Software Quality Measures to determine the Diagnosability of PLC Applications

Mohammed Bani Younis  
Helmut-Schmidt University  
Holstenhofweg 85  
22043 Hamburg, Germany  
baniy@hsu-hh.de

Georg Frey  
University of Kaiserslautern  
Erwin-Schrödinger-Str. 12  
67663 Kaiserslautern, Germany  
frey@eit.uni-kl.de

Abstract

In case of failures in a PLC controlled process the problem of diagnosis arises. Today this is often a manual task. Whenever the failure in the controlled system is not directly related to an actuator, its sources have to be tracked back through the PLC program to the corresponding sensors. This process can be very time consuming. The presented work aims at providing a means for determining this effort and to assist the diagnosis process. To do so, software quality measures are studied. The evaluation of these measures on a program shows the most complex parts and hence gives hints on possible improvements. During the calculation of the measures graphical representations of some dependency relations are derived. These can be used as an aid for the engineer in the actual diagnosis process. Two case studies illustrate the presented approach.

1. Introduction

Programmable Logic Controllers (PLCs) are still the workhorse of industrial automation. The main tasks of a PLC normally fall into the category of discrete event or logic control, possibly with underlying time-discrete implementations of continuous controllers like time-discrete PID algorithms. A PLC closes the loop in a system by reading sensor values produced in the plant, calculating a corresponding response, and sending the results back to the plant in the form of commands for actuators. The actuators influence the process and hence may trigger new sensor values again.

Given a system controlled by a PLC often the problem of diagnosis in case of a failure arises. Diagnosis is aimed at finding the source of a failure in a system. Today there are advanced methods for failure diagnosis for discrete event systems ([1], [2], [3], [12]). However those methods have not yet found wide acceptance in industry. The standard approach in manufacturing systems is still manual diagnosis.

In manual diagnosis, a failure normally is detected when an expected event in the controlled process is missing, e.g. an actuator that should do something is not doing it. Manual diagnosis obviously starts by checking the actuator responsible for the missing action. If the actuator is found to work correctly, the next step is to track back the reasons through the PLC. In this step generally the absence of errors in the algorithms implemented on the PLC is assumed. Hence, the task is to track back the dependency of the output signal corresponding to the failing actuator – possibly over several internal variables of the PLC – to the corresponding input signals. After these are found, the respective sensors have to be checked for failures.

Given the size and the complexity of today’s PLC applications this form of diagnosis can be very time consuming and in turn costly because the process can not run until the failure is found. The amount of time needed for diagnoses clearly depends on the structure and style of the PLC program. This is where the concept of Software Quality comes into focus.

The aim of the presented work is to investigate how known software quality measures can be used to determine the ease of understanding the connections inside a PLC program and hence the effort for manual diagnosis. The considered PLC programs are assumed to be given in Instruction List language (IL). IL is used for the application of the approach because IL is the most commonly used PLC language in Europe.

Since IL is a special form of Assembly language, for several metrics to be applied, it is necessary to transform the program to a higher level description. To this end, the presented method utilizes a reverse engineering approach for IL programs presented in [4]. During the calculation of the metrics graphical representations of dependency relations are derived. These can be used as an aid for the engineer in the actual diagnosis process.

The paper is structured as follows. Section 2 discusses software quality in general and the categories of it applicable for the described aim. Selected measures and their application to PLC programs are presented in Section 3. Section 4 explains the implementation of the measures. Two case studies are described in Section 5. Section 6 concludes this paper and gives an outlook on future work.
2. Software Quality and Diagnosis

In the Software Engineering community the field of measures or metrics to measure software characteristics is maturing, see e.g. [5] and [6] for overviews. Some of the best-known already date back to the late seventies ([7], [8]). However, in the area of PLC programs software quality is rarely studied. Frey [9] introduced the concept of Transparency to measure the understandability of PLC programs described by a special form of Petri Net. Dandachi et al. [10] provided similar metrics to be applied to PLC programs given in Sequential Function Chart language. A different application of software measures is presented by Lucas and Tilbury in [11]. There, the complexity of solutions to the same problem described with different PLC languages is measured and compared.

In general, the quality of a product is not quite easy to describe, because this concept signifies differently for different people. If for example the readability of a program is considered to be important for someone, for others the compactness of a software program could be of higher importance. With the help of this example another problem is quite apparent, namely the mutual influence and the distinctive existence of the characteristics. To make a program understandable, many explanations may exist in this program, but in consequence the volume of the software increases. Hence, it is important to agree on characteristics which describe the quality of a program in a most reasonable way. Quality is defined in ISO 8402 standard [12] as:

The totality of characteristics of an entity that bear on its ability to satisfy stated or implied needs.

The PLC program software for controlling a process is application software according to the ANSI/IEEE 610 [13]. The ISO/IEC 9126 standard defines the software quality for application software as [14]:

A set of attributes of a software product by which its quality is described and evaluated. A software quality characteristic may be refined into multiple levels of sub-characteristics.

Figure 1 shows the software quality model according to ISO/IEC 9126. The standard defines six quality characteristics; each is associated with a number of sub-characteristics. For the selection of characteristics it is important to know the main aim of the quality analysis. In the presented work this is the manual investigation of dependency relations in the program for the purpose of diagnosis. Therefore, here, high quality of a program means that after a failure in the controlled system, it is easy for maintenance personnel to find the corresponding structures of the PLC program that are involved.

This leads to the fact that some sub-characteristics from Figure 1 are less relevant. Another restriction of the sub-characteristics follows from the assumption that the code is accepted in itself as accurate and thus no tests on the IL itself should be performed. Hence, the functionality as a software characteristic is not considered here. Also the understandability is not considered here, because the subjective sub-characteristics, like the understandability, cannot be evaluated automatically on a code.

![Figure 1: Software quality model of ISO 9126 with quality characteristics and sub-characteristics.](image)

In the following, the characteristics which are seen to be relevant for the diagnosis and can be measured objectively will be explained in details.

The recoverability of a program can be examined by the determination of its complexity. If a program e.g., through the use of many internal variables, is elaborately built up, the restoration of the program, after a loss lasts accordingly longer.

The most important point for the diagnosis of IL is the analyzability. The complexity of a program again is an important influence here. If the code is simply structured, the analysis of dependencies is also simple.

The stability of a program indicates how the program behaves in case of changes. Again the complexity can help in this issue. An easy program is also easier to modify, without the risk to endanger its stability.

The last sub-characteristic which is related to the diagnosis task is the replaceability which is important for a modular description of the program. This modular structure not only helps in replacing parts of it but also in tracking values during diagnosis.

Since structure and complexity of IL programs plays the main role for the presented purpose, in the following Section related standard measures are presented and their applicability to IL programs as well as their impact on diagnosis is discussed.
3. Structure and Complexity Measures

There are a lot of measures which are normally used nowadays to determine the quality of software. Most measures concentrate more on the program’s structure than on the contents. The size measure belongs to the most often used measures, namely the LOC (Lines of Code) and the Halstead measure with which the software will be measured in terms of the operators and operands. Another classical complexity measure shows the “Cyclomatic Complexity” from McCabe which refers to the flow chart of the code. A measure which determines the complexity of a program by the investigation of its graph is the “Tree Impurity”.

These measures will be explained individually in the following sub-sections. Besides, for every measure the merits and demerits are listed, as well as generally the application of it to IL programs and in particular the ability to determine the quality for diagnosis.

3.1. Lines of code (LOC)

The size measures like the Lines of code (LOC) are the simplest software measures available. These are used to count the program lines of a code. It is supposed that a program with a lot of lines is disproportionately complicated and is not effective. There are several ways to measure the size of a program. Lines of comments to the code can be counted or skipped. This is also valid in case of blank lines. If the comments, as well as the blank lines are not counted, the measure is called “Non Commentary Source Statements” (NCSS).

The merit of this measure is its simplicity. It is easy to understand and the process for calculating is simple. The demerit originates from the different kinds of this measure. It is unclear how to account for comments which on the one side serve for the explanation of the code, but at the same time raise the LOC value.

The application of this measure to PLC IL is very simple. Besides, it is reasonable that the measurement is to be carried out without comments and blank lines and to compare the results of two different programs of the same plant with each other. However, for the diagnosis this measure delivers only a coarse prediction about the complexity of the single programs or sub-routines.

3.2. Halstead Measure

According to Halstead [7], a computer program is considered as a collection of tokens that can be classified as either operators or operands [6]. With these measures first the operands and operators in a program are counted. Operands are normally variables and constants, whereas operators are the symbols that influence the value or the arrangement of an operand.

Primary measures are: the number of distinct operators in a program (µ₁), the number of distinct operands in a program (µ₂), the total number of operator occurrences (N₁), and the total number of operand occurrences (N₂).

The size of the vocabulary arises from this as: µ = µ₁ + µ₂. The implementing length N is given by: N = N₁ + N₂. The volume V of the program is defined as V = N · log₂µ. Finally, the difficulty D of an algorithm in a programming language is defined by D = µ₁ · N₂ / (2 · µ₂).

If new operators are added to a code or the available operands are often used, the difficulty of the program rises. From the volume and the difficulty of a program, a new measure can be formed: the effort E which one must pursue to understand the whole program: E = V · D.

The merit of this measure (E) is again the simplicity of the determination of the required values and the possible automation of this procedure. Furthermore it was shown by experiments that the Halstead measures deliver a good measure of the complexity.

The demerits exist above all in the difficulty of the definition of operators and operands, because there is room for discussion for example on the point whether constants should be counted as operands for the complexity calculation. The measure also takes into consideration simply lexical or textual complexity of the program and states nothing about the internal complexity.

The effort measure (and also the measures of Halstead) can be applied to PLC IL programs very well. For this the distinct variables, like the inputs, outputs, internal variables, etc. are taken as operands and the logical instructions as operators. Those can be easily added up. Also the value of the effort can thereby be quickly calculated and used as a measure of the complexity of the IL program.

3.3. McCabe’s Cyclomatic Complexity

McCabe’s Cyclomatic Complexity is related to the flow graph of a program. With it the effort required to test a software module can be determined. It was designed to indicate a program's testability and understandability. With it, the number of linearly independent paths in a flow graph is measured.

If a flow graph has been disposed to a program, this measure can be calculated as follows:

\[ M = v(F) = e - n + 2p \]

where \( v(F) \) is called the cyclomatic number of the graph F, \( e \) is the number of edges, \( n \) is the number of nodes, and \( p \) is the number of unconnected parts of the graph. The nodes expose a block of sequential instructions which show no internal branching. The edges show a possible control flow between the nodes of the connected independent program components, e.g. modules or procedures.
If $p = 1$, then the cyclomatic complexity measure can also be calculated by $v(G) = d + 1$, with $d$ as the number of decisions in the graph.

The cyclomatic complexity is additive where the complexity of several graphs considered as a group is equal to the sum of the individual graph’s complexities [6]. This measure is also easy to calculate. The demerit is that it demands a flow graph of the code. In addition, the complexity of the data flow is not taken into consideration in a program.

For IL programs this measure can be used, e.g., in connection with conditional jumps, because in this case the number of decisions can be indicated and counted. An indirect application is possible by transformation of the IL program into automata first.

### 3.4. Tree Impurity

If the information flow can be specified between the single modules, a graph is created where the nodes represent the modules and the edges are the information flow. If this graph owns no cycle which means that no path starting from a node ends at the same node, then the outcome of the graph is a tree. If all nodes on the other hand in a graph are connected directly to each other by a single edge then this graph is called a complete graph, $K_n$, which owns $n$ nodes and $n(n - 1)/2$ edges. The more a system deviates from a pure tree structure to a graph structure, the more complex it will be [15]. This also means that a complete graph is the most complicated structure and thereby the description of program through it is not suitable. The difference between the graphs arising from the program and the tree structure for this graph is called tree impurity.

$$m(G) = \frac{2(e - n + 1)}{(n-1)(n-2)}$$

In this equation $e$ is the number of edges in G and $n$ is the number of nodes. If the value $m(G)$ tends to zero this implies that it is an easy graph namely closer to tree structure.

This measure is well suited to examine the complexity between the modules of a program. The more a system resembles a tree structure, the more comprehensive it should be, and the single modules can be more easily exchanged or modified. If a graph structure of the program is given the calculation of $m(G)$ value can be simply done.

Nevertheless, the demerit originates in the necessity to build a graph with which one must fix, which of the information exchanged between the modules are relevant for this measure.

To apply this measure to IL programs the calls of distinct blocks can be determined and build into a graph. The nearer the tree impurity of this graph is to zero the simpler diagnosis in case of a failure is.

### 3.5. Assessment of the Structural Measures

The following table briefly summarizes the evaluation of the measures presented above and shows their applicability to IL PLC programs (cf. Table 1).

<table>
<thead>
<tr>
<th>Measure</th>
<th>Effort of Evaluation</th>
<th>Applicability to IL</th>
<th>Connection to Diagnosis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>+</td>
<td>+</td>
<td>- (Serves for the coarse appraisal)</td>
</tr>
<tr>
<td>Halstead</td>
<td>+</td>
<td>+</td>
<td>0 (Overview about operators and operands)</td>
</tr>
<tr>
<td>McCabe</td>
<td>0</td>
<td>- (Graph is necessary)</td>
<td>- (Information about conditional jumps)</td>
</tr>
<tr>
<td>Tree Impurity</td>
<td>-</td>
<td>0 (Graph is necessary)</td>
<td>+ (Blocks call)</td>
</tr>
</tbody>
</table>

Table 1: Evaluation of the discussed measures (+ = good; 0 = fair - = bad).

### 4. Measures Implementation

This section describes the implementation of the measures described above. All blocks of a program (modules) for which the measures should be calculated must be specified. Nevertheless, their treatment always occurs individually, i.e. every transformation of an IL file into an XML file and then the information retrieval to the determination of the measures always occurs for every block separately. The used XML format is compliant with the working proposal of PLCOpen [17].

The derivation of these measures is an extension to the visualization of the IL PLC programs explained in [16]. The measures are built from the XML transformed from the PLC text which is transformed to contain only the Instruction and Operand elements. The visualized formal description is reformatted and manipulated in order to adjust it to be able to build the SW measures.

The implementation of the measures will be described in details in the following sub sections. Moreover an illustrating example of an STEP 5 IL PLC program (cf. Figure 2) will be used. The XML segment of the IL code after the transformation is shown in Figure 3

#### 4.1. Size Measure

With the determination of the measure a coarse overview about the size of the program as well as about its complexity should be given. Because one cannot get automated propositions regarding the complexity of comments, these as well as the blank lines are to be neglected. The size measure which will be measured is then the NCSS.
The single nodes which show the lines in IL (IL.Row) are processed individually. If the current tag contains an instruction childNode the counter of NCSS is incremented. The realization occurs to the complete XML file worked on. The result is given afterwards for each block distinctly. Applying the size measure to the program block give in Figure 2, the LOC is NCSS = 20.

4.2. Halstead Measure

The Halstead Measure is also applied to single blocks. Then the total Halstead measure of a program arises from the creation of the average value of all Halstead results.

To calculate the Halstead measure, an intermediate step is inserted where the conditions for setting, resetting or assigning a value to an operand of the IL-block are taken as condition of equations. The inception of the equations describes which operand in the program under which conditions is changed. The second element of the equation means in which way this operand is changed, i.e. set by “S”, reset “R” or assigned “=”. In the last part of the equation stand the variables that serve as conditions for this change. To keep an overview in case of several blocks, it is given behind every equation from which block this equation is generated. To explain the methodology the program block in Figure 2 serves as an example. The resulting equations to obtain the Halstead measure are:

\[
\begin{align*}
M33.3 & \text{ S M33.1 U M30.1} & \text{PB014} \\
M33.3 & \text{ R M32.0 ON T5} & \text{PB014} \\
M33.4 & \text{ S U M30.7 U E12.1 U M33.3 UN E9.5 U E13.1 U E13.4 U E13.6} & \text{PB014} \\
M33.4 & \text{ R O M32.0 O E0.3 ON T5} & \text{PB014}
\end{align*}
\]

After using the XML to calculate the total number of operators and operands all necessary variables for the Halstead measure are determined, then the related measures of Halstead such as volume (V), difficulty (D) and effort (E) can be evaluated (compare equations in section 3.2). After the calculation of the values for all PLC blocks, the average values of the whole project can be calculated.

The Halstead measure has an upwards open scale. Hence, it is not possible to give a limit value which foresaid when a program is complicated. The information of the average value makes it, nevertheless, possible to ascertain which blocks are exceptionally complicated and thereby hard to perform the diagnosis with. These must be checked and be reworked if necessary.

The Halstead measure for the PLC program of Figure 2 gives: \(N_1 = 18, N_2 = 22, \mu_1 = 6, \mu_2 = 13\), Volume: 17, Difficulty: 5.08, and Effort: 863.08.

4.3. McCabe’s Cyclomatic Complexity

To determine McCabe’s Cyclomatic complexity a flow graph is needed. It is built from the single treatment of the condition equations already described above. At the beginning the operand of each equation is filtered out. For the current operand an assignment is generated which shows the transition of the operand from the state \(n\), before the processing of the block, to the state \(n+1\), after processing.
Starting from this operand a transition directed to the condition of the assignment instruction is build. From this condition another edge is build directing to the assignment instruction. In case this operand is Set or Reset an edge to the Set node or to the Reset node is generated.

If the suitable labels for the setting, resetting or assignment are determined, one more label is generated which negates these conditions and is inserted as a "Label_NOT" between the state $n$ and $n+1$ of the current operand. If a condition is not fulfilled which is defined by this label, the Operand goes unchanged to the subsequent state. After all equations are worked upon, a graph is generated. A part of such a graph for the above example (cf. Figure 2) is shown in Figure 4.

4.4. Tree Impurity

This measure is used to determine the complexity of the calls between the PLC blocks. To be able to perform this, a graph is needed which shows the calls of the distinct blocks. To obtain such a graph, information about the jump terms must be processed from the given block. For this reason the XML of the IL is checked line by line for jump terms. This happens by the inquiry after a non-conditional jump "SPA", and after a conditional jump "SPB".

As an example to explain this concept the OB1 block is taken (cf. Figure 5, top). This block contains a conditional jump and two non-conditional ones. After the processing of this block the jump conditions are stored as in the form shown below:

<table>
<thead>
<tr>
<th>Source</th>
<th>destination: condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>OB001</td>
<td>FB141: U M10.5 U A12.9</td>
</tr>
<tr>
<td>OB001</td>
<td>FB140</td>
</tr>
<tr>
<td>OB001</td>
<td>FB141</td>
</tr>
</tbody>
</table>

The corresponding graph is shown in Figure 5. The tree impurity measure according to the formula given in section 3.4, results in 0 (perfect tree).

5. Case Studies

The approach presented above was successfully applied to a didactic Modular Production System (MPS) from FESTO and an example provided by the company Freudenberg. The PLC programs have been converted and analyzed automatically using JAVA programs implementing the measures presented above. For more details on the case studies see [4]

5.1. FESTO MPS

The system handles 41 binary input and 32 binary output signals. It is controlled by a Siemens PLC programmed in STEP 5 IL. The control software to be analyzed contains 21 modules adding up to a total of 1558 lines of IL code.

<table>
<thead>
<tr>
<th>Value</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-10</td>
<td>An easy program, little risk</td>
</tr>
<tr>
<td>11-20</td>
<td>Complex program, endurable risk</td>
</tr>
<tr>
<td>21-50</td>
<td>Very Complex program, high risk</td>
</tr>
<tr>
<td>&gt;50</td>
<td>Non testable program, extreme high risk</td>
</tr>
</tbody>
</table>

Table 2: Values of cyclomatic complexity.
Table 3 shows the LOC (NCCS) of the FESTO case study. With the help of this measure a first overview about the whole project is obtained. It is to be expected that above all the blocks OB21, PB50 and PB8 are to be considered as the hardest, because they own the most instructions.

The Halstead measure delivers an average difficulty of the blocks of 8.39. This value is used to find out exceedingly complicated blocks (cf. Table 3). The blocks PB50 and PB8 which already have high NCSS value, belong to these. A big difference can be ascertained in case of OB21 which will be explained in details in the next measure (McCabe). The comparison between PB8 and PB50 proves that, in spite of the small difference between the values of NCSS and the volumes of the both, the difficulty measure of PB50 is substantially higher. This signifies that PB50 either owns much more different operators or that the available operands are used substantially more often than in PB8. The blocks where no condition equations can be formed because they contain only jumps or no defined operators and operands are assigned a zero as a result. These blocks are not included in calculating the average value.

<table>
<thead>
<tr>
<th>Block</th>
<th>NCSS</th>
<th>Volume</th>
<th>Difficulty</th>
<th>Effort</th>
<th>McCabe</th>
</tr>
</thead>
<tbody>
<tr>
<td>FB122</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>OB1</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>OB21</td>
<td>177</td>
<td>2864.0</td>
<td>1.0</td>
<td>2864.0</td>
<td>2</td>
</tr>
<tr>
<td>PB1</td>
<td>72</td>
<td>828.0</td>
<td>8.0</td>
<td>6624.0</td>
<td>52</td>
</tr>
<tr>
<td>PB10</td>
<td>6</td>
<td>32.0</td>
<td>2.0</td>
<td>64.0</td>
<td>4</td>
</tr>
<tr>
<td>PB11</td>
<td>5</td>
<td>22.0</td>
<td>2.0</td>
<td>44.0</td>
<td>4</td>
</tr>
<tr>
<td>PB12</td>
<td>6</td>
<td>32.0</td>
<td>2.0</td>
<td>64.0</td>
<td>4</td>
</tr>
<tr>
<td>PB15</td>
<td>6</td>
<td>32.0</td>
<td>2.0</td>
<td>64.0</td>
<td>4</td>
</tr>
<tr>
<td>PB2</td>
<td>104</td>
<td>998.0</td>
<td>15.0</td>
<td>14970.0</td>
<td>11</td>
</tr>
<tr>
<td>PB20</td>
<td>40</td>
<td>357.0</td>
<td>2.0</td>
<td>714.0</td>
<td>6</td>
</tr>
<tr>
<td>PB21</td>
<td>81</td>
<td>1237.0</td>
<td>8.0</td>
<td>9896.0</td>
<td>26</td>
</tr>
<tr>
<td>PB3</td>
<td>20</td>
<td>104.0</td>
<td>1.0</td>
<td>104.0</td>
<td>2</td>
</tr>
<tr>
<td>PB4</td>
<td>56</td>
<td>495.0</td>
<td>9.0</td>
<td>4455.0</td>
<td>19</td>
</tr>
<tr>
<td>PB40</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PB5</td>
<td>6</td>
<td>32.0</td>
<td>2.0</td>
<td>64.0</td>
<td>4</td>
</tr>
<tr>
<td>PB50</td>
<td>209</td>
<td>3281.0</td>
<td>40.0</td>
<td>131240.0</td>
<td>67</td>
</tr>
<tr>
<td>PB55</td>
<td>99</td>
<td>862.0</td>
<td>15.0</td>
<td>12930.0</td>
<td>13</td>
</tr>
<tr>
<td>PB6</td>
<td>97</td>
<td>1261.0</td>
<td>12.0</td>
<td>15132.0</td>
<td>44</td>
</tr>
<tr>
<td>PB7</td>
<td>70</td>
<td>673.0</td>
<td>6.0</td>
<td>4038.0</td>
<td>11</td>
</tr>
<tr>
<td>PB8</td>
<td>261</td>
<td>3363.0</td>
<td>20.0</td>
<td>67260.0</td>
<td>25</td>
</tr>
<tr>
<td>PB9</td>
<td>56</td>
<td>390.0</td>
<td>4.0</td>
<td>1560.0</td>
<td>6</td>
</tr>
<tr>
<td>Average</td>
<td>66.48</td>
<td>936.83</td>
<td>8.39</td>
<td>15115.94</td>
<td>13,82</td>
</tr>
</tbody>
</table>

The rightmost column of Table 3 shows the Cyclomatic complexity of the blocks. Comparing the measures of McCabe to those of the Halstead measure shows that they correspond to each other. This means that the blocks which have delivered an exceptionally high difficulty measure in Halstead, e.g., PB50, are also classified with McCabe as a program with extremely high risk. Additionally, it is shown which blocks own no condition equations.

Comparing the results to the values of LOC, one finds out that not all blocks with the same LOC are identical. Examples for this are the blocks PB4 and PB9. Although they both own the same value of NCSS PB4 is classified as more complicated. A reason for this is that in PB9 operands are not dependent on the results of previous operations as it is the case in PB4.

The biggest difference of the results in LOC and McCabe is to be ascertained for the OB21. The reason for the fact is that in this block all internal variables and counters are reset. A big NCSS number originates from the corresponding instructions; however, because the variables are not dependent on each other, the structure of the flow graph is very simple as shown in the segment of the OB21 (cf. Figure 6).

For the tree impurity the value obtained for the complete project is 0.023. This result shows that the calls of the blocks are barely branched and that this also allows an easy investigation of the distinct blocks. The graph for the tree Impurity is shown in Figure 7.

To sum up: the whole program is not very complicated. Only a few blocks own a Halstead value which is much higher than the average value of the whole program. Also the value of McCabe shows only
5.2. Industrial Example

The system under consideration is a production line which produces non-woven at a plant of the Freudenberg Non-wovens Group in Kaiserslautern (Germany). Part of the non-woven production line is the Winder where the method is applied. The winder is controlled by a PLC of the type Siemens Simatic S5-115U handling a total of 336 binary inputs and outputs signals via digital I/O modules. The control software to be analyzed contains 74 blocks adding up to a total of about 3493 lines of IL code.

The industrial example is bigger and therefore on average the blocks contain more lines of code (higher NCSS value). However, comparing the results of both plants with each other, one finds out that the Halstead measure of the Winder delivers smaller values. This means that the ILs of the Winder are on average easier to understand than those of the FESTO. There are also no values which exceed the average as considerably as in the lab plant. The tree impurity in this example is negative. The negative value makes it clear that the graph of the tree impurity consists of more than one sub-graph. There are a total of three graphs; two of them are quite large but own a pure tree structure delivering a value of zero. The third quite small graph (cf. Figure 8) has a Tree impurity of 0.17.

![Figure 8: Tree Impurity of the Winder.](image)

To sum up the results one should notice that the complexity for doing diagnosis of this industrial IL lies only in the size of the whole program. There are a lot of long blocks which must be investigated.

6. Conclusion and Outlook

To allow easy diagnosis in the case of failures in a production system automated by PLCs, the quality of the PLC software is of crucial importance. In the presented work measures to assess this quality have been identified and adapted to the PLC language Instruction List. As a by-product of the measures calculation several dependence relations can be visualized. It is expected that this will help an engineer in performing the diagnosis task. Case studies show that the method works and provides reasonable results. However, the practical implications still remain to be investigated. To this end an extensive (and expensive) field study would be the logical next step.

References