Homework #5
CSE 566: Software Project/ Process/ Quality Management
(2015 spring)

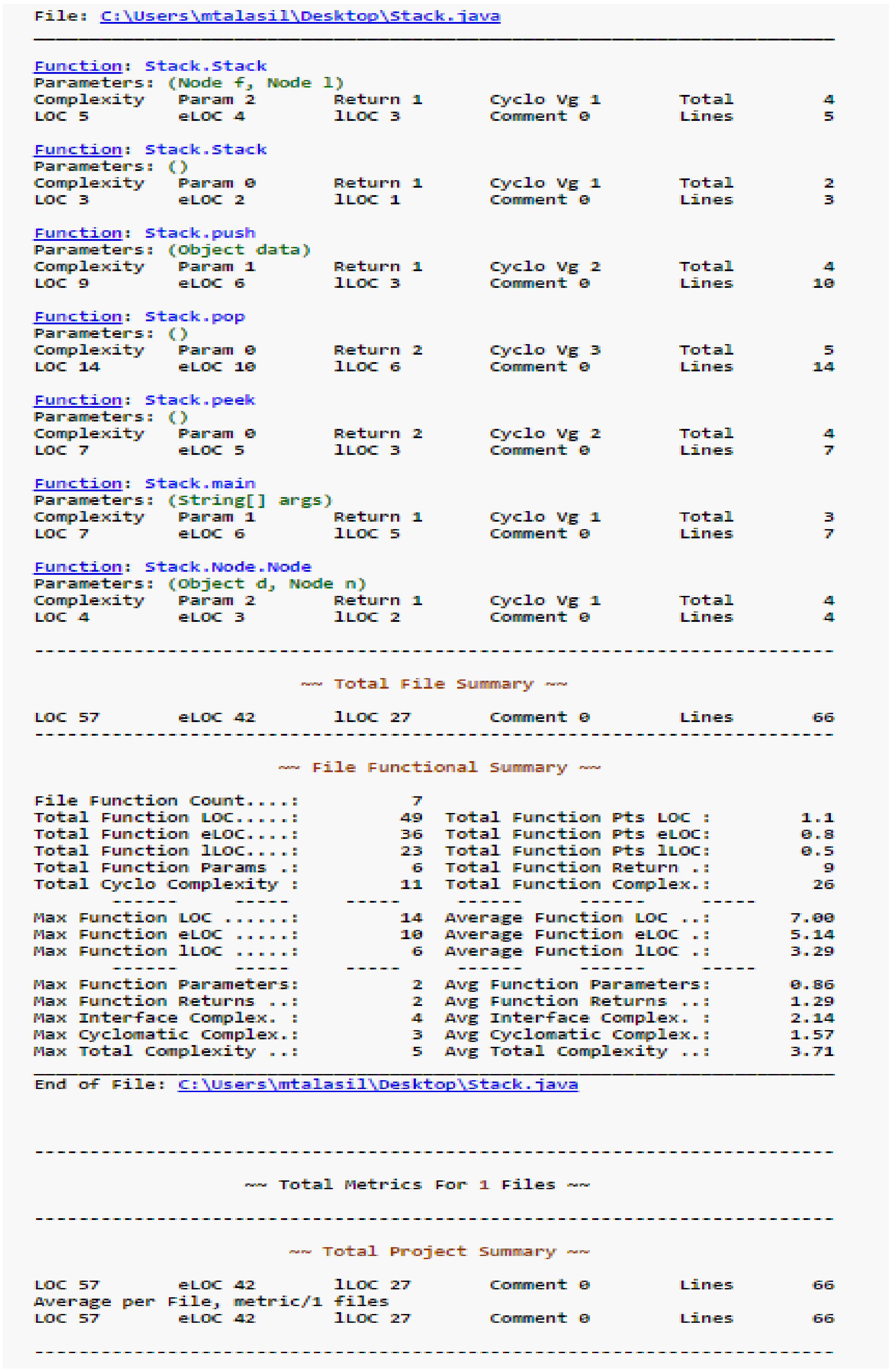
Submitted By:

Madhu Meghana Talasila

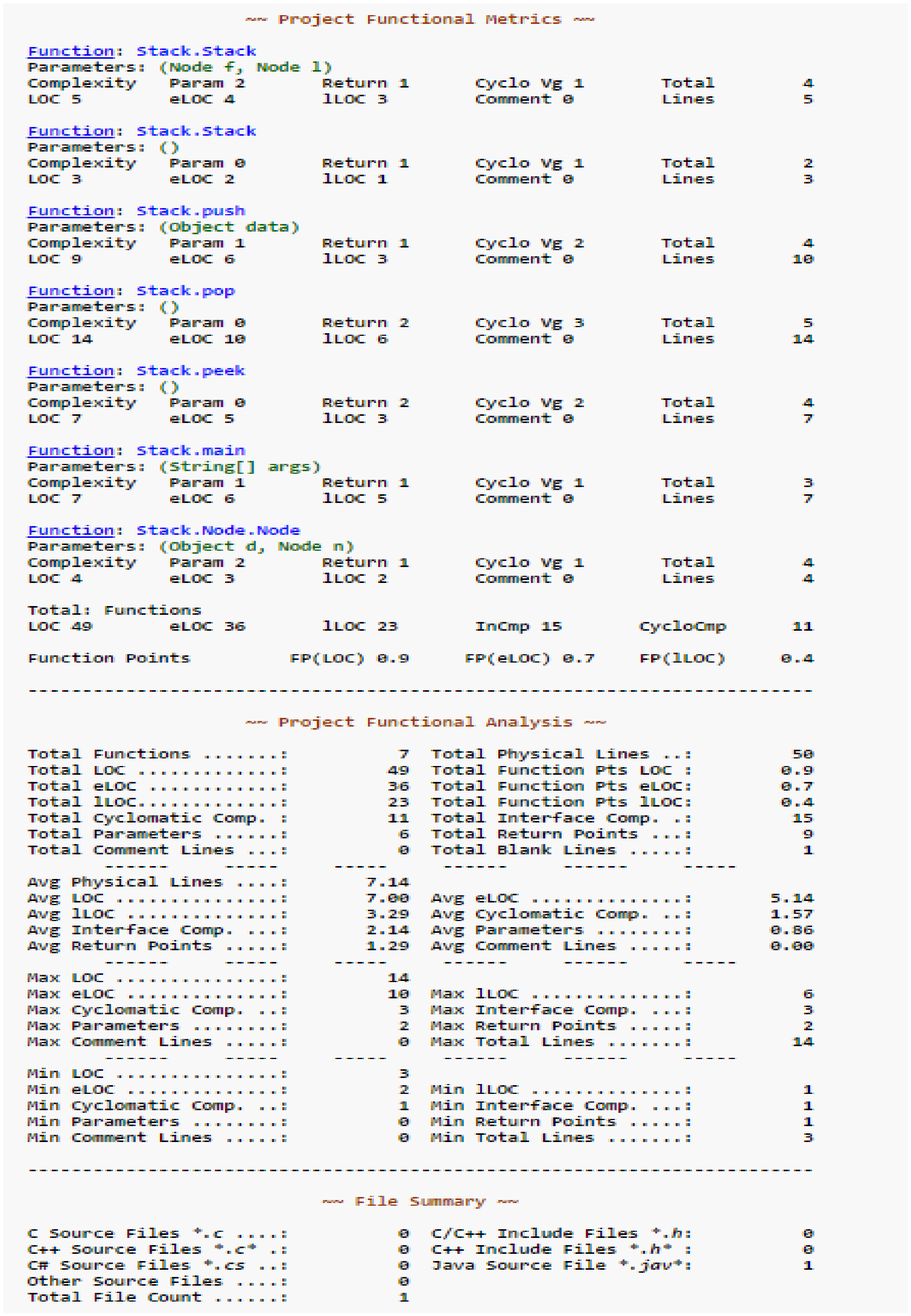
(1207740881)

<u>Part 1:</u> Analyzing quality of software using Resource Standard Metrics Tool

Complexity Metrics, Cyclomatic, Interface, Total (HTML): This metric deals with functional metric and Complexity Analysis



```
File: C:\Users\mtalasil\Desktop\Stack.java
________________________________________________________________________

Function: Stack.Stack
Parameters: (Node f, Node l)
Complexity   Param 2       Return 1      Cyclo Vg 1       Total        4
LOC 5        eLOC 4        lLOC 3        Comment 0        Lines        5

Function: Stack.Stack
Parameters: ()
Complexity   Param 0       Return 1      Cyclo Vg 1       Total        2
LOC 3        eLOC 2        lLOC 1        Comment 0        Lines        3

Function: Stack.push
Parameters: (Object data)
Complexity   Param 1       Return 1      Cyclo Vg 2       Total        4
LOC 9        eLOC 6        lLOC 3        Comment 0        Lines       10

Function: Stack.pop
Parameters: ()
Complexity   Param 0       Return 2      Cyclo Vg 3       Total        5
LOC 14       eLOC 10       lLOC 6        Comment 0        Lines       14

Function: Stack.peek
Parameters: ()
Complexity   Param 0       Return 2      Cyclo Vg 2       Total        4
LOC 7        eLOC 5        lLOC 3        Comment 0        Lines        7

Function: Stack.main
Parameters: (String[] args)
Complexity   Param 1       Return 1      Cyclo Vg 1       Total        3
LOC 7        eLOC 6        lLOC 5        Comment 0        Lines        7

Function: Stack.Node.Node
Parameters: (Object d, Node n)
Complexity   Param 2       Return 1      Cyclo Vg 1       Total        4
LOC 4        eLOC 3        lLOC 2        Comment 0        Lines        4

------------------------------------------------------------------------

                        ~~ Total File Summary ~~

LOC 57       eLOC 42       lLOC 27       Comment 0        Lines       66
------------------------------------------------------------------------

                      ~~ File Functional Summary ~~

File Function Count....:         7
Total Function LOC.....:        49   Total Function Pts LOC :       1.1
Total Function eLOC....:        36   Total Function Pts eLOC:       0.8
Total Function lLOC....:        23   Total Function Pts lLOC:       0.5
Total Function Params .:         6   Total Function Return .:         9
Total Cyclo Complexity :        11   Total Function Complex.:        26
       ------     -----     -----     ------     ------     -----
Max Function LOC ......:        14   Average Function LOC ..:      7.00
Max Function eLOC .....:        10   Average Function eLOC .:      5.14
Max Function lLOC .....:         6   Average Function lLOC .:      3.29
       ------     -----     -----     ------     ------     -----
Max Function Parameters:         2   Avg Function Parameters:      0.86
Max Function Returns ..:         2   Avg Function Returns ..:      1.29
Max Interface Complex. :         4   Avg Interface Complex. :      2.14
Max Cyclomatic Complex.:         3   Avg Cyclomatic Complex.:      1.57
Max Total Complexity ..:         5   Avg Total Complexity ..:      3.71
________________________________________________________________________
End of File: C:\Users\mtalasil\Desktop\Stack.java


------------------------------------------------------------------------

                   ~~ Total Metrics For 1 Files ~~

------------------------------------------------------------------------

                      ~~ Total Project Summary ~~

LOC 57       eLOC 42       lLOC 27       Comment 0        Lines       66
Average per File, metric/1 files
LOC 57       eLOC 42       lLOC 27       Comment 0        Lines       66

------------------------------------------------------------------------
```



```
                       ~~ Project Functional Metrics ~~

Function: Stack.Stack
Parameters: (Node f, Node l)
Complexity   Param 2       Return 1       Cyclo Vg 1       Total         4
LOC 5        eLOC 4        lLOC 3         Comment 0        Lines         5

Function: Stack.Stack
Parameters: ()
Complexity   Param 0       Return 1       Cyclo Vg 1       Total         2
LOC 3        eLOC 2        lLOC 1         Comment 0        Lines         3

Function: Stack.push
Parameters: (Object data)
Complexity   Param 1       Return 1       Cyclo Vg 2       Total         4
LOC 9        eLOC 6        lLOC 3         Comment 0        Lines        10

Function: Stack.pop
Parameters: ()
Complexity   Param 0       Return 2       Cyclo Vg 3       Total         5
LOC 14       eLOC 10       lLOC 6         Comment 0        Lines        14

Function: Stack.peek
Parameters: ()
Complexity   Param 0       Return 2       Cyclo Vg 2       Total         4
LOC 7        eLOC 5        lLOC 3         Comment 0        Lines         7

Function: Stack.main
Parameters: (String[] args)
Complexity   Param 1       Return 1       Cyclo Vg 1       Total         3
LOC 7        eLOC 6        lLOC 5         Comment 0        Lines         7

Function: Stack.Node.Node
Parameters: (Object d, Node n)
Complexity   Param 2       Return 1       Cyclo Vg 1       Total         4
LOC 4        eLOC 3        lLOC 2         Comment 0        Lines         4

Total: Functions
LOC 49       eLOC 36       lLOC 23        InCmp 15        CycloCmp      11

Function Points         FP(LOC) 0.9     FP(eLOC) 0.7    FP(lLOC)     0.4

------------------------------------------------------------------------

                       ~~ Project Functional Analysis ~~

Total Functions .......:          7  Total Physical Lines ..:        50
Total LOC .............:         49  Total Function Pts LOC :       0.9
Total eLOC ............:         36  Total Function Pts eLOC:       0.7
Total lLOC.............:         23  Total Function Pts lLOC:       0.4
Total Cyclomatic Comp. :         11  Total Interface Comp. .:        15
Total Parameters ......:          6  Total Return Points ...:         9
Total Comment Lines ...:          0  Total Blank Lines .....:         1
      ------     -----     -----     ------     ------     -----
Avg Physical Lines ....:       7.14
Avg LOC ...............:       7.00  Avg eLOC ..............:      5.14
Avg lLOC ..............:       3.29  Avg Cyclomatic Comp. ..:      1.57
Avg Interface Comp. ...:       2.14  Avg Parameters ........:      0.86
Avg Return Points .....:       1.29  Avg Comment Lines .....:      0.00
      ------     -----     -----     ------     ------     -----
Max LOC ...............:         14
Max eLOC ..............:         10  Max lLOC ..............:         6
Max Cyclomatic Comp. ..:          3  Max Interface Comp. ...:         3
Max Parameters ........:          2  Max Return Points .....:         2
Max Comment Lines .....:          0  Max Total Lines .......:        14
      ------     -----     -----     ------     ------     -----
Min LOC ...............:          3
Min eLOC ..............:          2  Min lLOC ..............:         1
Min Cyclomatic Comp. ..:          1  Min Interface Comp. ...:         1
Min Parameters ........:          0  Min Return Points .....:         1
Min Comment Lines .....:          0  Min Total Lines .......:         3

------------------------------------------------------------------------

                              ~~ File Summary ~~

C Source Files *.c ....:          0  C/C++ Include Files *.h:         0
C++ Source Files *.c* .:          0  C++ Include Files *.h* :         0
C# Source Files *.cs ..:          0  Java Source File *.jav*:         1
Other Source Files ....:          0
Total File Count ......:          1
```

As observed in the above image,

Comments in this file are Zero. So the program that I used has less maintainability. Also Logical lines of code are less than effective lines of code. There might be empty spaces and braces placed for readability. Cyclomatic[1] complexity is very little so there is less branching. Similarly no. of return points, no. of interface components and minimum, average and maximum eLOC, lLOC and LOC are given in project Functional analysis and File Functional analysis.

Now, we will check the Functional Quality Metrics: It deals with Function Metrics, Class/ Struct Metrics, Complexity Analysis, Quality Analysis.

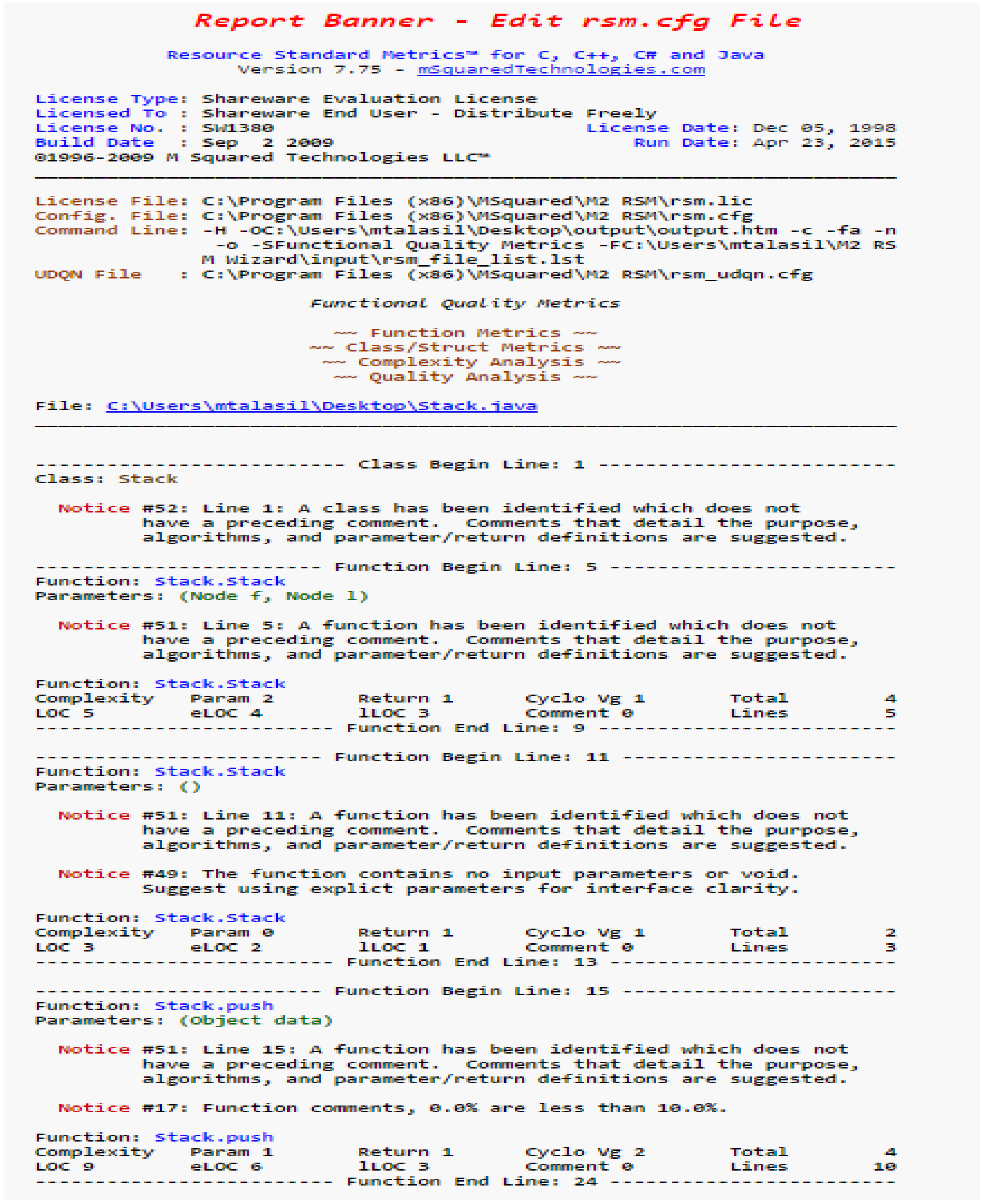
In this each of the class and function is analyzed in function metrics with their corresponding notices, complexity LOC, eLOC, lLOC, comments, lines etc. As mentioned earlier there are no comments in the file so comment count is shown zero. We can check each of the notice and work to ensure the quality of the program.

Then total functional summary and File functional Summary are given. Which analyses the total no. of modifications needed to be done to make the code effective is given.

Class/ Struct Metrics is the difference in thos Functional Quality Metrics, This defines the no. of access modifiers used in the program. No. of public, private, protected class variables, functions, methods are there in the program. Depending on that we can make the changes in the code and we can try to avoid particular access modifier depending on program need.

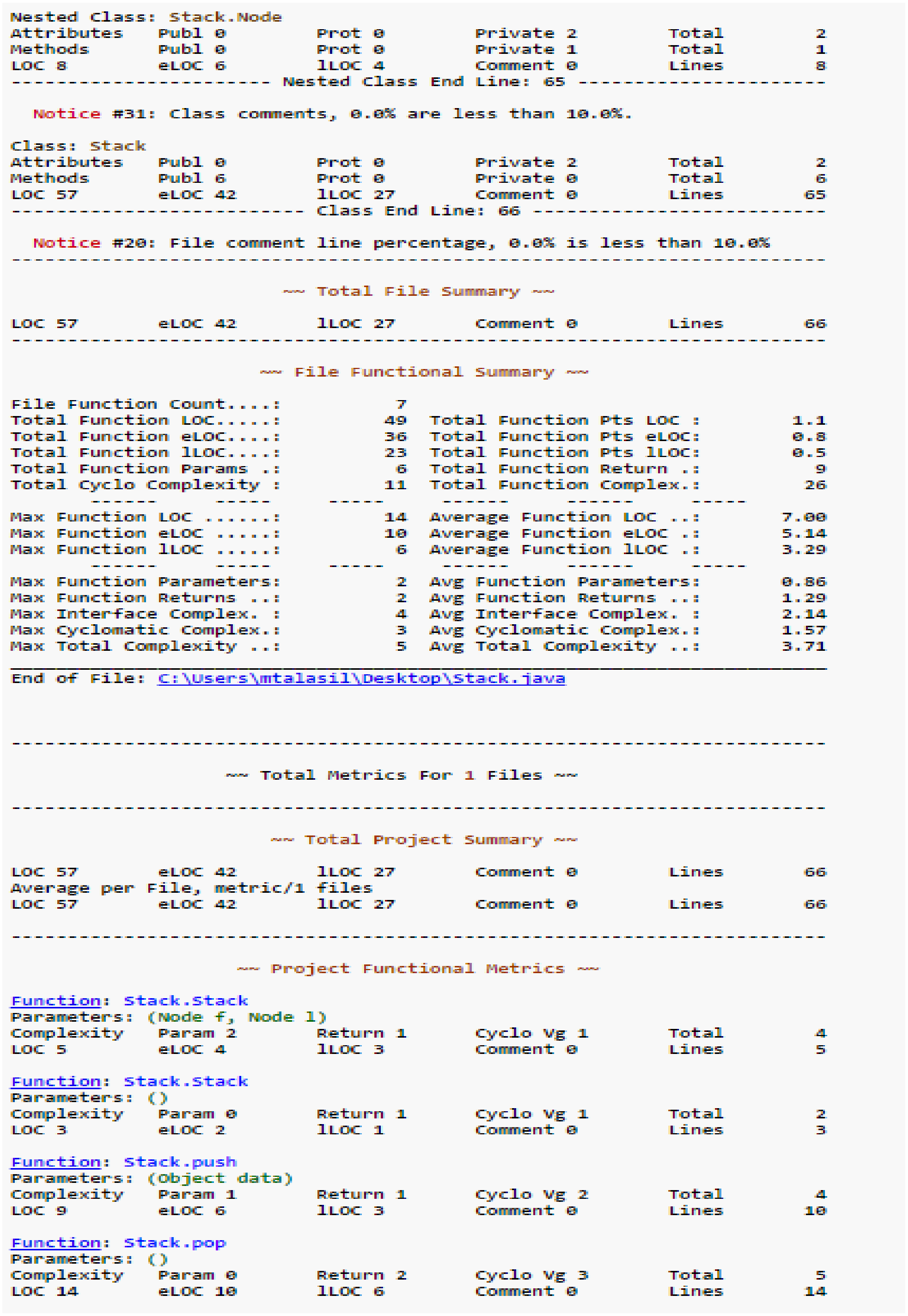
Finally a project quality profile and a quality density report are given. Which gives the total number of notices and where modifications are necessary in detail. For suppose file comment content is <10% which is very bad. Another instance, functions appears to have blank or null constraints. Which have a considerable impact on the memory of the program thereby effects the quality of code. Type, count and percent of a quality notice is clearly mentioned in the project quality profile. In project density profile, percentage of LOC, ELOC and lLOC are mentioned.

Functional Quality Metrics



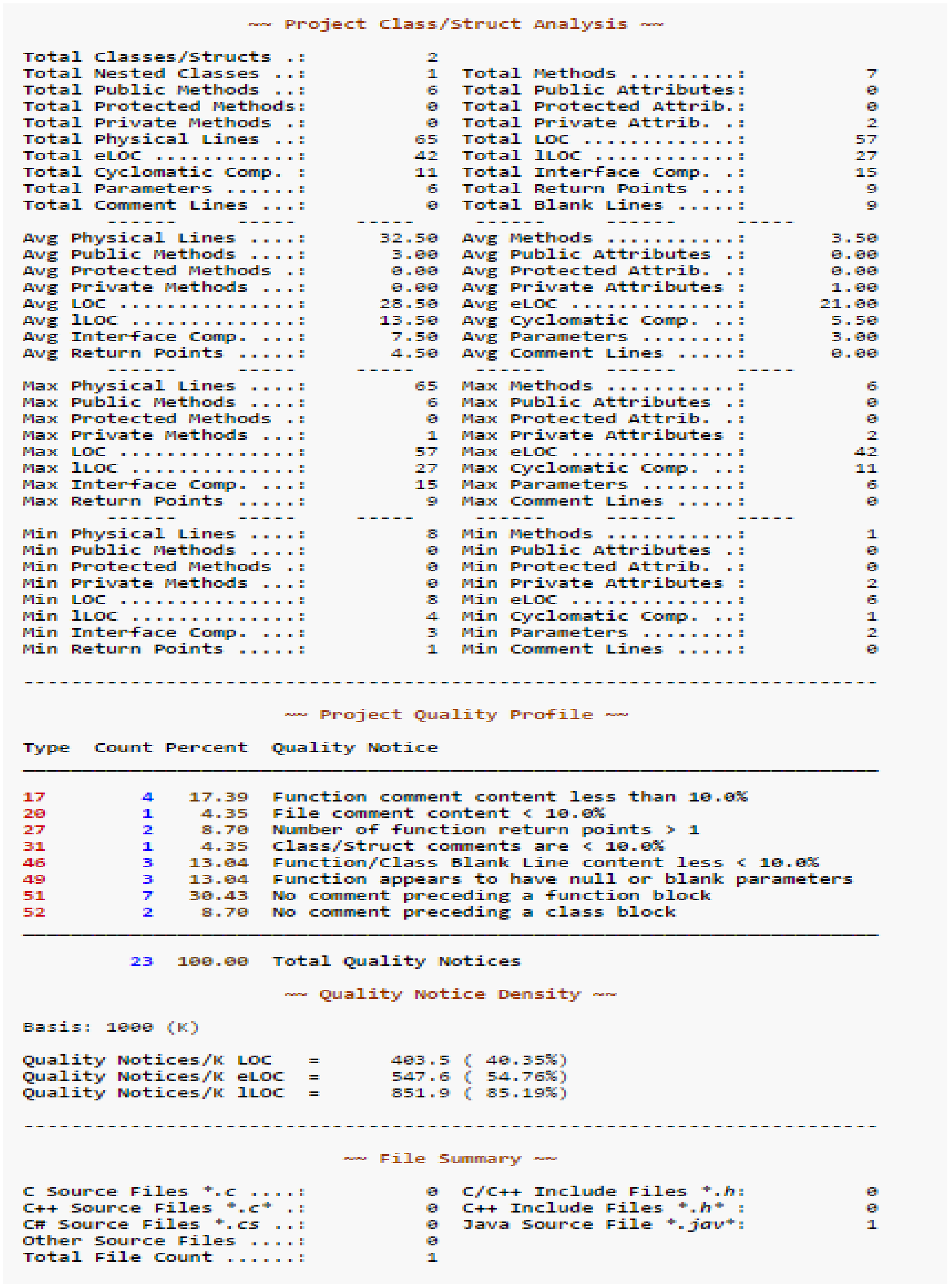
```
                  Report Banner - Edit rsm.cfg File

            Resource Standard Metrics™ for C, C++, C# and Java
                  Version 7.75 - mSquaredTechnologies.com

License Type: Shareware Evaluation License
Licensed To : Shareware End User - Distribute Freely
License No. : SW1380                          License Date: Dec 05, 1998
Build Date  : Sep  2 2009                         Run Date: Apr 23, 2015
©1996-2009 M Squared Technologies LLC™
________________________________________________________________________

License File: C:\Program Files (x86)\MSquared\M2 RSM\rsm.lic
Config. File: C:\Program Files (x86)\MSquared\M2 RSM\rsm.cfg
Command Line: -H -OC:\Users\mtalasil\Desktop\output\output.htm -c -fa -n
               -o -SFunctional Quality Metrics -FC:\Users\mtalasil\M2 RS
              M Wizard\input\rsm_file_list.lst
UDQN File    : C:\Program Files (x86)\MSquared\M2 RSM\rsm_udqn.cfg

                        Functional Quality Metrics

                          ~~ Function Metrics ~~
                        ~~ Class/Struct Metrics ~~
                         ~~ Complexity Analysis ~~
                          ~~ Quality Analysis ~~

File: C:\Users\mtalasil\Desktop\Stack.java
________________________________________________________________________


-------------------------- Class Begin Line: 1 -------------------------
Class: Stack

  Notice #52: Line 1: A class has been identified which does not
         have a preceding comment.  Comments that detail the purpose,
         algorithms, and parameter/return definitions are suggested.

------------------------ Function Begin Line: 5 ------------------------
Function: Stack.Stack
Parameters: (Node f, Node l)

  Notice #51: Line 5: A function has been identified which does not
         have a preceding comment.  Comments that detail the purpose,
         algorithms, and parameter/return definitions are suggested.

Function: Stack.Stack
Complexity   Param 2       Return 1      Cyclo Vg 1       Total        4
LOC 5        eLOC 4        lLOC 3        Comment 0        Lines        5
------------------------- Function End Line: 9 -------------------------

------------------------ Function Begin Line: 11 -----------------------
Function: Stack.Stack
Parameters: ()

  Notice #51: Line 11: A function has been identified which does not
         have a preceding comment.  Comments that detail the purpose,
         algorithms, and parameter/return definitions are suggested.

  Notice #49: The function contains no input parameters or void.
         Suggest using explict parameters for interface clarity.

Function: Stack.Stack
Complexity   Param 0       Return 1      Cyclo Vg 1       Total        2
LOC 3        eLOC 2        lLOC 1        Comment 0        Lines        3
------------------------- Function End Line: 13 ------------------------

------------------------ Function Begin Line: 15 -----------------------
Function: Stack.push
Parameters: (Object data)

  Notice #51: Line 15: A function has been identified which does not
         have a preceding comment.  Comments that detail the purpose,
         algorithms, and parameter/return definitions are suggested.

  Notice #17: Function comments, 0.0% are less than 10.0%.

Function: Stack.push
Complexity   Param 1       Return 1      Cyclo Vg 2       Total        4
LOC 9        eLOC 6        lLOC 3        Comment 0        Lines       10
------------------------- Function End Line: 24 ------------------------
```

```
------------------------ Function Begin Line: 26 -----------------------
Function: Stack.pop
Parameters: ()

  Notice #51: Line 26: A function has been identified which does not
          have a preceding comment.  Comments that detail the purpose,
          algorithms, and parameter/return definitions are suggested.

  Notice #17: Function comments, 0.0% are less than 10.0%.

  Notice #46: Function blank line percent, 0.0% is less than 10.0%.

  Notice #27: The number of function return points
          2 exceeds the specified limit of 1.

  Notice #49: The function contains no input parameters or void.
          Suggest using explict parameters for interface clarity.

Function: Stack.pop
Complexity   Param 0       Return 2      Cyclo Vg 3       Total         5
LOC 14       eLOC 10       lLOC 6        Comment 0        Lines        14
------------------------- Function End Line: 39 ------------------------

------------------------ Function Begin Line: 41 -----------------------
Function: Stack.peek
Parameters: ()

  Notice #51: Line 41: A function has been identified which does not
          have a preceding comment.  Comments that detail the purpose,
          algorithms, and parameter/return definitions are suggested.

  Notice #17: Function comments, 0.0% are less than 10.0%.

  Notice #46: Function blank line percent, 0.0% is less than 10.0%.

  Notice #27: The number of function return points
          2 exceeds the specified limit of 1.

  Notice #49: The function contains no input parameters or void.
          Suggest using explict parameters for interface clarity.

Function: Stack.peek
Complexity   Param 0       Return 2      Cyclo Vg 2       Total         4
LOC 7        eLOC 5        lLOC 3        Comment 0        Lines         7
------------------------- Function End Line: 47 ------------------------

------------------------ Function Begin Line: 49 -----------------------
Function: Stack.main
Parameters: (String[] args)

  Notice #51: Line 49: A function has been identified which does not
          have a preceding comment.  Comments that detail the purpose,
          algorithms, and parameter/return definitions are suggested.

  Notice #17: Function comments, 0.0% are less than 10.0%.

  Notice #46: Function blank line percent, 0.0% is less than 10.0%.

Function: Stack.main
Complexity   Param 1       Return 1      Cyclo Vg 1       Total         3
LOC 7        eLOC 6        lLOC 5        Comment 0        Lines         7
------------------------- Function End Line: 55 ------------------------

---------------------- Nested Class Begin Line: 57 ---------------------
Nested Class: Stack.Node

  Notice #52: Line 57: A class has been identified which does not
          have a preceding comment.  Comments that detail the purpose,
          algorithms, and parameter/return definitions are suggested.

------------------------ Function Begin Line: 61 -----------------------
Function: Stack.Node.Node
Parameters: (Object d, Node n)

  Notice #51: Line 61: A function has been identified which does not
          have a preceding comment.  Comments that detail the purpose,
          algorithms, and parameter/return definitions are suggested.

Function: Stack.Node.Node
Complexity   Param 2       Return 1      Cyclo Vg 1       Total         4
LOC 4        eLOC 3        lLOC 2        Comment 0        Lines         4
------------------------- Function End Line: 64 ------------------------
```



```
Nested Class: Stack.Node
Attributes    Publ 0        Prot 0        Private 2        Total         2
Methods       Publ 0        Prot 0        Private 1        Total         1
LOC 8         eLOC 6        lLOC 4        Comment 0        Lines         8
----------------------- Nested Class End Line: 65 ----------------------

  Notice #31: Class comments, 0.0% are less than 10.0%.

Class: Stack
Attributes    Publ 0        Prot 0        Private 2        Total         2
Methods       Publ 6        Prot 0        Private 0        Total         6
LOC 57        eLOC 42       lLOC 27       Comment 0        Lines        65
-------------------------- Class End Line: 66 --------------------------

  Notice #20: File comment line percentage, 0.0% is less than 10.0%
------------------------------------------------------------------------

                       ~~ Total File Summary ~~

LOC 57        eLOC 42       lLOC 27       Comment 0        Lines        66
------------------------------------------------------------------------

                     ~~ File Functional Summary ~~

File Function Count....:          7
Total Function LOC.....:         49   Total Function Pts LOC :       1.1
Total Function eLOC....:         36   Total Function Pts eLOC:       0.8
Total Function lLOC....:         23   Total Function Pts lLOC:       0.5
Total Function Params .:          6   Total Function Return .:         9
Total Cyclo Complexity :         11   Total Function Complex.:        26
       ------     -----     -----      ------     ------     -----
Max Function LOC ......:         14   Average Function LOC ..:      7.00
Max Function eLOC .....:         10   Average Function eLOC .:      5.14
Max Function lLOC .....:          6   Average Function lLOC .:      3.29
       ------     -----     -----      ------     ------     -----
Max Function Parameters:          2   Avg Function Parameters:      0.86
Max Function Returns ..:          2   Avg Function Returns ..:      1.29
Max Interface Complex. :          4   Avg Interface Complex. :      2.14
Max Cyclomatic Complex.:          3   Avg Cyclomatic Complex.:      1.57
Max Total Complexity ..:          5   Avg Total Complexity ..:      3.71
________________________________________________________________________
End of File: C:\Users\mtalasil\Desktop\Stack.java



------------------------------------------------------------------------

                    ~~ Total Metrics For 1 Files ~~

------------------------------------------------------------------------

                     ~~ Total Project Summary ~~

LOC 57        eLOC 42       lLOC 27       Comment 0        Lines        66
Average per File, metric/1 files
LOC 57        eLOC 42       lLOC 27       Comment 0        Lines        66

------------------------------------------------------------------------

                   ~~ Project Functional Metrics ~~

Function: Stack.Stack
Parameters: (Node f, Node l)
Complexity    Param 2       Return 1      Cyclo Vg 1       Total         4
LOC 5         eLOC 4        lLOC 3        Comment 0        Lines         5

Function: Stack.Stack
Parameters: ()
Complexity    Param 0       Return 1      Cyclo Vg 1       Total         2
LOC 3         eLOC 2        lLOC 1        Comment 0        Lines         3

Function: Stack.push
Parameters: (Object data)
Complexity    Param 1       Return 1      Cyclo Vg 2       Total         4
LOC 9         eLOC 6        lLOC 3        Comment 0        Lines        10

Function: Stack.pop
Parameters: ()
Complexity    Param 0       Return 2      Cyclo Vg 3       Total         5
LOC 14        eLOC 10       lLOC 6        Comment 0        Lines        14
```

```
Function: Stack.peek
Parameters: ()
Complexity   Param 0       Return 2      Cyclo Vg 2       Total        4
LOC 7        eLOC 5        lLOC 3        Comment 0        Lines        7

Function: Stack.main
Parameters: (String[] args)
Complexity   Param 1       Return 1      Cyclo Vg 1       Total        3
LOC 7        eLOC 6        lLOC 5        Comment 0        Lines        7

Function: Stack.Node.Node
Parameters: (Object d, Node n)
Complexity   Param 2       Return 1      Cyclo Vg 1       Total        4
LOC 4        eLOC 3        lLOC 2        Comment 0        Lines        4

Total: Functions
LOC 49       eLOC 36       lLOC 23       InCmp 15        CycloCmp     11

Function Points          FP(LOC) 0.9     FP(eLOC) 0.7    FP(lLOC)     0.4

------------------------------------------------------------------------

                   ~~ Project Functional Analysis ~~

Total Functions .......:          7 Total Physical Lines ..:         50
Total LOC .............:         49 Total Function Pts LOC :        0.9
Total eLOC ............:         36 Total Function Pts eLOC:        0.7
Total lLOC.............:         23 Total Function Pts lLOC:        0.4
Total Cyclomatic Comp. :         11 Total Interface Comp. .:         15
Total Parameters ......:          6 Total Return Points ...:          9
Total Comment Lines ...:          0 Total Blank Lines .....:          1
     ------     -----     -----     ------     ------     -----
Avg Physical Lines ....:       7.14
Avg LOC ...............:       7.00 Avg eLOC ..............:       5.14
Avg lLOC ..............:       3.29 Avg Cyclomatic Comp. ..:       1.57
Avg Interface Comp. ...:       2.14 Avg Parameters ........:       0.86
Avg Return Points .....:       1.29 Avg Comment Lines .....:       0.00
     ------     -----     -----     ------     ------     -----
Max LOC ...............:         14
Max eLOC ..............:         10 Max lLOC ..............:          6
Max Cyclomatic Comp. ..:          3 Max Interface Comp. ...:          3
Max Parameters ........:          2 Max Return Points .....:          2
Max Comment Lines .....:          0 Max Total Lines .......:         14
     ------     -----     -----     ------     ------     -----
Min LOC ...............:          3
Min eLOC ..............:          2 Min lLOC ..............:          1
Min Cyclomatic Comp. ..:          1 Min Interface Comp. ...:          1
Min Parameters ........:          0 Min Return Points .....:          1
Min Comment Lines .....:          0 Min Total Lines .......:          3

------------------------------------------------------------------------

                  ~~ Project Class/Struct Metrics ~~
              Parent LOC Metrics Include Nested LOC Metrics

Class: Stack
Attributes   Publ 0        Prot 0        Private 2        Total        2
Methods      Publ 6        Prot 0        Private 0        Total        6
Complexity   Param 6       Return 9      Cyclo Vg 11      Total       26
LOC 57       eLOC 42       lLOC 27       Comment 0        Lines       65

Nested Class: Stack.Node
Attributes   Publ 0        Prot 0        Private 2        Total        2
Methods      Publ 0        Prot 0        Private 1        Total        1
Complexity   Param 2       Return 1      Cyclo Vg 1       Total        4
LOC 8        eLOC 6        lLOC 4        Comment 0        Lines        8

Total: All Parent Classes/Structs
Attributes   Publ 0        Prot 0        Private 2        Total        2
Methods      Publ 6        Prot 0        Private 0        Total        6
Complexity   Param 6       Return 9      Cyclo Vg 11      Total       26
LOC 57       eLOC 42       lLOC 27       Comment 0        Lines       65

------------------------------------------------------------------------
```



```
                     ~~ Project Class/Struct Analysis ~~

Total Classes/Structs .:          2
Total Nested Classes ..:          1  Total Methods .........:         7
Total Public Methods ..:          6  Total Public Attributes:         0
Total Protected Methods:          0  Total Protected Attrib.:         0
Total Private Methods .:          0  Total Private Attrib. .:         2
Total Physical Lines ..:         65  Total LOC .............:        57
Total eLOC ............:         42  Total lLOC ............:        27
Total Cyclomatic Comp. :         11  Total Interface Comp. .:        15
Total Parameters ......:          6  Total Return Points ...:         9
Total Comment Lines ...:          0  Total Blank Lines .....:         9
      ------     -----       -----    ------     ------     -----
Avg Physical Lines ....:      32.50  Avg Methods ...........:      3.50
Avg Public Methods ....:       3.00  Avg Public Attributes .:      0.00
Avg Protected Methods .:       0.00  Avg Protected Attrib. .:      0.00
Avg Private Methods ...:       0.00  Avg Private Attributes :      1.00
Avg LOC ...............:      28.50  Avg eLOC ..............:     21.00
Avg lLOC ..............:      13.50  Avg Cyclomatic Comp. ..:      5.50
Avg Interface Comp. ...:       7.50  Avg Parameters ........:      3.00
Avg Return Points .....:       4.50  Avg Comment Lines .....:      0.00
      ------     -----       -----    ------     ------     -----
Max Physical Lines ....:         65  Max Methods ...........:         6
Max Public Methods ....:          6  Max Public Attributes .:         0
Max Protected Methods .:          0  Max Protected Attrib. .:         0
Max Private Methods ...:          1  Max Private Attributes :         2
Max LOC ...............:         57  Max eLOC ..............:        42
Max lLOC ..............:         27  Max Cyclomatic Comp. ..:        11
Max Interface Comp. ...:         15  Max Parameters ........:         6
Max Return Points .....:          9  Max Comment Lines .....:         0
      ------     -----       -----    ------     ------     -----
Min Physical Lines ....:          8  Min Methods ...........:         1
Min Public Methods ....:          0  Min Public Attributes .:         0
Min Protected Methods .:          0  Min Protected Attrib. .:         0
Min Private Methods ...:          0  Min Private Attributes :         2
Min LOC ...............:          8  Min eLOC ..............:         6
Min lLOC ..............:          4  Min Cyclomatic Comp. ..:         1
Min Interface Comp. ...:          3  Min Parameters ........:         2
Min Return Points .....:          1  Min Comment Lines .....:         0

------------------------------------------------------------------------

                        ~~ Project Quality Profile ~~

Type   Count Percent  Quality Notice
________________________________________________________________________

17         4   17.39  Function comment content less than 10.0%
20         1    4.35  File comment content < 10.0%
27         2    8.70  Number of function return points > 1
31         1    4.35  Class/Struct comments are < 10.0%
46         3   13.04  Function/Class Blank Line content less < 10.0%
49         3   13.04  Function appears to have null or blank parameters
51         7   30.43  No comment preceding a function block
52         2    8.70  No comment preceding a class block
________________________________________________________________________

          23  100.00  Total Quality Notices

                        ~~ Quality Notice Density ~~

Basis: 1000 (K)

Quality Notices/K LOC    =       403.5 ( 40.35%)
Quality Notices/K eLOC   =       547.6 ( 54.76%)
Quality Notices/K lLOC   =       851.9 ( 85.19%)

------------------------------------------------------------------------

                             ~~ File Summary ~~

C Source Files *.c ....:          0  C/C++ Include Files *.h:         0
C++ Source Files *.c* .:          0  C++ Include Files *.h* :         0
C# Source Files *.cs ..:          0  Java Source File *.jav*:         1
Other Source Files ....:          0
Total File Count ......:          1
```

Code:

```
public class Stack {
    Node first;
    Node last;

    public Stack(Node f, Node l) {
        first = f;
        last = l;
        first.next = last;
    }

    public Stack() {
        first.next = last;
    }

    public void push(Object data) {
        if(first == null) {
            first = new Node(data, null);
        }
        else {
            last.next = new Node(data, null);
            last = last.next;
        }

    }

    public Object pop() {
        if(first == null) {
```

```
        return -1;
    }
    else {
        Object item = last.data;
        Node cur = first;
        while (cur.next.next != null) {
            cur = cur.next;
        }
        last = cur;
        return item;
    }
}

public Object peek() {
    if(first == null) {
        return -1;
    }
    Object item = last.data;
    return item;
}

public static void main(String[] args) {
    Stack stack = new Stack(new Node(1, null), new Node(2, null));
    stack.push(3);
    System.out.println(stack.peek() == 3);
    stack.pop();
    System.out.println(stack.peek() == 2);
}
```

```
    private static class Node {
        Object data;
        Node next;

        private Node(Object d, Node n) {
            data = d;
            next = n;
        }
    }
}
```

<u>Part 2</u>

The proposed metric is on “Metrics of Software Architecture Changes Based on Structural Distance”.

Proposed Software Architecture Metric in “Metrics of Software Architecture Changes Based on Structural Distance”: “This paper concentrates on different versions of code and is trying to analyze the architectural change using Structural Distance rather than at architecture metrics like size, complexity, coupling and cohesion. This concentrates on changes between the versions at each individual component level. “In this paper, two endpoints of a major change are taken as reference points, and intermediate connectivity changes are examined relative to the endpoints”. They defined a graph kernel function to measure distance between software structures. This measures the architecture change as a transition between two end points [2]”.

“How does it relates to RSM tool Metrics?” It is also considering the functions, classes and lines but differently. Instead of finding size, complexity, coupling and soon it is trying to find the Structural Distance between various components in different versions of program and trying to analyze the architectural (i.e., what contents and where the contents are changes in the file physically so that modified there will saves a lot of time and can be done efficiently).

“My Personal Opinion”: This metric seemed to be useful as while different versions of the code are created. Most of the structure remains same expects the functions or modules where changes are required. Tracking those changes based on number of lines of code or functionality of the code or coupling between the codes seems to be less effective. For suppose while comparing two triangles if we compare distance between the edges then there is high probability of accuracy than comparing triangles for number of points or number of sides. Also, Working at module level seems to be very easy for a programmer to understand the changes than at line level or number of lines. I feel this metric is useful while comparing different versions of software rather than using RSM.

“Validation of Proposed Metric”: Used on four open source projects which used java as their source code on different versions of four projects. Then considered class as nodes and if any changes to nodes are not considered as, even if the names of the classes are changed software architecture won’t be effected. But this is not possible with lines of code metric as change of class name is also considered as change of line or addition of a space also a possible change even though which has less impact on architectural change. Then they captured properties like inheritance, association and user dependencies which have major impact on the code. Then analyzed the metric on different versions of open source code and presented the graphical analysis of transition between two architectures. Then they compared their metric to Lines of Code. Using Lines of Code they do observed continuous increase which does not mean no significant changes in code or Lines of Code failed to capture the changes properly. This metric clearly validated the Software Architecture metric which enhances the quality of the program.

References

1. http://msquaredtechnologies.com/m2rsm/docs/index.htm
2. https://www.cs.umd.edu/~basili/publications/proceedings/P114.pdf