CHAPTER - 4

NEW PROGRAM WEIGHTED COMPLEXITY MEASURE
CHAPTER 4(*)

NEW PROGRAM WEIGHTED COMPLEXITY MEASURE

4.1 INTRODUCTION

In software engineering, various metrics have been designed in an attempt to measure the complexity of systems. Software complexity directly affects maintenance activities like software understandability, modifiability, and software testability. Estimates suggest that about 40 to 50% of annual software expenditure involve maintenance of existing systems. Predicting software complexity can save millions in maintenance [2,11-12]. Clearly, if complexities could somehow be identified, then programmers could adjust development, maintenance, and testing procedures accordingly. This concern has motivated several researchers to define and validate software complexity measures[3,15,33,36].

Different complexity measures take into account different aspects of program complexity and thus it is difficult to have a single metric which may satisfy all aspects of program characteristics [4-5]. Halstead's software science metrics [36] and McCabe's cyclomatic number [38] are the two prominent contributions but account for different characteristics of a computer program. For Halstead's metric, the basis of detecting and measuring the complexity is the size of software module only and it does not take into account other characteristics of software systems such as program flow control, nesting and so on [36-38]. Similarly, cyclomatic number measures the complexity due to program flow control only, but does not keep track of other features of software systems such as operators, operands, etc.

Complexity is also affected by design and inter-dependence of statements in a program. For example, an assignment statement involving some variables is dependent upon those statements in which values of these variables have been calculated. These statements may be located at different positions in the program. In this metric, we have attempted to take into account the complexities due to size of software modules, position of a statement in the logic of the program, type of control structures present in the module, and their nesting. The concept of weight is used to quantify these various aspects of complexity of a software module.

Before we explain the weighted complexity measure in depth, first we recall the Halstead's software science family of measures, McCabe's cyclomatic number, and composite complexity measure [36,38,24].

4.2 DESIGN OF WEIGHTED COMPLEXITY MEASURE

A program is a set of statements which in turn include operators and operands. Thus, the program statements, operators and operands are basic units of a program. The prominent factors which contribute to complexity of a program are:

i) Size

Large programs incur problems just by virtue of the volume of information that must be absorbed to understand the program and more resources have to be used in their maintenance [3,18]. Therefore, size is a factor that adds complexity to a program.

ii) Position of a statement

We assume that the statements which are at the beginning of the program logic are simple and hence easily understandable and thus contribute less complexity than those which are at a deeper level of the logic of a program. So we assign a weight (multiplier) 1 to the first executable statement and 2 to the second and so on. It may be treated as positional weight \( W_p \).

iii) Type of control structures

A program with more control structures is considered to be more complex and vice versa. But, we assume that different control structures contribute to the complexity of a program differently. For example, iterative control structures like while, do, repeat until, for to do contribute more complexity than decision making control structures like if then else. So, we assign different weights to different control structures.

iv) Nesting

A statement which is at a deeper level (higher nesting level) is harder to understand and thus contributes more complexity than otherwise. We take effect of nesting by assigning weight 1 to statements at level one, weight 2 for those statements which are at level 2 and so on. The weight for sequential statements is taken as zero.

By taking these assumptions into account, a weighted complexity measure of a program P is suggested as:

\[
C_w(P) = \sum_{i=1}^{j} (W_t)_i * (m)_i
\]

where

- \( C_w(P) \) : Proposed weighted complexity measure of program P,
- \( (m)_i \) : Count of unique operators (n1) and operands (n2) in the \( i \) th executable statement of program module P,
- \( j \) : Total number of executable statements in program P,
- \( (W_t)_i \) : Total weight of \( i \) th executable statement in program P

where

\[
W_t = W_p + W_h + W_c
\]
Here

\( W_p \): positional weight i.e. weight due to position of the statement,

\( W_h \): weight due to nesting level of control structures and it is

- = 0 for sequential statement
- = 1 for a control structure at outer level,
- = 2 for a control structure at next inner level
and so on.

\( W_c \): weight due to type of control structures and it is

- = 0 for sequential statement
- = 1 for decision making control statements like if.. then.. else, case statements.
- = 2 for iterative control structures like while.. do.. for.. to.. do.. repeat.. until statements.

\( \Sigma \): Summation symbol

\( i \): Index variable

### 4.3 EXPERIMENTAL RESULTS

Consider the two test programs P1 and P2 as listed below:

**Test Program P1**

PROGRAM P1(var a : elements; n : integer);
VAR
  i, j, k : integer;
begin
  i := 0;
  begin
    for j = 1 to n-i do
      if a[j] > a[j+1] then
        begin
          k := a[j];
          a[j] := a[j+1];
          a[j+1] := k;
        end.
    i := i + 1;
  end.
end.

**Test Program P2**

PROGRAM P2(var a : elements; n : integer);
VAR
  i, j, k : integer;
begin
  flag := boolean;
  begin
    i := 0;
    while (i < n) and (not flag) do
      begin
        flag := true;
        i := i + 1;
      end.
    for j = 1 to n-i do
      if a[j] > a[j+1] then
        begin
          k := a[j];
          a[j] := a[j+1];
          a[j+1] := k;
          flag := false;
        end.
    end.
end.
Applying Halstead's theory of software science [36], we get the following results

<table>
<thead>
<tr>
<th></th>
<th>For P1</th>
<th>For P2</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>54</td>
<td>73</td>
</tr>
<tr>
<td>V</td>
<td>211</td>
<td>335</td>
</tr>
<tr>
<td>E</td>
<td>3376</td>
<td>7976</td>
</tr>
</tbody>
</table>

From these values, it is clear that program P1 is less complex than program P2. But, this study does not reflect the effect of control structures, nesting level and positional status of a statement in a program.

Applying McCabe's Complexity metric to the same programs P1 and P2, we get the following values:

\[ V(G_1) = 2 + 1 = 3, \text{ here } G_1 \text{ is graph structure of program P1.} \]

\[ V(G_2) = 3 + 1 = 4, \text{ here } G_2 \text{ is graph structure of program P2.} \]

By looking at these values, we infer that program P2 is more complex than program P1. But, this study does not take into account the complexity due to size, position of a statement in the logic of the program, and nesting level of control structures.

Now, by applying the weighted program complexity measure to the same programs P1 [Table 4.1] and program P2 [Table 4.2], we get the following results:

<table>
<thead>
<tr>
<th></th>
<th>For P1</th>
<th>For P2</th>
</tr>
</thead>
<tbody>
<tr>
<td>( C_w(P1) )</td>
<td>270</td>
<td>563</td>
</tr>
</tbody>
</table>

For the count of \((m)\), we have used the same counting rules as in [33].

From these results, we infer that program P1 is less complex than program P2. But, here, differences are due to size, position of a statement in the logic, structured design methodology of control structures, and nesting level of control structures.

We have applied the above mentioned complexity measures to 7 pairs of programs [60], collected form open literature. In each test pair, the second program is supposed to be less complex than the first. Therefore, the values of the complexity metrics applied to these pairs should be less for the second program than the first. Results of our calculations are presented in Table 4.3. From this table, it is clear that this trend holds. Thus, we infer that, the proposed weighted complexity measure detects complexity and gives realistic estimates. This is because the weighted complexity measure also takes into account the complexity due to factors not considered earlier in the Halstead's or McCabe measures, that is, size, position of a statement, control structures and their nesting level. As a software system is developed in terms of modules, of size about 50 - 60 lines of code, so the proposed complexity measure can be used to calculate and compare the complexities of software modules and hence of software systems.
The proposed complexity measure, also, attempts to resolve the following issues relating to McCabe’s complexity measure:

a) The value of McCabe’s cyclomatic number $V(G)$ is 1 for a linear sequence of any length i.e. it is unable to detect the complexities of linear sequences of different lengths. But in the proposed complexity measure we are assigning different weights to different statements according to their positions and thus there is no meaning of having the same complexity with the increased program length.

b) Consider the following two code segments:

i) IF $a=0$ THEN $p_1$
   ELSE $p_2$;

ii) IF $a=0$ AND $b=2$ THEN $p_1$
   ELSE $p_2$;

McCabe’s cyclomatic number $V(G)$ makes no difference between these since $V(G) = 1$ for both. But according to our measure

$$C_w(i) = 24 \quad \text{and} \quad C_w(ii) = 33$$

Clearly, $C_w(ii)$ is greater than $C_w(i)$, which indicates that code segment (ii) is more complex than (i).

c) Again, McCabe’s cyclomatic measure $V(G)$ makes no difference between the complexities of the following two code segments (i) and (ii) since $V(G) = 2$ for both.

i) IF $a$ THEN $x$ ELSE $y$
   IF $b$ THEN $y$ ELSE $z$;

ii) IF $a$ THEN
   IF $b$ THEN $y$
   ELSE $z$
   ELSE $y$;

In other words, $V(G)$ does not reflect the effect of nesting but in the proposed complexity measure we are assigning weights due to nesting of control structures and thus we are taking the effect of nesting level of control structures.

Clearly, $C_w(ii) = 48 > C_w(i) = 32$. 
### Table 4.1

Application of weighted complexity measure to program P1

<table>
<thead>
<tr>
<th>Statement no.</th>
<th>Executable Statement</th>
<th>(m), W_p</th>
<th>W_h</th>
<th>W_c</th>
<th>(W_p), (W_h), (W_c)</th>
<th>(W_p) + W_h + W_c</th>
<th>(W_p) * (m),</th>
</tr>
</thead>
<tbody>
<tr>
<td>(s1)</td>
<td>i := 0;</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>4 * 1 = 4</td>
</tr>
<tr>
<td>(s2)</td>
<td>for j:= 1 to n-i do</td>
<td>7</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>7 * 5 = 35</td>
</tr>
<tr>
<td>(s3)</td>
<td>if a [j] &gt; a [j+1]</td>
<td>7</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>6</td>
<td>7 * 6 = 42</td>
</tr>
<tr>
<td>(s4)</td>
<td>k := a [j];</td>
<td>6</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>6</td>
<td>6 * 6 = 36</td>
</tr>
<tr>
<td>(s5)</td>
<td>a[i+1] := a[i];</td>
<td>7</td>
<td>5</td>
<td>2</td>
<td>0</td>
<td>7</td>
<td>7 * 7 = 49</td>
</tr>
<tr>
<td>(s6)</td>
<td>a[i+1] := k;</td>
<td>8</td>
<td>6</td>
<td>2</td>
<td>0</td>
<td>8</td>
<td>8 * 8 = 64</td>
</tr>
<tr>
<td>(s7)</td>
<td>i := i+1;</td>
<td>5</td>
<td>7</td>
<td>1</td>
<td>0</td>
<td>8</td>
<td>5 * 8 = 40</td>
</tr>
</tbody>
</table>

So

\[
C_w(P) = \Sigma (W_i) * (m), = 4 + 35 + 42 + 36 + 49 + 64 + 40 = 270
\]

### Table 4.2

Application of weighted complexity measure to program P2

<table>
<thead>
<tr>
<th>Statement no.</th>
<th>Executable Statement</th>
<th>(m), W_p</th>
<th>W_h</th>
<th>W_c</th>
<th>(W_p), (W_h), (W_c)</th>
<th>(W_p) + W_h + W_c</th>
<th>(W_p) * (m),</th>
</tr>
</thead>
<tbody>
<tr>
<td>(s1)</td>
<td>flag := false;</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>4 * 1 = 4</td>
</tr>
<tr>
<td>(s2)</td>
<td>i := 0;</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>4 * 2 = 8</td>
</tr>
<tr>
<td>(s3)</td>
<td>while (i &lt; n) and (not flag) do</td>
<td>8</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>6</td>
<td>8 * 6 = 48</td>
</tr>
<tr>
<td>(s4)</td>
<td>flag := true;</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>5</td>
<td>4 * 5 = 20</td>
</tr>
<tr>
<td>(s5)</td>
<td>i := i+1;</td>
<td>5</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>6</td>
<td>5 * 6 = 30</td>
</tr>
<tr>
<td>(s6)</td>
<td>for j:= 1 to n-i do</td>
<td>7</td>
<td>6</td>
<td>2</td>
<td>2</td>
<td>10</td>
<td>7 * 10 = 70</td>
</tr>
<tr>
<td>(s7)</td>
<td>if a [j] &gt; a [j+1]</td>
<td>7</td>
<td>7</td>
<td>3</td>
<td>1</td>
<td>11</td>
<td>7 * 11 = 77</td>
</tr>
<tr>
<td>(s8)</td>
<td>k := a [j];</td>
<td>6</td>
<td>8</td>
<td>3</td>
<td>0</td>
<td>11</td>
<td>6 * 11 = 66</td>
</tr>
<tr>
<td>(s9)</td>
<td>a[j] := a[j+1];</td>
<td>7</td>
<td>9</td>
<td>3</td>
<td>0</td>
<td>12</td>
<td>7 * 12 = 84</td>
</tr>
<tr>
<td>(s10)</td>
<td>a[j+1] := k;</td>
<td>8</td>
<td>10</td>
<td>3</td>
<td>0</td>
<td>13</td>
<td>8 * 13 = 104</td>
</tr>
<tr>
<td>(s11)</td>
<td>flag := false;</td>
<td>4</td>
<td>11</td>
<td>3</td>
<td>0</td>
<td>14</td>
<td>4 * 14 = 56</td>
</tr>
</tbody>
</table>

So

\[
C_w(P) = \Sigma (W_i) * (m), = 4 + 8 + 48 + 20 + 30 + 70 + 77 + 66 + 84 + 104 + 56 = 563
\]
Table 4.3
Application of complexity metrics to pairs of programs of different complexities

<table>
<thead>
<tr>
<th>P.No</th>
<th>N</th>
<th>V</th>
<th>E</th>
<th>E_w</th>
<th>C_w</th>
<th>V(G)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1a</td>
<td>39</td>
<td>163</td>
<td>2717</td>
<td>3504</td>
<td>147</td>
<td>2</td>
</tr>
<tr>
<td>1b</td>
<td>30</td>
<td>117</td>
<td>1463</td>
<td>1632</td>
<td>64</td>
<td>1</td>
</tr>
<tr>
<td>2a</td>
<td>79</td>
<td>376</td>
<td>9400</td>
<td>10225</td>
<td>318</td>
<td>2</td>
</tr>
<tr>
<td>2b</td>
<td>66</td>
<td>299</td>
<td>5980</td>
<td>7906</td>
<td>283</td>
<td>1</td>
</tr>
<tr>
<td>3a</td>
<td>62</td>
<td>291</td>
<td>5820</td>
<td>9800</td>
<td>382</td>
<td>3</td>
</tr>
<tr>
<td>3b</td>
<td>53</td>
<td>240</td>
<td>4000</td>
<td>6112</td>
<td>301</td>
<td>2</td>
</tr>
<tr>
<td>4a</td>
<td>65</td>
<td>306</td>
<td>5100</td>
<td>7105</td>
<td>309</td>
<td>2</td>
</tr>
<tr>
<td>4b</td>
<td>52</td>
<td>232</td>
<td>2900</td>
<td>4109</td>
<td>225</td>
<td>1</td>
</tr>
<tr>
<td>5a</td>
<td>38</td>
<td>145</td>
<td>2071</td>
<td>2407</td>
<td>175</td>
<td>3</td>
</tr>
<tr>
<td>5b</td>
<td>26</td>
<td>96</td>
<td>533</td>
<td>1973</td>
<td>103</td>
<td>2</td>
</tr>
<tr>
<td>6a</td>
<td>63</td>
<td>289</td>
<td>5780</td>
<td>9708</td>
<td>182</td>
<td>2</td>
</tr>
<tr>
<td>6b</td>
<td>60</td>
<td>264</td>
<td>4400</td>
<td>6109</td>
<td>120</td>
<td>1</td>
</tr>
<tr>
<td>7a</td>
<td>82</td>
<td>394</td>
<td>13133</td>
<td>18223</td>
<td>564</td>
<td>3</td>
</tr>
<tr>
<td>7b</td>
<td>73</td>
<td>335</td>
<td>11167</td>
<td>13010</td>
<td>425</td>
<td>2</td>
</tr>
</tbody>
</table>

4.4 SUMMARY

A new weighted complexity measure has been proposed which takes into account different aspects of complexity: Halstead's size, McCabe's control structures, their nesting, and positional complexity of a statement in the logic of a program. Our study shows that the effect of these structures on complexity is quite significant. We have taken small programs in our present study, but we expect that these results will serve as guidelines to explore more complex systems. Application of conclusions to real-life situations needs further study. We have also tried to resolve some issues relating to McCabe's complexity measure as well.