEAN:
    meanA = sum/arrA.length;
    //ARRAY B SUM

```

```

        for (int i=0; i<arrB.length; i++)
        {
            sum += arrB[i];
        }
        //ARRAY B MEAN:
        meanB = sum/arrB.length;

        System.out.println("Average value for array A: " + meanA);
        System.out.println("Average value for array B: " + meanB);
        System.out.println();
    }
    //-----END FUNCTION DISPLAY AVERAGE

//-----FUNCTION OVERLAP
public static void overlap(double[] arrA, double[] arrB)
{
    //FUNCTION VARIABLES:
    double minA, maxA, minB, maxB;

    //SORT ARRAY THE CHEATY WAY:
    Arrays.sort(arrA);
    Arrays.sort(arrB);
    minA = arrA[0];
    minB = arrB[0];
    maxA = arrA[arrA.length-1];
    maxB = arrB[arrB.length-1];

    //FIND OVERLAPS:
    if ((minA > minB && minA < maxB) || (minB > minA && minB < maxA))
    {
        System.out.println("Your arrays have some overlap");
        System.out.println();
    }
    else
    {
        System.out.println("Your arrays have no overlap");
        System.out.println();
    }
}
//-----END FUNCTION OVERLAP

//-----FUNCTION INTERSECTION

```

```

public static void intersect(double[] arrA, double[] arrB)
{
    //FUNCTION VARIABLES:
    List<Double> dupes = new ArrayList<Double>();

    System.out.println("The values which appear in both of your arrays are: ");
    //LOOP THROUGH BOTH ARRAYS TO FIND MATCHES:
    for (int i = 0; i < arrA.length; i++)
    {
        for (int j = 0; j < arrB.length; j++)
        {
            if (arrA[i] == arrB[j])
            {
                dupes.add(arrA[i]);
                //System.out.println("Match: " + arrA[i]);
            }
        }
    }
    System.out.println("Duplicates: " + dupes);
    System.out.println();
}

//-----END FUNCTION INTERSECTION

//-----FUNCTION SET DIFFERENCE
public static void setDiff(double[] arrA, double[] arrB)
{
    //FUNCTION VARIABLES:
    List<Double> dupes = new ArrayList<Double>();
    List<Double> uniques = new ArrayList<Double>();

    System.out.println("The values which appear in just one of your arrays are: ");
    //LOOP THROUGH BOTH ARRAYS TO FIND MATCHES:
    for (int i = 0; i < arrA.length; i++)
    {
        for (int j = 0; j < arrB.length; j++)
        {
            if (arrA[i] == arrB[j])
            {
                dupes.add(arrA[i]);
            }
        }
    }
    //MAKE LIST OF BOTH ARRAYS:

```

```
        for (int i=0; i<arrA.length; i++)
        {
            uniques.add(arrA[i]);
        }
        for (int i=0; i<arrB.length; i++)
        {
            uniques.add(arrB[i]);
        }
        uniques.removeAll(dupes);
        System.out.println("Uniques: " + uniques);
        System.out.println();
    }
    //-----END FUNCTION SET DIFFERENCE

}
```

Part 1

GRADEMARK REPORT

FINAL GRADE

77/100

GENERAL COMMENTS

Instructor

PART ONE

=====

Engagement with Literature Skills (25) **22**

Knowledge and Understanding Skills (10) **9**

Cognitive and Intellectual Skills (10) **8**

Transferable Skills (5) **3**

Practical Application Skills (50) **35**

Part 1 Software Testing

A good explanation of the testing process at the DVLA, although some parts were a little unclear. I would like to have seen the technologies used addressed, but you went the other route of making the process used as clear as possible. I found your description clearer, so will give you a few more marks.

Code Part1 Number Sets

a. Display numbers in sets COMPLETE

b. Determine max and min COMPLETE

c. Calculate average INCORRECT

d. Determine overlap COMPLETE

e: Intersection COMPLETE f:

Set difference COMPLETE

Use of JUnit NO

Use of Git YES Use of Java Doc NO

Average is incorrect, was correct for first set.
Trying testing more than one set of numbers, you
need to zero the total. Should have been picked up
in testing!!

PART TWO

=====

Engagement with Literature Skills (25) **12**

Knowledge and Understanding Skills (10) **5**

Cognitive and Intellectual Skills (10) **5**

Transferable Skills (5) **4**

Practical Application Skills (50) **35**

Part 2 Design Principles

Far too little detail in this report - you're a long way
from the word count with standard stuff
(references, titles, code, figures etc.) removed.

Only 4 references, two of which are pretty low
academic credibility.

What's there is of a good standard - there's just far
too little of it.

Code Part 2 Connect 4

Display board COMPLETE

Enter column COMPLETE

Check column position is empty COMPLETE

Drop counter to position COMPLETE

Check win/lose COMPLETE

Blitz feature INCOMPLETE

Time bomb feature COMPLETE

Use of JUnit NO Use of Git YES with branches

Use of Java Doc NO

Blitz functionality left one counter at top row preventing the column being used. Usability could have been improved.

Overall Comments

A number of Git commits made and some branching. No unit testing, but you have provided test files where you have built your own experimental test code and a test plan.

Marks Part One: **77**

Marks Part Two: **61**



Comment 1

Are you sure?



Comment 2

Define this acronym before using it



Comment 3

Nice diagram