# Part 3 – Theoretical Foundations

## Control Flow Graph

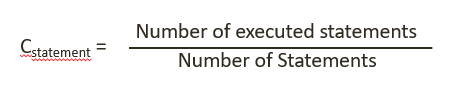
A Control Flow Graph (CFG) is a directed graph that models a single procedure. Within a CFG, nodes represent regions of the code, while the directed edges represent possible branches of execution. The given code snippet was overserved and the following CFG was created.

A close up of a logo

Description automatically generated

## Test Suite with 100 % Statement Coverage.

Statement coverage is define as :



Therefore to achieve 100 % statement overage, all the nodes need to be visited at least once.

This can be done by running a test containing the flowing passed parameters:

|  |  |
| --- | --- |
| Parameter | Content |
| text | “test” |
| Set happy | [“test”] |
| Set unhappy | [“test”] |

Within one loop of execution, each node in the CFG will be executed, but not each branch will be executed, and hence branch coverage will not be 100%. Note that in the nodes D and E, the false branches will not be tested with this case.

## Test Suite with 100 % Statement and Branch Coverage.

This can be achieved with the following test cases

Test One:

|  |  |
| --- | --- |
| Parameter | Content |
| text | “test” |
| Set happy | [“test”] |
| Set unhappy | [“test”] |

Test Two:

|  |  |
| --- | --- |
| Parameter | Content |
| text | “test” |
| Set happy | [“NOTTest”] |
| Set unhappy | [“NOTTest”] |

With the first test case, as mentioned before, all the branches will be visited except the false branches of the D and E nodes. These branches on the other hand and now visited in test case 2, and all branches are covered, achieving 100 % branch coverage.

Another possible testing suit can include one test, with the sets of Happy and Unhappy stated above, joined into one each as follows.

Alternative test

|  |  |
| --- | --- |
| Parameter | Content |
| text | “test” |
| Set happy | [“NOTTest”, “test”] |
| Set unhappy | [“NOTTest”, “test”] |

## Paths for 100% boundary Interior path coverage

Since the number of possible paths can be unbounded, different testing criteria can be set to limit the number of paths tested. To achieve 100 % boundary interior path coverage, each path differing by a subpath will be visited. Note that the number of subpaths can still grow exponentially.

Paths to achieve 100% boundary interior coverage are:

* A -> B -> H
* A -> B -> C -> D -> E -> B-> H
* A -> B -> C -> D -> E-> G-> B-> H
* A -> B -> C -> D -> F-> E-> G ->B ->H
* A -> B -> C -> D -> F-> E ->B ->H

## Data Flow Graph

A Data Flow Graph (DFG) links the relationships between def-use-pairs. These are pairs of points where data is defined and used respectively.

A close up of a logo

Description automatically generated

## Data Flow adequacy

The minimal test suit in Q2 does not full fill the definition clear path criteria since there is no definition clear path since the flag will be reassigned between the nodes F and G.

## Static Code analysis

Static code analysis is a type of analysis that can be performed without running the system, and aims to find faults, ie incorrect step, process or data definition.This can be done in the early stages of the system design, preventing potential bugs to develop further down the line

Static Code analysis be used to compliment/ support dynamic code analysis. This is often required as static testing offers no insight to the intent of the program.

## Issues detected

After reviewing possible manual static code methods such as Requirements Testing, Buddy Checking / Peer Review, Code walk-throughs, it was noted that given the task at hand, not all were possible. For instance the no formal requirements were provided so making Requirement testing impossible. The Buddy checking approach was also not adopted since the work assigned was required on an individual bases. For this reason, a walk though of both the CFG and the DFG was manually carried out, and potential faults were identified. These included:

* The last print statement will never be executed since it occurs after the return statement.
* The **flag** Boolean is never made use of, but is assigned.
* The **word**  string is redefined each iteration, instead of be just reassigned.