Refactored WFG Algorithm – Test Plan (A)

The purpose of this document is to outline the testing strategy and overall approach for the WFG Hypervolume project. This includes test case derivation techniques, both fundamental approaches (functional and structural), with detail explanations to why the chosen methods were selected.

The focus of the test cases is to evaluate the compliance of the entire system, giving a sense of validation that every unit of the project is performing to the provided specifications. The cases are covering the performance and reliability of the system, this includes recording user interaction, data output and the speed in which the program performs the task.

The following testing techniques that will be used in this task are:

- *Equivalence Partitioning*
- *Boundary Value Analysis*
- *Decision Testing*
- *Error Guessing*

The first action with selecting feasible techniques in this test plan is to evaluate the system based on the program's complexity. The conclusion is that using Functional Testing in this project is the most optimal way of discovering bugs. The most widely used and effective Functional Testing techniques includes using Equivalence Partitioning (EP) (*Thomas, 2018*) and Boundary Value Analysis (BVA) together.

EP is a very favourable technique as it most divides up the system into smaller partitions, allowing the number of test cases to diminish without compromising the test quality or coverage in the system (*Rajkumar, 2018*). This technique also reduces the testing time of a software as checking every possible number in the partitions would be time-consuming (*Deriskqa, 2018*).
Another reason why EP is the best test technique for this distinct task is because of experience using this testing method, making it a more accessible approach to find errors as oppose to alternative functional testing strategy.

An efficient technique that will be used in conjunction with EP is the use of BVA. The behaviour at each of the partitions could potentially have an incorrect behaviour, which would yield possible defects (*Rajkumar, 2018*). It's also designed to cover valid and invalid boundary values and applied at all testing levels, whilst being computationally and theoretically inexpensive to create test cases.

The use of EP and BVA produces a large test coverage in the system. However, singular errors could potentially be still in the program and left unnoticed. Those tests are found via Error Guessing (EG), where using developer instincts will identify potential discrepancies. Learning from past work made it easier to analyse the program and reveal bugs. Advice from Shi (*p.51*) claiming that using EP, BVA and EG as the main testing tactics. They're also proven testing methodologies for *"…deriving effective and economic test cases in the business practice".* This was one of the main reasons why such functional tests were chosen.

Finally, Structural Testing will be practiced which complements the existing Functional Testing techniques. The decision to include Structural Testing was heavily influenced by research which show that there is as much as 70% of all the code making up the system that might never be executed at all in the testing phase (*Mitchell, J. and Black, R., 2015*).

To gain a better verification as to what the written code is expected to accomplish, Decision Testing (DT) will be performed. This will check the flow of the different paths in the program, ensuring all paths are tested at some point.

DT will also have the capability to check which blocks of code have been executed in each decision and which have not. This will help with code coverage very efficiently. The DT coverage will be calculated to find out how much of the program has been covered in each decision with this approach. The higher the DT coverage is, the greater the quality of the written code will be.

Word Count: 617

References

Thomas, T., 2018. Purpose of Equivalence Partitioning in testing [online]. Zyxware Technologies. Available from: https://www.zyxware.com/articles/3766/purpose-of-equivalence-partitioning-in-testing [Accessed 2 Nov 2018].

Rajkumar, 2018. Equivalence Partitioning Test Case Design Technique [online]. Software Testing Material. Available from: https://www.softwaretestingmaterial.com/equivalence-partitioning-testing-technique/ [Accessed 2 Nov 2018].

Deriskqa, 2018. Test Design Techniques: Equivalence Partitioning | DeRisk QA [online]. Deriskqa.com. Available from: http://www.deriskqa.com/Article-Equivalence-Partitioning.html [Accessed 2 Nov 2018].

Rajkumar, 2018. Boundary Value Analysis Test Case Design Technique [online]. Software Testing Material. Available from: https://www.softwaretestingmaterial.com/boundary-value-analysis-testing-technique/ [Accessed 2 Nov 2018].

Shi, M., 2010. Software Functional Testing from the Perspective of Business Practice [online]. Pdfs.semanticscholar.org. Available from: https://pdfs.semanticscholar.org/6c2e/33324e97d12825201cd77b7d23738ca18f35.pdf [Accessed 3 Nov 2018].

Mitchell, J. and Black, R., 2015. Advanced Software Testing: Volume 3 [online]. Rbcs-us.com. Available from: https://rbcs-us.com/site/assets/files/1194/structural-testing-techniques.pdf [Accessed 3 Nov 2018].

Functional Testing

Equivalence Partitioning (EP) table identifying the valid and invalid partitions.

EP Class Description	EP Class ID	Valid/Invalid Partition	Partitions Description	Additional Information
Data Point(s) (front.txt file)	**01**	Valid	All Data Points contain the **same dimensions** as the reference point	Positive values only
	02	Invalid	All data points **don't** have the same dimensions.	All data points are a positive value – Reference Point greater value than the data points
	03	Valid	All Data Points are **less than/equal** to the reference point	N/A
	04	Invalid	All Data Points **greater** than the reference point	Reference Point should be dominating the data points
Front.txt Missing Credentials	**05**	Valid	Fronts **separated correctly** with hashes	As described in the system requirements
	06	Invalid	Hashes **not separated correctly** in the Front.txt	As described in the system requirements
	07	Invalid	**No** data points	Still contains both hashes in the Front.txt file
	08	Invalid	**Empty** Front.txt	Doesn't contain anything in the Front.txt file
	09	Invalid	**No Front.txt** file available	N/A
Invalid Credentials	**10**	Invalid	Data Point - **Invalid Inputs**	Strings, characters, Boolean etc.
	11	Invalid	Reference Point containing **invalid** inputs	Non-numerical values (String, characters)

Test Cases derived from the WFG Hypervolume program using **Equivalence Partitioning (EP)** and **Boundary Value Analysis (BVA)** techniques.

No.	Test	Additional Information	Testing Technique	Input	Expected Output
01	Testing **1-Dimensional** Hypervolume	Front.txt only **one dimension** only. Reference Point **greater than** the data points.	Equivalence Partitioning	Front.txt file: *#* *0.1* *0.2* *0.3* *#* Input command "*wfg Front.txt*"	Measuring the length of a line – **No error occurring**
02	Testing **2-Dimensional** Hypervolume EP Class ID: 1 Data Point/Reference have the **same dimensions**	Front.txt **two dimensions** only. Reference Point **greater than** the data points. Two tests at once.	Equivalence Partitioning	Front.txt file: *#* *1 1* *2 2* *3 2* *#* Input command "*wfg Front.txt 5 5*"	Outputs successfully
03	Testing **3-Dimensional** Hypervolume Partition EP Class ID: 3 **Data Points <= Reference point**	Front.txt **three dimensions** only. Reference Point **greater than** the data points. Two tests at once.	Equivalence Partitioning	Front.txt file: *#* *1 1 1* *2 2 2* *3 2 2* *#* Input command "*wfg Front.txt 5 5 5*"	Outputs successfully
04	EP Class ID: 2 Data Point/Reference **don't have the same dimensions**	Data Points – Positive Value Reference Point **>** Data Points	Equivalence Partitioning	Front.txt file: *#* *1 1 1* *2 2* *3* *#* Input command "*wfg Front.txt 5 5*"	Error Message – Runs the program unsuccessfully

05	EP Class ID: 4 **Data Points > Reference point**	*N/A*	Equivalence Partitioning	Front.txt file: *#* *2 2* *2 3* *3 4* *#* Using command "*wfg Front.txt 1 1*"	Error Message – Runs the program unsuccessfully
06	EP Class ID: 5 Fronts **separated correctly with hashes**	Meeting the correct requirements in the brief	Equivalence Partitioning	Front.txt file: *#* *1 1* *2 2* Input command "*wfg Front.txt 5 5*"	Outputs successfully
07	EP Class ID: 6 **Hashes not separated correctly**	Meeting the correct requirements in the brief – Fidelity in the requirements	Equivalence Partitioning	Empty Front.txt file Input command "*wfg Front.txt 5 5*"	Error Message – Runs the program unsuccessfully
08	EP Class ID: 7 **No data points**	Still contains the Front.txt file and "#". Contains no data inside the front though.	Equivalence Partitioning	Front.txt file: *#* *#* Input command "*wfg Front.txt 5 5*"	Error Message – Runs the program unsuccessfully
09	EP Class ID: 8 **Empty Front.txt**	Still contains the Front.txt file, but not the "#". Empty .txt file.	Equivalence Partitioning	*No Front.txt file* Input command "*wfg Front.txt 5 5*"	Error Message – Runs the program unsuccessfully
10	EP Class ID: 9 **No Front.txt file** available	Program can't find the Front.txt file	Equivalence Partitioning	Front.txt file: *#* *1 2* *2 B* *#* Input command "*wfg Front.txt 5 5*"	Error Message – Runs the program unsuccessfully

11	EP Class ID: 10 **Invalid Data Points Inputs**	Invalid characters entered in the commands.	Equivalence Partitioning	Front.txt file: *#* *1 2* *2 3* *#* Input command "*wfg Front.txt 5 B*"	Error Message – Runs the program unsuccessfully

Test Cases derived from the WFG Hypervolume program using **Error Guessing (EG)** techniques

No.	Test	Additional Information	Testing Technique	Input	Expected Output
12	Test containing **more than one front** in the Front.txt file	Making sure that the program is working with more than one front	Error Guessing	Front.txt file: *#* *1 2* *2 3* *#* *2 2* *3 2* *#* Input command "*wfg Front.txt 5 5*"	Successful output of Hypervolume each front.
13	Test containing a **space after a hash** in the Front.txt file	Test to see if spaces affect the Front.txt file in a negative way. Error message should appear based on experience.	Error Guessing	Front.txt file: #[SPACE] 1 1 2 2 # Input command "*wfg Front.txt 5 5*"	Error Message – Runs the program unsuccessfully
14	Test containing a **space before the hash** in the Front.txt file	Test to see if spaces affect the Front.txt file in a negative way. Error message should appear based on experience.	Error Guessing	Front.txt file: *[SPACE]#* *1 1* *2 2* *#* Input command "*wfg Front.txt 5 5*"	Error Message – Runs the program unsuccessfully
15	All data in the Front.txt file to **appear in the first line** of the text file	Test to see if the program is meeting the requirements provided as each data point should be in a separate line.	Error Guessing	Front.txt file: *#1 1 2 2#* Input command "*wfg Front.txt 5 5*"	Error Message – Runs the program unsuccessfully
16	Front.txt contains a non-numerical symbol (**£, $, %, &, *, ~**) instead of a hash symbol	Checking if the Requirements are met as "#" should be in between each front	Error Guessing	Front.txt file: *%* *1 1* *2 2* *%* Input command "*wfg Front.txt 5 5*"	Error Message – Runs the program unsuccessfully

Decision Testing

Decision Testing (DT) table identifying the true/false decisions

DT ID	DT Process	DT Statement Test	DT Coverage
1	Reference Point Input	Providing **no Reference Point**	(5/5) * 100 = *100% Coverage*
		Providing the **Incorrect Reference Point** dimensions	
		Providing the **Correct Reference Point** dimensions	
2	Memory Allocation Input	Many **fronts**	(10/15) * 100 = *75% Coverage*
		Many **data points**	

Test Cases derived from the WFG Hypervolume program using **Decision Testing (ST)** techniques.

No.	Test	Additional Information	Testing Technique	Input	Expected Output
17	DT ID: 1 **No Reference Point** is provided	According to the code – Should reveal a message *"No reference point provided: using the origin"*	Statement Testing	Front.txt file: *#* *1 2* *2 3* *#* Input command "*wfg Front.txt*"	Unsuccessful output – Error message appearing – **Defaults to origin**
18	DT ID: 1 **Wrong Reference Point dimensions** is provided	According to the code **–** Should reveal a message *"Your reference point should have x values"* – *"x"* being the number of dimensions in the Front.txt **Similar test was made in Test No. 05**. This is considering the code in the **wfgkw.c file** and each line of code in detail.	Statement Testing	Front.txt file: *#* *1 2* *2 3* *#* Input command "*wfg Front.txt 5 5 5*"	**Unsuccessful output** – Error message appearing
19	DT ID: 1 Correct **Reference Point dimensions** is provided	**Similar test was made in Test No. 02**. This is considering the code in the **wfgkw.c file**. Analysing each line of code in greater detail	Statement Testing	Front.txt file: *#* *1 2* *2 3* *#* Input command "*wfg Front.txt 5 5*"	**Successful test**
20	DT ID: 2 Checking how the program handles a **larger subset of data points** in a single process	Checking if **data memory errors** occur when many data points is in the Front.txt file.	Statement Testing	Front.txt file: *#* *1 2* *2 3* *2 2* *3 2* *3 4* *5 5* *4 5* *6 6* *5 6* *8 8* *6 7* *#* Input command "*wfg Front.txt 15 15*"	**Successful test**

21	DT ID: 2 Checking how the program handles many **fronts** in a single process	Checking if data memory errors occur when many fronts is in the Front.txt file.	Statement Testing	Front.txt file: *#* *1 2* *2 3* *#* *2 2* *3 2* *#* *3 4* *5 5* *#* *4 5* *6 6* *#* *5 6* *8 8* *#* *9 7* *6 7* *#* Input command “*wfg Front.txt 15 15*“	**Successful test**
22	**Removing the Front.txt** file during while the program is running	Can add many fronts in the Front.txt file to get more time to remove the Front.txt After examining the code and how it reads the Front.txt file. No error messages available in the program if an error occurs during the process of reading the file	Statement Testing	Front.txt file: *#* *1 2* *2 3* *#* *2 2* *3 2* *#* *3 4* *5 5* *#* *4 5* *6 6* *#* *5 6* *8 8* *#* *9 7* *6 7* *#* Input command “*wfg Front.txt 15 15*“ **Then remove the Front.txt file while the program is running.**	**Error occur**/crash program

Assumptions

Expectations of test cases for the **entire process** provided in the table below.

No.	Description	Assumption Fundamentals	Actual Assumptions
1	**New line format** in the Front.txt file for data points	The specification provided indicates each dimension is **separated by a new line**. But information is not available while running the program and, in the code, provided.	If the **specifications are correct**. Therefore, data points will always be in each line.
2	**Spaces between each data value** in Front.txt	Spaces are separating each data value – each data point. This has been shown in the specifications. While running the code or examining the code, no information regarding it.	Assume that the **specifications are correct**. Data values separated with a space.
3	"**#**" in the Front.txt	Specifications provided show that each front is separated by a "#". Nothing in the	Assume that the **specifications are right** and the **"#" is required** to separate the fronts in the file.
4	**Cygwin1.dll** file	Specifications doesn't describe the fact that the cygwin1.dll file is a requirement in the program. However, running the code produces issues without it.	Assume that **the program requires the file** for functional testing to work as expected.