Semantic-based Analysis of Rules for Decision Support for Software Maintenance

Marek Z. Reformat1*, Petr Musilek1, Aashima Kapoor1, Nicolino J. Pizzi2

(1) Electrical and Computer Engineering, University of Alberta Edmonton, Alberta T6G 1H9, Canada
(2) Computer Science, University of Alberta Winnipeg, Manitoba R3T 2N2, Canada

Abstract
Decision support systems assist users in making suitable and competent decisions regarding variety of issues. They embed knowledge about facts and features related to topics that are subjects of decision-making processes and use this knowledge to identify most appropriate alternatives. Representation of knowledge could take multiple forms. Quite often, if-then rules are used, as they are perceived as one of the easiest ways of capturing knowledge. In general, multiple methods can be employed to construct rules – some of them are fully depended on the experts’ knowledge, some on available data, yet some on a combination of both approaches.

The paper introduces a methodology for identifying most suitable and representative if-then rules. Semantic-based analysis of these rules is described. All rules are evaluated based on their classification performance, as well as their ability to represent knowledge. This constitutes a step towards an automatic construction of rule-based decision-making systems.
Software maintenance is a very effort-intense phase in a software life cycle. It focuses on keeping already released software fully functional and up to date. Maintenance engineers use different methods to gain understanding of software from the point of view of required maintenance efforts. That allows them to make proper decisions regarding maintenance tasks associated with different software objects. In the paper, numerous rules are extracted from a data set representing human evaluation of maintainability of software objects using different rule-generating tools. We analyze rule-based descriptions of software objects requiring different levels of maintainability. A set of selected rules is used to construct a decision support system for software maintenance activities.

**Keywords:** If-then rules, decision support, software maintenance, semantic comparison.

1 Introduction

The cost of maintenance activities accounts for more than half of the typical software budget [16, 31]. This fact alone means that any effort leading to increase effectiveness of maintenance activities is essential for any software developing company. Application of decision-making tools that support maintenance tasks is important, while a good understanding of what makes software more or less maintainable is of profound significance.

There is an ongoing research focused on finding a single metric that estimates level of maintainability of software. The most popular one is the SEI Maintenance Index (MI) built based on the Coleman-Oman model [9, 30]. The MI is calculated using a combination of common software measures:

\[
MI = 171 - 5.2 \times \ln(\text{aveV}) - 0.23 \times \text{aveV}(g) - 16.2 \times \ln(\text{aveLOC}) + 50 \times \sin(2.4 \times \text{perCM})
\]  

(1.1)

where **aveV** is the average Halstead Volume per module, **aveV(g)** is the average extended cyclomatic complexity per module, **aveLOC** is the average lines of code per module, and **perCM** is the average percent of lines of comment per module. The coefficients of the equation are the result of calibration using data from numerous software systems being maintained by Hewlett-Packard [10]. In a major research effort due to Hewlett Packard, the following thresholds for the evaluation of the MI, calculated by means of the previous models, have been determined: **MI < 65** for poor maintainability, **65 < MI < 85** for fair maintainability, and **85 < MI** for excellent maintainability.

Although, the concept of MI is interesting, an approach based on the human estimation of maintenance levels is an appealing alternative to a formula-based evaluation. In such a case, programmers or maintenance engineers are asked to evaluate maintainability of software objects via visual inspection. This kind of maintenance estimation is influenced by the person’s individual experience and background. However, it is very motivating to perform such evaluation in order to better understand what aspects of software programming are critical for deciding about excellent/fair/poor levels of maintainability of software. In other words, we would like to find out what attributes of software objects are critical/influential for individuals when they identify maintainability levels of software objects, and what attributes should be used to develop decision-making tools.

The issues presented above motivated us to focus on a thorough analysis of maintenance data in order to identify the most essential factors defining excellent/fair/poor levels of maintainability. The approach presented in the paper is the result of that activity. The core element of the approach is extraction of IF-THEN rules from data and semantic-based analysis of those rules.
The extraction of rules from data in a well-known research topic \([2, 5, 6, 7, 15, 19, 22, 23, 24, 26, 27, 33]\), and it is not covered in this paper. We relay on existing rule-generating tools. Availability of numerous tools led to one of the challenges that the proposed approach addresses – dealing with large number of different rules constructed by variety of rule-generating tools. Each of these tools ‘looks’ at data differently and each of them generates different IF-THEN rules. The proposed approach is designed in a way that a large variety of rules leads to even more interesting results.

In the paper, we propose a novel methodology for analyzing rules extracted from data. The analysis process targets such issues as: good coverage, similarity of rules, and inclusion of rules. Similarity and inclusion are not preformed at the level of syntax, but at the level of semantics of rules. It means that we compare the rules based on data points cover by those rules.

We use the proposed method to analyze rules extracted from software maintenance data. This software maintenance data contain evaluation of maintainability levels of software objects preformed independently by three software engineers. In the first experiment, we analyze the obtained data in order to find out how each evaluator ‘recognizes’ three levels of maintainability: excellent, fair, and poor. Additionally, we compare these findings, identify software attributes that evaluators implicitly recognize as important, and find out if there is an agreement among engineers in identifying objects of different levels of maintainability.

In the second experiment, we analyze rules representing a single engineer constructed using different rule-generating tools and compare findings in order to find generic rules describing an individual.

The paper is organized in the following way. Section 2 is dedicated to a very briefly description of analysis of maintenance data. The concepts of similarity and inclusion of rules are presented in Section 3. A detailed description of maintenance data used here is included in Section 4. Section 5 explains the approach we used to select the most important rules and compare them. Section 6 is dedicated to the description of our system for automatic comparison of rules, and identification of the most unique rules. Sections 7 and 8 contain the description of main rules and all findings related to different levels of maintainability of software objects done for each programmer, and for different rule-generating tools, respectively. The conclusions constitute Section 9.

2 Software Maintenance: Data Models and Measures

An importance of software maintenance is reflected in a number of research activities dedicated to this topic. The two activities related to this paper are modeling of maintenance data and development of a maintenance measure.

Data models representing different software maintenance activities are used for many years \([14, 21, 28]\). One of the first such papers dedicated to this topic was \([20]\). This paper reported on the development and use of several software maintenance models that were built applying regression analysis, neural networks, and the optimized set reduction method. The variables included in the models were: a cause of task, a degree of change on a code, a type of operation on a code, and confidence of maintainer. An explanation of efforts associated with software changes made to correct faults while software is undergoing development was investigated in \([12, 13]\). In this case, the ordinal response models were developed to predict efforts needed to isolate and fix a defect. The model input variables included extent of change, a type of change, an internal complexity of the software components undergoing the change, as well as fault locality and characteristics of the software components being changed. A model for estimating adaptive software maintenance efforts in person hours was described in \([18]\). It was found that a number of metrics such as the lines of code changed, and the number of operators changed are strongly correlated to maintenance efforts.

Another maintenance related research is focused on attempts to measure maintainability. Great effort has
been put into constructing formulas for describing maintainability. Following the opinion that maintainability is the set of attributes, these formulas describe maintainability as a function of directly measurable attributes. Many researchers have tried to quantify maintainability in different types of measures [1, 3, 25], of which the most noticeable is the Maintainability Index, MI [30]. The Halstead source code measures (Halstead Length, Halstead Volume, Halstead Effort) proposed in the seventies [17, 29] have been also used for describing maintainability [29, 30].

To the best of authors’ knowledge, there is no reported work on the topic of using methods of data modeling to describe what maintainability means for humans.

3 Similarity and Inclusions of Rules

A rule generation process can be performed using a number of approaches and methods. Each of them leads to generation of different rules. The rules differ in a number of ways. If we assume that an atomic component is built in the following way: value of attribute \( a \) is larger than number \( n \), then we can identify a variety of differences among rules: rules can have different number of atomic components in antecedents, as well as different attributes and their values. All these differences are related to the syntax of rules. We would like to focus on a different kind of comparison of rules – the semantic similarity. This similarity ‘looks inside’ rules and compares them based on a number and identity of points that satisfy the antecedent parts of rules.

3.1. Tversky’s Similarity Model

The two main approaches for assessing similarity are content models and distance models. Content models conceptualize the characteristics with respect to which objects are similar “as more or less discrete and common elements” [4]. Distance models conceptualize these characteristics “as dimensions on which the objects have some degree of proximity” [4]. Many set theoretic measures in the content model category are generalized by Tversky’s parameterized ratio model of similarity [32]. It expresses similarity between objects as a ratio of the measures of their common and distinctive features:

\[
S(X, Y) = \frac{f(X \cap Y)}{f(X \cap Y) + \alpha f(X - Y) + \beta f(Y - X)}
\]  

(3.2)

\( X \) and \( Y \) represent sets describing respective objects \( x \) and \( y \). \( (X \cap Y) \) represents the features that objects \( x \) and \( y \) have in common. \( (X - Y) \) represents the features that \( X \) has but \( Y \) does not. \( (Y - X) \) represents the features that \( Y \) has, but \( X \) does not. The function \( f \) measures the contribution of any specific feature to the value of similarity between objects. The value \( f(X) \) for object \( x \) is considered a measure of the overall salience of that object. Factors adding to an object’s salience include “intensity, frequency, familiarity, good form, and informational content” [32]. The function \( f \) is additive on disjoint sets, for example, set cardinality. The factors and are nonnegative valued unbounded parameters specifying the importance of these two components. This measure is normalized, \( 0 \leq S(X, Y) \leq 1 \). For \( \alpha = 1 \), \( S \) becomes the Jaccard index:

\[
S - \text{jaccard}(X, Y) = \frac{f(X \cap Y)}{f(X \cup Y)}
\]  

(3.3)

With \( \alpha = 1, \beta = 0 \), \( S \) becomes the degree of inclusion for \( X \), i.e., the proportion of \( X \) overlapping with \( Y \).
S - inclusion(X, Y) = \frac{f(X \cap Y)}{f(X)} \tag{3.4}

Similarly with \( a = 0, \ b = 1 \), S becomes the degree of inclusion for Y, the proportion of Y overlapping with X. This parameterization is not necessary, however, since Eq. 4 can be formulated as \( S\text{-inclusion } (Y, X) \).

3.2. Tversky’s Model for Similarity & Inclusion of Rules

Presented Tversky’s model of similarity can be easily applied for comparison of rules. In order to compare the rules semantically the contingency table is used. An example of such a table is presented below, Table 1.

<table>
<thead>
<tr>
<th></th>
<th>R1</th>
<th>no R1</th>
</tr>
</thead>
<tbody>
<tr>
<td>R2</td>
<td>a</td>
<td>c</td>
</tr>
<tr>
<td>no R2</td>
<td>b</td>
<td>d</td>
</tr>
</tbody>
</table>

The number \( a \) represents a number of data points that are satisfied by antecedents of both rules R1 and R2. The number \( b \) identifies a number of data points that satisfy the rule R2 but do not the rule R1, while the number \( c \) opposite – a number of points that satisfy the rule R1 but do not R2. The numbers \( a, b \) and \( c \) can be directly plug in into the Eq. 3. The equation is presented below:

\[
S \text{-jaccard}(R1, R2) = \frac{a}{a + b + c} \tag{3.5}
\]

Tversky’s model allows us to calculate inclusions of rules. The formulas can be presented in the following forms:

\[
S \text{-inclusion}(R1 \text{ in } R2) = \frac{a}{a + b} \tag{3.6}
\]

\[
S \text{-inclusion}(R2 \text{ in } R1) = \frac{a}{a + c} \tag{3.7}
\]

4 Data Description and Experimental Setup

The data used in the paper were collected during an experiment conducted in National Research Council (NRC), Canada. In the experiment, three software engineers have independently analyzed software objects of the system EvIdent® (is a user-friendly, algorithm-rich, graphical environment for the detection, investigation, and visualization of novelty in a set of images as they evolve in time or frequency. EvIdent® is written in Java and C++ and is based upon VIStA, an application-programming interface (API) developed at the National Research Council).

The aim of that evaluation was estimation of easiness of performing maintenance tasks on software objects. For each of the 366 software objects, three software engineers, named here ‘A’, ‘D’ and ‘V’, were asked to independently rank easiness of objects’ maintainability in the scale from 1 to 5 (where 1 means \textit{POOR maintainability}, 2 means \textit{POOR-FAIR maintainability}, 3 means \textit{FAIR maintainability}, 4 means \textit{FAIR-EXCELLENT maintainability}, and 5 means \textit{EXCELLENT maintainability}). The engineers determined maintainability of objects based on their own judgment.

At the same time, 64 software metrics were calculated for each software object. As the result, the collected data set consists of 366 data points represented by a set of 64 software metrics and three values assigned to each object by three engineers. Some of the extracted metrics are shown in Table 2.
<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>GUI (=1), Data Model (3.2), Algorithm (3.3), Other (3.4).</td>
</tr>
<tr>
<td>METH</td>
<td>Number of Methods</td>
</tr>
<tr>
<td>LOC</td>
<td>Number of lines of code.</td>
</tr>
<tr>
<td>ALOC</td>
<td>Mean LOC per method.</td>
</tr>
<tr>
<td>RCC1</td>
<td>Ratio of comment lines of code to lines of code including whitespace &amp; comments.</td>
</tr>
<tr>
<td>MTOK</td>
<td>Median TOK (# tokens) per method</td>
</tr>
<tr>
<td>DEC</td>
<td>Number of decisions.</td>
</tr>
<tr>
<td>WDC</td>
<td>Weighted measure of the number of nested decisions for an object (the greater the nesting the greater the associated weight).</td>
</tr>
<tr>
<td>MWDC</td>
<td>Median WDC per method.</td>
</tr>
<tr>
<td>DINH</td>
<td>Depth of inheritance.</td>
</tr>
<tr>
<td>CHLD</td>
<td>Number of children.</td>
</tr>
<tr>
<td>SIBL</td>
<td>Number of siblings.</td>
</tr>
<tr>
<td>RCR</td>
<td>Code reuse: ratio of overloaded inherited methods to those that are not.</td>
</tr>
<tr>
<td>CBO</td>
<td>Coupling between objects – Number of other objects to which the corresponding object is coupled.</td>
</tr>
<tr>
<td>RFO</td>
<td>Response for an object. Response set contains the methods that can be executed in response to a message being received by the object.</td>
</tr>
<tr>
<td>MNL1</td>
<td>Maximum method name length.</td>
</tr>
<tr>
<td>MNL3</td>
<td>Mean method name length.</td>
</tr>
<tr>
<td>ATCO</td>
<td>Attribute Complexity.</td>
</tr>
<tr>
<td>DAC</td>
<td>Data Abstraction Coupling - Number of reference types used in the attribute declaration.</td>
</tr>
<tr>
<td>HLDF</td>
<td>Halstead difficulty.</td>
</tr>
<tr>
<td>HLPL</td>
<td>Halstead program Length.</td>
</tr>
<tr>
<td>HLVC</td>
<td>Halstead Program Vocabulary.</td>
</tr>
<tr>
<td>HLVL</td>
<td>Halstead Program Volume.</td>
</tr>
<tr>
<td>HLOR</td>
<td>Halstead # operators.</td>
</tr>
<tr>
<td>HLUN</td>
<td>Halstead # unique operands.</td>
</tr>
<tr>
<td>HLUR</td>
<td>Halstead # unique operators.</td>
</tr>
<tr>
<td>MIC</td>
<td>Method Invocation Coupling.</td>
</tr>
<tr>
<td>MAXQ</td>
<td>Maximum size operations.</td>
</tr>
<tr>
<td>ATTR</td>
<td>Number of attributes.</td>
</tr>
<tr>
<td>CONS</td>
<td>Number of constructors.</td>
</tr>
<tr>
<td>IMST</td>
<td>Number of Import statements.</td>
</tr>
<tr>
<td>OVRM</td>
<td>Number of overridden methods.</td>
</tr>
<tr>
<td>REMM</td>
<td>Number of various remote method calls.</td>
</tr>
<tr>
<td>PKGM</td>
<td>% Package members in a class.</td>
</tr>
<tr>
<td>WMC2</td>
<td>Weighted methods per class.</td>
</tr>
</tbody>
</table>
The graph in Figure 1 shows the distribution of data points identified as 1, 2, 3, 4 or 5. The graph clearly shows that there are very few data points classified as 1 (POOR maintainability). Consequently, it would be difficult to generate rules for such a small data subset. In other words, rules generated for 1 will be specific to these data points and will lack generality.

In order to create rules that are more general, we have processed the data. The data points were grouped together in a way to increase uniform distribution of data points over categories (scale values). The data points classified as 2 earlier were added to the group 1, and data points classified as 4 earlier were added to the group 3.

In all, maintainability of software objects is classified as 1, 3 or 5 (where 1 means POOR maintainability, 3 means FAIR maintainability, 5 means EXCELLENT maintainability). Such division into 3 categories is used in all experiments presented in the paper.

![Maintainability Chart](image_url)

Figure 1: Distribution of maintainability data

5 Rule Analysis Methodology

5.1. Concept

The purpose of data analysis is to discover relations describing phenomena represented by data, and further use this knowledge to support decision-making processes. One of the methods commonly used for data analysis is extraction of IF-THEN rules. In such a scenario, we are interested in obtaining useful rules, i.e., rules that have good coverage, good classification capabilities, and are unique. Importance of these aspects depends on intended utilization of extracted rules. For example, we can be interested in obtaining a diversified set of rules in order to discover multiple views of relationships existing among the same data points.

There are multiple techniques and tools that are used to extract IF-THEN rules from data. Each of these methods ‘sees’ data differently and generates different rules. The difference in rules is judged based on their visual inspection. This means that rules are different syntactically. At that point a number of interesting questions could be asked: How these rules are different? Do they ‘cover’ different data points? Can they be used together, for example, for prediction purposes?

Mentioned above issues – finding useful, similar or different rules – led us to taking a deeper look at rules. This section is focused on describing a proposed methodology for semantic-based analysis of rules. We use this approach to identify the most interesting rules based on two characteristics of rules: how good a given rule is; and how unique that rule is among all rules extracted from data.
The process of finding good rules can be done in a number of different ways. The most common one is a method that uses measures representing performance of rules (see Appendix for some possible metrics useful for rule performance evaluation). The evaluation metric applied in this paper is a simple measure called the Laplace ratio (Eq. 8, Appendix).

A method that can be applied to find unique rules is proposed in this paper. It is based on comparison of rules using concepts of similarity and inclusion. These two concepts are determined at a semantic level. The term semantic is used here because comparison is performed based on data points covered by rules. It means that values of similarity and inclusion are obtained by applying equations 5, 6, and 7 with numeric values representing a number of data points exclusively and mutually covered by rules. The methodology is described below.

5.2. Methodology

The process of semantic-based analysis of rules is divided into a number of steps. The algorithmic representation of the methodology is shown below:

0 generate sets of rules using different rule-generating tools

1 for each set of rules do

1.1 for each category (for each unique value from consequences of rules) do

1.1.1 calculate rule performance measures

1.1.2 select rules with the highest values of measures

1.1.3 evaluate similarity of rules

1.1.4 evaluate inclusion of rules and identify unique rules – the rules that are not similar to others, and the rules that are not included in other rules

end

1.2 gather all unique rules for each category

end

2 combine rules for each category that have been selected from sets of rules generated by different tools

3 for each category do

3.1 analyze rules (similarity and inclusion)

3.2 identify unique rules

end

4 gather all unique rules for each category

The very first step (Step 0) involves utilization of rule-generating tools. There are multiple tools that can be used to generate IF-THEN rules. There are no any special requirements that these tools should satisfy, the only thing that is needed is the ability to create rules. Description of such tools is outside the scope of this paper. Here, we use two rule-generating tools: C5.0/See5 and 4cRuleBuilder (The C5.0/See5 tool is an updated version of the well-known C4.5 algorithm [27, 34]. It has capability of generating classifiers that are expressed as decision trees or sets of IF-THEN rules. The commercially available tool called 4cRuleBuilder [35] is used to build the second data model. This tool uses supervised learning to generate a data model from discrete numerical or nominal data, and has build-in descretization schemes for continuous attributes).
Tasks that constitute other steps are specific to the proposed approach and are described below. Step 1.1.1 embraces activities related to evaluation of rules based on their ability to perform a proper classification. In such a case, any technique that allows us to determine how good a given rule is can be applied (Appendix). The evaluation process is performed on a set of data points. Each rule, created by any rule-generating tool, is checked against that set. Therefore, comparison and analysis of rules is sound, i.e., all rules ‘see’ the same data points, and calculated ‘rule goodness’ measures are comparable.

After goodness of rules is determined, the next step is to select the best rules, Step 1.1.2. The term best refers to rules with the highest scores obtained during their evaluation. That step requires setting up a parameter. A threshold value has to be identified in order to distinguish best rules from all other rules. The value of this threshold depends on a type of rule performance measure used in the Step 1.1.1. The threshold value should be set up specifically for it. It is possible that the value has to be adjusted during the analysis process. One of the possible reasons for that adjustment is concern about a number of best rules. If the threshold value is set too high and all rules have values of performance measures less than it – the threshold value could be reduced to ensure that some rules are selected for further analysis.

Once the best rules are identified, a comparison process of the best rules takes place – Steps 1.1.3 and 1.1.4. Equations 5, 6 and 7 are used for identifying sets of different and similar rules representing a phenomenon under investigation. The procedure consists of two phases:

- first phase (Step 1.1.3) is dedicated to identifying rules that are semantically similar to each other (Eq. 5); calculation of similarity measures for all pairs of rules is performed – the pairs with values of similarity measure larger than specified by a user are considered for removal (a value of a threshold for identifying similar rules depends on application – a user can indicate that he/she is interested only in a perfect similarity (1.0), or he/she can tolerate deviation up to 10-20% from a perfect similarity);
- the second phase (Step 1.1.4) is focused on selecting rules that will be removed; this process is performed based on inclusion measures (Eqs. 6 and 7); for each rule identified as considered for removal (previous phase) values of inclusion of this rule in other rules are calculated; rules with the highest values of inclusion measure are removed, (once again a value of inclusion threshold is set up based on user’s preferences regarding deviation form a perfect inclusion (1.0)).

The steps described above (from 1.1.1 to 1.1.4) are repeated for each category, and for each set of generated rules. Step 2 is a simple process of combining all rules for a single category into one set. Each set contains unique rules that have been selected across rules constructed for one category by all tools.

The next two steps: Step 3.1 and Step 3.2 are repetitions of Steps 1.1.3 and 1.1.4 but this time they are performed for all previously selected rules for a single category without distinguishing rule-generating tool.

As the final result, Step 4, we obtain rules (for each category) that are the most unique across all rules constructed by rule-generating tools.

5.3. Example
To illustrate the proposed methodology a simple example is presented. The rules shown are randomly picked and hence do not represent the exact results from any of the experiments presented later in the paper.

Let D be the dataset under consideration. D has 366 data points with 64 attributes \(\{a1, a2, a3, a4, \ldots, a64\}\), and two categories namely Class 0 and Class 1. C5.0/See5 was used to generate rules. The rules generated are presented in Figure 2.
R1: (38/3, lift 5.7)
\[ a_{41} > 963 \rightarrow \text{class 0 } [0.900] \]

R5: (68/4, lift 2.4)
\[ a_2 \leq 3 \]
\[ a_{14} \leq 11 \]
\[ a_{27} \leq 21 \]
\[ a_{64} \leq 5 \]
\[ \rightarrow \text{class 1 } [0.929] \]

R2: (25/2, lift 5.6)
\[ a_2 > 26 \]
\[ a_{14} > 11 \]
\[ a_{52} \leq 1 \]
\[ \rightarrow \text{class 0 } [0.889] \]

R6: (160/28, lift 2.2)
\[ a_{37} \leq 142 \]
\[ \rightarrow \text{class 1 } [0.821] \]

R3: (29, lift 6.1)
\[ a_3 > 819 \]
\[ a_{39} > 11253 \]
\[ \rightarrow \text{class 0 } [0.968] \]

R7: (75/2, lift 2.5)
\[ a_4 \leq 2 \]
\[ a_{33} \leq 1 \]
\[ a_{37} \leq 142 \]
\[ \rightarrow \text{class 1 } [0.961] \]

R4: (21, lift 6.0)
\[ a_{31} > 52 \]
\[ a_{39} > 11253 \]
\[ \rightarrow \text{class 0 } [0.957] \]

R8: (135/10, lift 2.4)
\[ a_{27} \leq 24 \]
\[ a_{37} \leq 142 \]
\[ a_{43} \leq 11 \]
\[ \rightarrow \text{class 1 } [0.920] \]

R9: (26/2, lift 1.4)
\[ a_{20} \leq 6 \]
\[ a_{21} > 0 \]
\[ a_{27} \leq 20 \]
\[ a_{36} \leq 23440 \]
\[ \rightarrow \text{class 1 } [0.649] \]

Figure 2: Set of rules generated using C5.0/See5

C5.0/See5 also calculates rule performance measures (Step 1.1.1) – the values of Laplace ratio are given together with rules. In the next step, Step 1.1.2, we need to set up a threshold value. For the example, the value of threshold is equal to 0.7. In general, the threshold value should be ‘tuned’ for each performance measure and each rule-generating tool. This should be done by careful examinations of the ranges of performance measure values.

All rules for the category Class 0 (R1, R2, R3 and R4) have high values of the Laplace ratio, thus all rules qualify for further analysis and interpretation. For the category Class 1, the last rule has the Laplace value of

---

1 The rule R1 reads as: if attribute a41 > 963 then category predicted is Class 0. Additionally, the numbers in brackets 38/3 mean: 38 – is a total of classified points, 3 – is a number of data points that were misclassified. The Laplace ratio is shown at the end of each rule in square brackets [0.900].
that is below the threshold, and the rule R9 needs to be discarded. We are left with 4 rules for Class 0, and 4 rules for Class 1.

Now, we need to compare rules within each category, Steps 1.1.3 and 1.1.4. In other words, we need to select rules on the basis of their importance and uniqueness for representing each category. In the end we will be left with best rules for Class 0 and Class 1 respectively.

We start the process by building a contingency table for rules to be compared, Table 3. How to build this table is explained in Section 3.2.

Table 3: Values of Similarity and Inclusion

<table>
<thead>
<tr>
<th>rules: R’, R”</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>S(R’,R”)</th>
<th>Inc(R’-&gt;R”)</th>
<th>Inc(R”-&gt;R’)</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1, R2</td>
<td>17</td>
<td>21</td>
<td>8</td>
<td>366-(17+21+8)=320</td>
<td>0.369</td>
<td>0.447</td>
<td>0.680</td>
</tr>
<tr>
<td>R1, R3</td>
<td>29</td>
<td>0</td>
<td>9</td>
<td>366-(29+0+9)=328</td>
<td>0.763</td>
<td>0.763</td>
<td>1.000</td>
</tr>
<tr>
<td>R1, R4</td>
<td>19</td>
<td>2</td>
<td>19</td>
<td>366-(19+2+19)=326</td>
<td>0.475</td>
<td>0.500</td>
<td>0.900</td>
</tr>
<tr>
<td>R2, R3</td>
<td>17</td>
<td>12</td>
<td>8</td>
<td>366-(17+12+8)=329</td>
<td>0.459</td>
<td>0.680</td>
<td>0.580</td>
</tr>
<tr>
<td>R2, R4</td>
<td>8</td>
<td>13</td>
<td>17</td>
<td>366-(8+13+17)=328</td>
<td>0.210</td>
<td>0.380</td>
<td>0.320</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

a - # of points covered by both rules; b – # of points covered exclusively by R’; c - # of points covered by R” only, S – stands for similarity, Inc – for inclusion.

For each of these pairs, we plug in the values a, b, c, and d into equations 5, 6, and 7 to calculate the similarity (S(R’,R”)), and inclusion between each pair of rules (Inc(R’->R”), Inc(R”->R’)). The results for some pairs are shown in Table 3. The purpose is to keep rules that are important in terms of defined ‘goodness’ measure, the Laplace ratio in our case, and uniqueness. The pairs with high semantic similarity value are the ones that will be considered for further analysis.

For example, the pair of rules R1 and R3 shows a semantic similarity of about 76% with each other. It means that the chances of one rule being redundant or inclusive in the other are high. Rule inclusion values clearly show that R3 is 100% included in R1. In other words, all the data points that are classified by R3 are also classified by R1. Thus, there will be no coverage degradation if R3 is discarded because all data points covered by R3 are being covered by R1. On the other hand, these two rules are interesting because they are different syntactically but yet cover the same 29 data points. It bring an interesting observation that if a point has a41>963 then the same point has a3>819, and a39>11253.

For the next pair, we notice 90% inclusion of R4 in R1 even if the similarity of those two rules is not so high. In such case, it depends on a domain expert if he/she is willing to take some loss of coverage if he/she chooses to remove R4. Next the two pairs – (R2, R3) and (R2, R4) show fair and low ranges of similarity and inclusion, respectively.

6 System for Semantic-based Rule Analysis

6.1 Motivation and Objectives

Extraction of IF-THEN rules from data is of significant importance for discovering important/interesting relationships existing among different data attributes. Therefore, we decided to develop a system that could perform the whole task automatically, using multiple rule-generating tools, provide high consistency of
results, use less time, and handle large datasets. The developed system is called the SSACR – the System for Semantic-based Analysis and Comparison of Rules.

The following objectives were taken into account during a design process of the SSACR:

- ability to handle different rule-generating tools, this means capability to evaluate rules generated by different tools and represented using different formats;
- ability to evaluate different rules (generated by different tools, above) against the same set of data points, so a uniform comparison of rules is possible – this is necessary for semantic-based analysis of rules;
- ability to handle different splits of data, it means that generation and evaluation of extracted rules can be done based on different subsets of data;
- ability to alternate actions of the SSACR in a simple way, this means that there is a simple way of modifying a rule analysis process – additions, replacements, and deletions of different stages and steps.

6.2. System Architecture and Design

Presented above requirements led us to the selection of an architectural pattern called the blackboard [8] as the architectural framework of the SSACR. That pattern is suited to address issues of information sharing, flexibility in performing different tasks, as well as modifiability and extendibility of a system itself. Blackboard systems were designed to resolve complex Artificial Intelligence (AI) problems. The blackboard system works based on the following metaphor [11]:

“Imagine a group of human specialists seated next to a large blackboard. The specialists are working cooperatively to solve a problem, using the blackboard as the workplace for developing the solution. Problem solving begins when the problem and initial data are written onto the blackboard. The specialists watch the blackboard, looking for an opportunity to apply their expertise to the developing solution. When a specialist finds sufficient information to make a contribution, she records the contribution on the blackboard, hopefully enabling other specialists to apply their expertise. This process of adding contributions to the blackboard continues until the problem has been solved”.

There are a number of features that make this architectural pattern suitable for solving complex problems. The blackboard systems are flexible in selection of steps that should be taken to accomplish a specific task. They can be defined as repositories of solutions and contributions to the current problem that are repeatedly updated. The blackboard architecture consists of three major components:

- The specialized modules, which are build by the experts to provide specific expertise needed by application systems.
- The blackboards, which contain all the data, problem statements, updated solutions and any contributions leading towards a solution. The blackboards are being updated and watched continuously.
- Control unit that controls the flow of problem-solving activities. This can be seen as a way to organize the use of data in other a flow of data and intermediate solutions among working units in effective and coherent manner.

When compared with the methodology presented in Section 5, the SSACR has ability to utilize multiple data splits. This concept has been introduced to address the issue of generalization of rules. Multiple splits allows for application of different splits for generation and evaluation of rules. For example, a dataset can be divided into three subsets: training, validation, and testing. The training subset is used to generate rules, the
validation subset is used to evaluate them, and the testing subset is left to evaluate rules that have been selected through an analysis process. Such an approach leads to selection of more generic rules.

Figure 3 below represents the blackboard architecture of the SSACR. A number of blackboards are used to ease the readability of the solution. The blackboards are split on the basis of the type of information they contain:

- **data_blackboard** includes datasets under consideration; every time we run a system for a different dataset, a new data are copied into the data_blackboard replacing the existing dataset;
- **results_blackboard** contains the results obtained for a given dataset; it also contains all intermediate results (classification results of all rules) necessary for performing the final semantic-based analysis of rules;
- **goal_blackboard** is a blackboard that controls all tasks involved in generation, evaluation and analysis of rules;
- **main_config** contains basic information about data files and their formats, as well as rule-generation tools that are used.

The **Control Unit** is in charge of the whole analysis process, and supervises execution of individual steps/activities. It is responsible for calling different specialized modules in an ordered fashion. It may also be defined as a central control component that evaluates the current state of processing and coordinates activities of specialized modules [8].

The **Specialized Modules** form the main structure of the SSACR blackboard architecture. These modules are used collectively to select the best rules among large number of rules generated by rule-generating tools. These modules are built as independent units. Following is the description of the major modules of the SSACR:

- **write_xyz_data** – this module is responsible for reading the datasets - training, validation, and/or test dataset – and writing them into the data_blackboard. A version of this module – with a different name – exists for each dataset. For example: write_val_data will deal with the validation data, and similarly the write_test_data will deal the test dataset.
- **RuleBuilders** – this module converts rules into a C code. Every rule-generating tool generates rules in a specific format. Therefore, there is a need to translate these different formats into a generic format that will be used by the SSACR. Each RuleBuilder module takes as its input a file with rules generated by a specific rule-generating tool and creates based on this file a C program containing these rules as conditional statements. In other words, each RuleBuilder module “translates” IF-THEN rules into a program. A generated C program is then compiled and used as dynamic library. The main reason for such a solution is to efficiently evaluate rules – once they are transformed into a program they can be used to classify any data points. Therefore, rules are compared based on coverage calculated using the same data sets.
- **fire_rules** – this module implements one of the important steps in the rule analysis process. The fire_rules module picks a single data point, “takes” a rule form a respective dynamic library, and “fires” the rule. The result is <ruleID, category> – the information identifying a rule has been fired <ruleID, …>, and the category predicted by that rule <…, category>.
- **compare_rules** – this module performs the most important step in the process of analysis of rules. Each rule is semantically compared with all the other rules of the same category. They are compared based on their goodness, similarity and inclusion as discussed in Section 5. Rules that are similar to other rules and the rules that are included in others are marked for removal.
The SSACR is implemented in Python 2.4, and Extensible Markup Language (XML) is used as the format for storing data, results, and control information in blackboards.

7 Maintainability Data – Individual Programmer Approach

The experimental section of the paper describes analysis of software maintenance data representing individual software engineers. In the first experiment, we extract rules representing (modeling) each engineer and find out commonalities among these rules. These activities illustrate a process of identifying most suitable rules for transparent – white-box type – decision-making processes.

![Blackboard Architecture](image)

Figure 3: Blackboard Architecture

All 366 data points that belong to three categories 1 (POOR), 3 (FAIR), and 5 (EXCELLENT) are used for generation of rules (see Section 4 for details). Two experiment scenarios are defined to generate rules for each category:

a. **all-category scenario:**
   rules for all three categories – 1, 3, 5 – are generated in one run of the rule-generating tool (The rule generation tool used in this section is C5.0/See5); the input file is a single data file consisting of data points identified as 1, 3 or 5; the rules for each category 1, 3 and 5 are generated;

b. **two-category scenario:**
   the original data file is used to create three data files: each new file consists of data points that “belong” to two groups – a group representing one of the categories 1, 2 or 3, and a group of points that do not belong to the category 1, 2, or 3; for example a data file for category 1 consists just two categories – 1 and 0 (the data points identified as 3 or 5 in original file are now points that belong to the category 0).

Once multiple rules are generated for each category and for each software engineer, we select the best rules. We focus only on two categories Poor Maintainability and Excellent Maintainability. The goal is to identify just two rules for each engineer, each category. Therefore, we do not specify any specific threshold values (Section 5.3). We want to emphasize the fact that these thresholds are flexible, and it is up to a domain expert to decide on their values. Those values depend on expert’s needs and his/her ability to tolerate imperfections in similarity and inclusion of rules.
7.1 Poor Maintainability

7.1.1. Software Engineer V

The software engineer V indicated that 58 objects (15.8 % out of 366) are of Poor Maintainability. The application of both scenarios for generating rules results in two rules for the all-category scenario, and two rules for the two-category scenario. The values of Laplace ratios for the rules are in the range of 0.88 to 0.90 for rules generated by all-category scenario, and 0.95 to 0.97 for rules generated by two-category scenario. The initial selection of rules is done solemnly on the basis of “goodness” of rules. From all these rules, the rules with the Laplace ratio above 0.9 are selected: a single rule V-L-1A (The letter A indicates that the rules with the letter were generated using all-category scenario) from the first set, and two rules V-L-1 and V-L-2 from the second set. All three rules are shown below.

Rule$_{V-L-1A}$ (35, 5, 0.900$^2$):

IF HLOR > 963  
THEN category: poor_maintainability

Rule$_{V-L-1}$ (29, 0, 0.968):

IF LOC > 819 & HLVL > 11253  
THEN category: poor_maintainability

Rule$_{V-L-2}$ (19, 2, 0.957):

IF ATCO > 52 & HLVL > 11253  
THEN category: poor_maintainability

The inspection of these three rules leads to the conclusion that the rule V-L-1 is the best. It properly classifies 29 points and does not misclassify any points. A contingency table for all three rules is built, Table 4. It can be observed what the rules V-L-1 and V-L-1A are the most similar ones.

Table 4: Contingency table for Poor_Maintainability rules representing the software engineer V

<table>
<thead>
<tr>
<th>Pair of rules (R1, R2)</th>
<th>Contingency table entries</th>
<th>Similarity measure</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>(V-L-1, V-L-1A)</td>
<td>29</td>
<td>0</td>
</tr>
<tr>
<td>(V-L-2, V-L-1A)</td>
<td>19</td>
<td>2</td>
</tr>
<tr>
<td>(V-L-1, V-L-2)</td>
<td>13</td>
<td>16</td>
</tr>
</tbody>
</table>

High similarity of rules V-L-1 and V-L-1A means that one of them can be redundant and removed from the set of rules. The inclusion measures are calculated for V-L-1 and V-L-1A in reference to the other two rules, Table 5. The values of inclusion measures indicate that the rule V-L-1 is fully included in the rule V-L-1A. This means that the rule V-L-1 can be removed from the set of rules without any degradation of coverage. Overall, two rules V-L-1A and V-L-2 (rules which are underlined) are used to represent the software engineer V. The first rule indicates that if the number of operators in a class (HLOR) is above 963 a software object belongs to the category Poor Maintainability. The second rule classifies that a software object is of

---

$^2$ The triple (35, 5, 0.900) means that the rule properly classified 35 data points, that it misclassified 5 data points, and its value of Laplace ratio is 0.900.
Poor Maintainability if it has the sum of each attribute’s value (ATCO) higher than 52, and its Halstead Program Volume (HLVL) is higher than 11,253.

Table 5: Inclusion table for rules V-L-1 and V-L-1A

<table>
<thead>
<tr>
<th>Inclusion pair</th>
<th>Inclusion measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>V-L-1 in V-L-2</td>
<td>0.4483</td>
</tr>
<tr>
<td>V-L-1 in V-L-1A</td>
<td>1.0000</td>
</tr>
<tr>
<td>V-L-1A in V-L-1</td>
<td>0.7632</td>
</tr>
<tr>
<td>V-L-1A in V-L-2</td>
<td>0.5000</td>
</tr>
</tbody>
</table>

7.1.2. Software Engineer A

The software engineer A identified the smallest number of software objects as of Poor Maintainability. He pointed to only 18 such objects (4.9% out of 366). The application of both scenarios resulted in four rules for the all-category scenario, and three for the two-category scenario. The Laplace ratios for these rules are much lower then for rules representing the software engineer V. The best three rules across all of them are presented below.

Rule\(_{A-L-1A} (5, 0, 0.857)\):
IF HLDF > 135
THEN category: poor_maintainability

Rule\(_{A-L-2A} (4, 0, 0.833)\):
IF WDC <= 228 & ATCO <= 36 & HLVC > 162 & ATTR > 14
THEN category: poor_maintainability

Rule\(_{A-L-1} (9, 1, 0.833)\):
IF TYPE = 1 & LOC > 695 & MWDC <= 1 & CHLD <= 0 & OVRM > 1
THEN category: poor_maintainability

The semantic similarities are presented in Table 6. As it can be observed all the rules are quite different from each other. The highest value of the measure is obtained for the pair A-L-1 and A-L-2A.

The calculations of inclusion measures for those two rules resulted in the elimination of the rule A-L-2A (its inclusion in the rule A-L-1 is 0.7500). The two remaining rules define Poor Maintainability, from the point of view of the software engineer A, in the following way: a software object is of Poor Maintainability if its Halstead Difficulty (HLDF) is above 135, or if it is a GUI object with more than 695 lines of code (LOC), median number of decisions (MWDC) less or equal one, no children (CHLD), and more than 1 overridden methods (DVRM).
Table 6: Contingency table for Poor_maintainability rules representing the software engineer A

<table>
<thead>
<tr>
<th>Pair of rules</th>
<th>Contingency table entries</th>
<th>Similarity measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>(R1 &amp; R2)</td>
<td>a (R1 &amp; R2)</td>
<td>b (R1 &amp; nR2)</td>
</tr>
<tr>
<td>(A-L-1, A-L-1A)</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>(A-L-1, A-L-2A)</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>(A-L-1A, A-L-2A)</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>

7.1.3. Software Engineer D
The software engineer D evaluated 35 (9.6 % out of 366) software objects as of Poor Maintainability. Both rule generation approaches resulted in five rules for the all-category scenario, and two for the two-category scenario. The best rules are shown:

**Rule\textsubscript{D-L-1A} (13, 0, 0.933):**
IF REMM > 101
THEN category: poor maintainability

**Rule\textsubscript{D-L-2A} (10, 0, 0.917):**
IF LOC > 401 & DINH <= 2 & SIBL > 1 & RFO > 80 & MNL3 > 13
THEN category: poor maintainability

**Rule\textsubscript{D-L-1} (12, 0, 0.929):**
IF MTOK <= 154 & HLVC >198 & MIC > 2
THEN category: poor maintainability

According to the proposed analysis, similarities measures are calculated, Table 7. Following this, inclusion measures of rules D-L-1 and D-L-1A in reference to other rules are obtained. As the result, the rule D-L-1 is removed (its inclusion in D-L-1A is 0.8333). The simplest of rules that represent the software engineer D indicates that Poor Maintainability software objects are ones with number of remote methods (REMM) exceeded 101. The second rule is more complex: the Poor Maintainability software objects have more than 401 lines of code (LOC), less or equal to two depth of inheritance (DINH), more than one siblings (SIBL), more than 80 methods executed in response to a message received by the class (RFO), and a mean method name length (MNL3) over 13 characters.

Table 7: Contingency table for Poor_Maintainability rules representing the programmer D

<table>
<thead>
<tr>
<th>Pair of rules</th>
<th>Contingency table entries</th>
<th>Similarity measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>(R1 &amp; R2)</td>
<td>a (R1 &amp; R2)</td>
<td>b (R1 &amp; nR2)</td>
</tr>
<tr>
<td>(D-L-1, D-L-1A)</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>(D-L-1, D-L-2A)</td>
<td>0</td>
<td>12</td>
</tr>
<tr>
<td>(D-L-1A, D-L-2A)</td>
<td>2</td>
<td>11</td>
</tr>
</tbody>
</table>
7.1.4. Comparison of Rules Generated for Software Engineers V, A, and D
The analysis of all generated rules has led to selection of three pairs of rules, with each pair “representing” a single engineer. The next step is comparison of those three pairs of rules. This comparison should provide us with an indication if there is any common ground used to identify Poor Maintainability objects. The similarity measures are calculated for eight triples. We can conclude, based on the results, that there is a very little “agreement” among all three software engineers in identifying Poor Maintainability software objects. The values of similarity measures are very small – in just a few cases the similarity measures are around 0.25.

7.2. Excellent Maintainability
7.2.1. Software Engineer V
The number of objects identified by the software engineer V as of Excellent Maintainability is 139 (38.0% out of 366). Such a large number of objects led to generation of rules with high coverage. The best three rules are shown below:

**RuleV-H-1A (64, 4, 0.929):**
IF METH \(\leq 3\) & WDC \(\leq 11\) & MNL1 \(\leq 21\) & WMC2 \(\leq 5\)
THEN category: excellent maintainability

**RuleV-H-1 (73, 2, 0.961):**
IF ALOC \(\leq 2\) & DAC \(\leq 1\) & HLPL \(\leq 142\)
THEN category: excellent maintainability

**RuleV-H-2 (125, 10, 0.920):**
IF MNL1 \(\leq 24\) & HLPL \(\leq 142\) & HLUR \(\leq 11\)
THEN category: excellent maintainability

The similarity measures for all three pairs indicate that the rules are alike. The rules of the last pair, Table 8, are the most similar. After calculating inclusion measures it becomes obvious that the rule V-H-1 can be removed (its inclusion measure with the rule V-H-2 is 0.920).

The rules describing what does it mean Excellent Maintainability for software engineer V indicate that these software objects have: a number of methods (METH) less than three, a number of decisions (WDC) less than or equal 11, a maximum method name length (MNL1) less than 22, and a number of methods (WMC2) is less or equal to five; or a maximum method name length (MNL1) less than 24, Halstead Program Length (HLPL) less than 142, and Halstead number of unique operators (HLVR) less than 11.

<table>
<thead>
<tr>
<th>Pair of rules (R1, R2)</th>
<th>Contingency table entries</th>
<th>Similarity measure</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a (R1 &amp; R2)</td>
<td>b (R1 &amp; nR2)</td>
</tr>
<tr>
<td>(V-H-1, V-H-1A)</td>
<td>45</td>
<td>30</td>
</tr>
<tr>
<td>(V-H-2, V-H-1A)</td>
<td>64</td>
<td>71</td>
</tr>
<tr>
<td>(V-H-1, V-H-2)</td>
<td>69</td>
<td>6</td>
</tr>
</tbody>
</table>
7.2.2. Software Engineer A
The software engineer A identified 248 objects (67.8% out of 366) as of Excellent Maintainability. The best rules that represent his decisions are shown:

Rule_{A-H-1} (34, 0, 0.972):
\[
\text{IF TYPE} = 4 \text{ } \& \text{ HLDF} \leq 28 \\
\text{THEN category: excellent maintainability}
\]

Rule_{A-H-2} (163, 5, 0.965):
\[
\text{IF CBO} \leq 5 \text{ } \& \text{ HLDF} \leq 28 \\
\text{THEN category: excellent maintainability}
\]

Rule_{A-H-3} (70, 2, 0.959):
\[
\text{IF TYPE} = 2 \text{ } \& \text{ MAXQ} \leq 3 \text{ } \& \text{ CONS} \leq 2 \text{ } \& \text{ OVRM} \leq 13 \\
\text{THEN category: excellent maintainability}
\]

The similarities measures are quite low indicating that the rules cover quite different objects, Table 9. In order to eliminate one rule we pick the first pair and after calculating the inclusion measure we remove the rule A-H-1 (the measure of inclusion in the rule A-H-2 is 0.8529).

Table 9: Contingency table for Excellent Maintainability rules representing the software engineer A

<table>
<thead>
<tr>
<th>Pair of rules ((R_1, R_2))</th>
<th>Contingency table entries</th>
<th>Similarity measure</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(a) ((R_1 &amp; R_2))</td>
<td>(b) ((R_1 &amp; nR_2))</td>
</tr>
<tr>
<td>(A-H-1, A-H-2)</td>
<td>29</td>
<td>5</td>
</tr>
<tr>
<td>(A-H-1, A-H-3)</td>
<td>0</td>
<td>34</td>
</tr>
<tr>
<td>(A-H-2, A-H-3)</td>
<td>13</td>
<td>155</td>
</tr>
</tbody>
</table>

7.2.3. Software Engineer D
The software engineer D recognized 207 objects (56.6% out of 366) as of Excellent Maintainability. The rules are below:

Rule_{D-H-1} (135, 7, 0.944):
\[
\text{IF DEC} \leq 7 \text{ } \& \text{ RCR} \leq 0 \text{ } \& \text{ HLUN} \leq 63 \\
\text{THEN category: excellent maintainability}
\]

Rule_{D-H-2} (116, 7, 0.936):
\[
\text{IF RCC1} \leq 9.7 \text{ } \& \text{ AWDC} \leq 7 \text{ } \& \text{ IMST} \leq 12 \text{ } \& \text{ REMM} \leq 35 \text{ } \& \text{ PKGM} \leq 1 \\
\text{THEN category: excellent maintainability}
\]

Rule_{D-H-3} (160, 20, 0.885):
\[
\text{IF DAC} \leq 2 \text{ } \& \text{ HLUN} \leq 63 \\
\text{THEN category: excellent maintainability}
\]
The pair D-H-1 and D-H-2 has the highest value of similarity measure (Table 10), and the rule D-H-1 has a high value of inclusion measure (with the rule D-H-3 – 0.9437).

Table 10: Contingency table for Excellent_Maintainability rules representing the programmer A

<table>
<thead>
<tr>
<th>Pair of rules (R1, R2)</th>
<th>Contingency table entries</th>
<th>Similarity measure</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a (R1 &amp; R2)</td>
<td>b (R1 &amp; nR2)</td>
</tr>
<tr>
<td>(D-H-1, D-H-2)</td>
<td>92</td>
<td>50</td>
</tr>
<tr>
<td>(D-H-1, D-H-3)</td>
<td>134</td>
<td>8</td>
</tr>
<tr>
<td>(D-H-2, D-H-3)</td>
<td>101</td>
<td>22</td>
</tr>
</tbody>
</table>

7.2.4. Comparison of Rules Generated for Software Engineers V, A, and D

The comparison of all rules generated for all programmers shows some similarities. The similarity value calculated for the three rules V-H-2, A-H-2, D-H-3 (0.6462) indicates that these rules are quite similar. This leads to the statement that three software engineers identified reasonable number of the same software objects as of Excellent Maintainability. These objects, that are practically the same, are represented by rules that look quite differently. This means that each engineer “saw” different aspects of software objects. A better description of Excellent Maintainability objects can be obtained when those three rules can be combined.

IF MNL1 <=24 & HLPL <=142 & HLUR <=11 & CBO <= 5 & HLDF <= 28 & DAC <= 2 & HLUN <= 63 THEN category: excellent maintainability

Another relatively high value of similarity measure (0.5000) is obtained for a pairs of rules: A-H-2, D-H-2

IF CBO <= 5 & HLDF <= 28 & RCC1 <=9.7 & AWDC <=7 & IMST <=12 & REMM <= 35 & PKGM <= 1 THEN category: excellent maintainability

8 Maintainability Data – Multiple Rule Generators

The second experiment illustrates the ability of the SSACR to utilize multiple data sets and rule-generating tools. The SSARC is applied to process the maintenance data associated with the software engineer V. Two different rule-generating tools are used to extract IF-THEN rules from the data. One of them is C5.0/See5 already known from Sections 5 and 7. The other one is 4cRuleBuilder. The maintainability data were pre-processed and three output categories were identified: POOR, FAIR, and EXCELLENT (Section 4). The approach described here is a little bit different than the one presented in Section 7. The whole data set – 366 data points – has been split into two subsets: training and testing. The sizes of these sets are 245 and 121 data points respectively. The training subset is used to generate rules, and the testing subset is utilized during semantic analysis of rules for calculating similarity and inclusion measures. This time, the values of thresholds (Section 5.3) are fixed. The threshold for “goodness” is set up to 0.7 for C5.0/See5, and 0.6 for 4cRuleBuilder. In the case of similarity and inclusion, the value of 1.0 indicates an automatic removal of a rule.
8.1. Rules Generated by C5.0/See5
The C5.0/See5 generates 11 rules. Among those 11 rules, five rules are for the category Poor Maintainability, three rules for the category Fair Maintainability, and remaining three rules are for the category Excellent Maintainability.

Excellent Maintainability. The rules generated for category Excellent Maintainability are as follows:

Rule\(_{C5-M-1}^\): (17, 0, 0.947)\(^3\)
   IF WDC <= 16 & FACE <= 0 & MNL1 <= 21 & OPER <= 1
   THEN category: excellent maintainability

Rule\(_{C5-M-2}^\): (42, 1, 0.956)
   IF MNL1 <= 25 & DAC <= 3 & HLOR <= 55 & WMC2 <= 27
   THEN category: excellent maintainability

Rule\(_{C5-M-3}^\): (14, 1, 0.882)
   IF WDC <= 16 & FACE > 0 & MNL1 <= 21 & PROM <= 43
   THEN category: excellent maintainability

Those rules are evaluated using test dataset. It is important to verify how they perform on the previously “unseen” data. The values of Laplace ratio calculated for test data for all the three rules range from .882 to .956. All these three rules have the Laplace ratio values above the threshold value, therefore, all of them are considered further for finding the best rules for prediction of Excellent Maintainability. The values of similarity and inclusion measures are presented in Table 11.

The above comparison of rules for Excellent Maintainability clearly shows that the rule C5-M-2 dominates the other two rules. The rule C5-M-1 is completely included in the rule C5-M-2 (Figure 6), and hence can be discarded without any loss of coverage. The rule C5-M-3 is more than 86% included in the rule C5-M-2, and if the analysis is focused on finding a small number of rules with possible loss of coverage, the rule can be also discarded.

Table 11: Contingency table for Excellent_Maintainability rules representing the software engineer V and generated by C5.0/See5

<table>
<thead>
<tr>
<th>Pair of rules (R1, R2)</th>
<th>Contingency table entries</th>
<th>Similarity measure</th>
<th>Inclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a (R1&amp;R2)</td>
<td>b (R1&amp;nR2)</td>
<td>c (nR1&amp;R2)</td>
</tr>
<tr>
<td>(C5-M-1, C5-M-2)</td>
<td>17</td>
<td>0</td>
<td>26</td>
</tr>
<tr>
<td>(C5-M-1, C5-M-3)</td>
<td>0</td>
<td>17</td>
<td>15</td>
</tr>
<tr>
<td>(C5-M-2, C5-M-3)</td>
<td>101</td>
<td>22</td>
<td>158</td>
</tr>
</tbody>
</table>

\(^3\) The triple (17, 0, 0.947) means that the rule properly classified 17 data points, that it misclassified 0 data point, and its value of Laplace ratio is 0.947. The values are for the testing dataset.
The fact that the rule C5-M-2 appears to be the most important and relevant is supported by the high value of its Laplace ratio - 0.956 which it the highest among all three rules. Elimination of the rule C5-M-3 depends on the type of application and requirements of domain experts. There is a slight loss of coverage if the rule C5-M-3 is ignored, thus domain experts have to critically analyze a possible loss before making any decision regarding elimination of this rule.

Another interesting observation is made when we look at those rules from the point of view of their syntax. These rules are very different syntactically. For example, the first two rules hardly have the same attributes. The only attribute in common is MNL1 (Maximum Method Name Length), but if we look at the results of comparison of those rules, Figure 4, we find that the rule C5-M-1 is 100 percent included in the rule C5-M-2. All the data points that are covered by rule C5-M-1 are also covered by rule C5-M-2. This proves the redundancy of these rules - they cover the same points. Thus, an important observation is that two rules might be very different syntactically but based on semantic measures they both cover the same data points. In other words, two syntactically different rules provide us with different descriptions of the same data points – this can be useful during a process of knowledge extraction.

**Fair Maintainability.** For the category *Fair Maintainability*, there are three rules generated by C5.0/See5 based on the training dataset. Firing these rules on test dataset and ignoring the ones with Laplace ratio values less than 0.70 (threshold), leads to two rules. They are as follows:

**Rule C5-M-6:** (12, 4, 0.722)

IF MNL1 > 25 & DAC <= 3

THEN category: *fair maintainability*

**Rule C5-M-7:** (52, 19, 0.726)

IF HLOR > 55

THEN category: *fair maintainability*

The rules are compared, Table 12 and Figure 5. Rules C5-M-6 and C5-M-7 have low similarity and inclusion values. Based on the above comparison of these two rules none of them can be ignored. Both rules are used to predict the category *Fair Maintainability* on maintainability data.
Table 12. Contingency table for *Fair_Maintainability* rules representing the software engineer V and generated by C5.0/See5

<table>
<thead>
<tr>
<th>Pair of rules (R1, R2)</th>
<th>Contingency table entries</th>
<th>Similarity measure</th>
<th>Inclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a (R1&amp;R2)</td>
<td>b (R1&amp;nR2)</td>
<td>c (nR1&amp;R2)</td>
</tr>
<tr>
<td>(C5-M-6, C5-M-7)</td>
<td>12</td>
<td>4</td>
<td>59</td>
</tr>
</tbody>
</table>

Rule **C5-M-4**: (12, 1, 0.857)

IF MNL2 <= 8 & HLOR > 933

THEN category: poor maintainability

Rule **C5-M-5**: (7, 1, 0.778)

IF INCL <= 1 & HLOR > 933

THEN category: poor maintainability

The values of similarity and inclusion obtained for both rules are presented in Table 13. Figure 6, illustrates that the rules are quite different. None of them is fully included in the other. The inclusion measures, Table 13, are 0.5000 and 0.8570 – although the measures are relatively high we do not discard any rule – at the beginning of Section 8 we stated that removal of a rule happens only when inclusion equals to 1.0. Thus, both the rules should be used to predict *Poor Maintainability*.

### Poor Maintainability

For the category *Poor Maintainability* five rules are generated. Only two rules out of five have qualified for further analysis with the Laplace ratio values above the established threshold.

Rule **C5-M-4**:

IF MNL2 <= 8 & HLOR > 933

THEN category: poor maintainability

Rule **C5-M-5**:

IF INCL <= 1 & HLOR > 933

THEN category: poor maintainability

The values of similarity and inclusion obtained for both rules are presented in Table 13. Figure 6, illustrates that the rules are quite different. None of them is fully included in the other. The inclusion measures, Table 13, are 0.5000 and 0.8570 – although the measures are relatively high we do not discard any rule – at the beginning of Section 8 we stated that removal of a rule happens only when inclusion equals to 1.0. Thus, both the rules should be used to predict *Poor Maintainability*.

Table 13: Contingency table for *Poor_Maintainability* rules representing the software engineer V and generated by C5.0/See5

<table>
<thead>
<tr>
<th>Pair of rules (R1, R2)</th>
<th>Contingency table entries</th>
<th>Similarity measure</th>
<th>Inclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a (R1&amp;R2)</td>
<td>b (R1&amp;nR2)</td>
<td>c (nR1&amp;R2)</td>
</tr>
<tr>
<td>(C5-M-4, C5-M-5)</td>
<td>6</td>
<td>6</td>
<td>1</td>
</tr>
</tbody>
</table>
8.2. Rules Generated by 4cRuleBuilder

4cRuleBuilder has generated a total of nine rules. One rule is for the category Poor Maintainability, five rules for the category Fair Maintainability, and three rules are for the category Excellent Maintainability.

Excellent Maintainability. Given below are the three rules for the category Excellent Maintainability generated on training data by 4cRuleBuilder. We include these rules to illustrate how different they are syntactically. Semantic similarity of those rules with the ones generated by C5.0/See5 is analyzed in Section 8.3

Rule 4C-M-1 (24, 0, 0.962):
IF
   ATOK != [64.50,81.00),[50.50,64.50),[96.50,118.00),[118.00,144.00),
   [201.00,322.00),[81.00,96.50), [144.00,169.00),
   MTOK != [52.50,68.00),[42.50,52.50),[68.00,103.00),
   DEC != [9.50,20.50),[20.50,32.50), [32.50,48.50),[48.50,62.00),[81.00,124.00),
   [62.00,81.00),[124.00,925.00),
   FACE != [3.50,4.50),[0.50,2.50),[2.50,3.50),
   RCR != [1.20,2.75),
   RFC != [385.00,401.00),[37.50,52.50),[483.00,1.16E3),[441.00,483.00), [70.50,111.00),
   MNL3 != [21.50,38.00),
   MNL4 != [23.50,41.50),
   ATCO != [83.00,128.00),[49.00,64.50),[128.00,887.00),
   HLPL != [310.00,350.00),[350.00,412.00),[1.96E3,2.43E3),[541.00,680.00), [810.00,1.06E3),
   [680.00,810.00),[1.06E3,1.43E3),[1.43E3,1.96E3),[2.43E3,3.43E3), [3.43E3,1.99E4),
   [456.00,541.00),[412.00,456.00),
   CHCL != [1.50,5.50),<CONS !=[3.50,5.50),[5.50,9.00),
   OPER != [21.50,34.50),[34.50,58.00),[58.00,212.00),
THEN category: excellent maintainability
Rule_4C-M-2 (20, 0, 0.955):

IF TYPE != 3,4
LOC != [98.50,122.00),[869.00,1.22E3),[203.00,250.00),[364.00,466.00), [250.00,303.00), [579.00,713.00),[1.22E3,8.16E3),[303.00,364.00),[170.00,203.00),[466.00,579.00),
[11.50,25.00),[148.00,170.00),[713.00,869.00)
ATCO != [36.50,49.00),[7.50,17.50),[17.50,26.50),[83.00,128.00),
[64.50,83.00), [49.00,64.50), [128.00,887.00)
HLPL != [310.00,350.00),[130.00,172.00),[1.96E3,2.43E3),[810.00,1.06E3),[1.06E3,1.43E3),
[1.43E3,1.96E3),[2.43E3,3.43E3),[3.43E3,1.99E4),
[456.00,541.00),[172.00,235.00),[412.00,456.00),[58.50,87.50), [235.00,310.00), [87.50,130.00)>
PUBM != [12.50,25.50),[25.50,35.50),[66.50,76.50),[-0.50,12.50)

THEN category: excellent maintainability

Rule_4C-M-3 (12, 2, 0.813):

IF LOC != [-1.00,11.50),[122.00,148.00),[65.00,80.50),[98.50,122.00),
[869.00,1.22E3),[203.00,250.00),[364.00,466.00), [250.00,303.00),[579.00,713.00),[1.22E3,8.16E3),
[303.00,364.00),[1.43E3,1.96E3),[2.43E3,3.43E3),[3.43E3,1.99E4),
[456.00,541.00),[172.00,235.00),[412.00,456.00),[58.50,87.50), [235.00,310.00), [87.50,130.00)>
MNL3 != [8.50,15.50),[15.50,21.50)>

THEN category: excellent maintainability

All the rules above for the category Excellent Maintainability have high values of Laplace ratio (threshold Laplace ratio for 4cRuleBuilder is .60) and hence they are compared with each other, Table 14 and Figure 7.

Table 14: Contingency table for Excellent_Vaintainability rules (the software engineer V)

<table>
<thead>
<tr>
<th>Pair of rules (R1, R2)</th>
<th>Contingency table entries</th>
<th>Similarity measure</th>
<th>Inclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>a (R1&amp;R2)</td>
<td>b (R1&amp;nR2)</td>
<td>c (nR1&amp;R2)</td>
<td>d (nR1&amp;nR2)</td>
</tr>
<tr>
<td>(4C-M-1, 4C-M-2)</td>
<td>11</td>
<td>13</td>
<td>9</td>
</tr>
<tr>
<td>(4C-M-1, 4C-M-3)</td>
<td>12</td>
<td>12</td>
<td>2</td>
</tr>
<tr>
<td>(4C-M-2, 4C-M-3)</td>
<td>8</td>
<td>12</td>
<td>6</td>
</tr>
</tbody>
</table>

Figure 7: Rule Inclusion Diagram for rules generated by 4cRuleBuilder for the category Excellent_Maintainability
It can be easily seen that there is no dominating rule. However, the levels of inclusion will give domain experts some insight into interpretation of the rules. There is some overlapping between the rules, but overall, they cover different points. Thus, rules 4C-M-1, 4C-M-2 and 4C-M-3 are selected as best rules generated by 4cRuleBuilder to predict the category *Excellent Maintainability*.

**Fair Maintainability**. Now, we look at initial rules generated for the category *Fair Maintainability*. Five rules are generated by 4cRuleBuilder on the training dataset. These rules were fired on test dataset. Three rules are left after ignoring rules with low Laplace ratio values.

**Rule4CM-4 (40, 21, 0.650):**

IF TYPE != 4  
LOC != [869.00,1.22E3],[25.00,43.50],[1.22E3,8.16E3],[43.50,65.00],[122.00,148.00]  
TOK != [3.57E3,4.25E3],[52.00,86.00],[6.23E3,3.65E4],[121.00,173.00]  
DEC != [62.00,81.00],[124.00,925.00]  
INCL != [2.50,5.50],[9.50,16.00]  
LCOM != [119.00,170.00]  
HLEF != [82.50,323.00],[6.08E5,9.72E5],[0.50,82.50],[2.04E6,1.23E7],[323.00,694.00],[3.94E3,8.13E3],[1.73E3,2.34E3],[1.25E4,2.05E4]  
HLOR != [-0.50,12.50],[1.72E3,1.01E4],[26.50,39.50]  
MEMB != [40.50,67.00]>PKGM !=[14.50,35.50]  
THEN category: *fair maintainability*

**Rule4CM-5 (10, 5, 0.647):**

IF TYPE != 2,3  
LOC != [250.00,303.00],[579.00,713.00],[25.00,43.50],[1.22E3,8.16E3],[43.50,65.00],[11.50,25.00],[303.00,364.00],[203.00,250.00],[98.50,122.00]  
TOK != [1.25E3,1.41E3],[4.25E3,6.23E3],[52.00,86.00],[-2.00,24.50],[24.50,52.00],[781.00,1.02E3],[1.02E3,1.25E3],[583.00,663.00]  
ATOK != [96.50,118.00],[144.00,169.00],[322.00,771.00],[22.50,37.50]  
MTOK != [179.00,736.00],[68.00,103.00]  
FACE != [3.50,4.50],[2.50,3.50],[0.50,2.50]  
LCOM != [601.00,1.37E3],[1.37E3,1.43E4],[49.00,84.50]  
RFC != [12.50,22.50],[336.00,358.00],[0.50,12.50],[111.00,297.00]  
HLEF != [82.50,323.00],[1.73E5,2.33E5],[694.00,1.73E3],[-0.50,82.50],[2.04E6,1.23E7],[2.34E3,3.94E3],[8.23E4,1.23E5],[2.05E4,2.96E4]  
THEN category: *fair maintainability*
Rule \(4C-M-6(3,1,0.667)\):

IF LOC \(!=\) [(-1.00,11.50), [869.00,1.22E3), [1.22E3,8.16E3), [43.50,65.00), [65.00,80.50), [43.50,466.00), [466.00,579.00), [713.00,869.00), [203.00,250.00)]

RCC1 \(!=\) [7.05,8.80), [8.80,10.30), [11.90,14.50), [10.30,11.90), [14.50,16.60), [4.65,5.75), [57.00,434.00)]

RCC2 \(!=\) [17.50,26.50), [90.00,112.00), [73.50,90.00), [112.00,136.00), [26.50,35.50), [61.00,73.50)]

TOK \(!=\) [3.57E3,4.25E3), [3.05E3,3.57E3), [4.25E3,6.23E3), [6.23E3,3.65E4), [24.50,52.00), [1.41E3,1.84E3), [781.00,1.02E3), [263.00,374.00), [1.02E3,1.25E3), [1.84E3,3.05E3), [374.00,523.00)]

DINH \(!=\) [3.50,4.50), [0.50,2.50)

RFC \(!=\) [323.00,336.00), [401.00,441.00), [483.00,1.16E3), [52.50,70.50), [0.50,12.50), [111.00,297.00), [385.00,401.00), [297.00,323.00), [358.00,385.00)

THEN category: \textit{fair maintainability}

They are compared for similarity and inclusion measures and the result is shown in Table 15 and Figure 8.

Table 15: Contingency table for \textit{Excellent_Maintainability} rules representing the software engineer V

<table>
<thead>
<tr>
<th>Pair of rules (R1, R2)</th>
<th>Contingency table entries</th>
<th>Similarity measure</th>
<th>Inclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a (R1&amp;R2)</td>
<td>b (R1&amp;nR2)</td>
<td>c (nR1&amp;R2)</td>
</tr>
<tr>
<td>(4C-M-4, 4C-M-5)</td>
<td>5</td>
<td>56</td>
<td>10</td>
</tr>
<tr>
<td>(4C-M-4, 4C-M-6)</td>
<td>1</td>
<td>60</td>
<td>3</td>
</tr>
<tr>
<td>(4C-M-5, 4C-M-6)</td>
<td>0</td>
<td>15</td>
<td>4</td>
</tr>
</tbody>
</table>

Figure 8: Rule Inclusion Diagram for rule generated by 4cRuleBuilder for the category \textit{Fair_Maintainability}

The comparison clearly shows that none of the rules can be discarded. The rules are very different semantically and show very low of redundancy.

\textbf{Poor Maintainability}. There is only one rule generated by 4cRuleBuilder for the category \textit{Poor Maintainability}. The rule is shown below. Additionally, its value of Laplace ratio is below threshold. The rule is not considered for further investigations.
Rule4C-M-7 (6, 25, 0.212):

\[
\text{IF } \begin{align*}
\text{RCC1} & \neq [7.05,8.80),[10.30,11.90),[22.90,34.40),[34.40,57.00),[57.00,434.00),[-0.85,3.45),[11.90,14.50),[3.45,4.65),[8.80,10.30) \\
\text{LCOM} & \neq [-0.50,10.50),[10.50,28.50),[28.50,49.00),[84.50,119.00),[325.00,601.00) \\
\text{HLVL} & \neq [81.00,161.00),[2E3,2.25E3),[549.00,895.00),[2.25E3,2.7E3),[3.74E3,4.9E3),[6.02E3,7.61E3),[1.57E3,2E3),[161.00,270.00),[1.00,37.00),[3.01E3,3.74E3),[1.07E3,1.57E3),[2.7E3,3.01E3),[895.00,1.07E3),[37.00,81.00),[387.00,549.00),[270.00,387.00),[1.94E4,2.99E4),[1.58E4,1.94E4) \\
\text{THEN category: poor maintainability}
\end{align*}
\]

Table 16 is the summary of all rules, generated by both tools – C5.0/See5 and 4cRuleBuilder, that have been found to be unique.

<table>
<thead>
<tr>
<th>Category: Excellent Maintainability</th>
<th>Category: Fair Maintainability</th>
<th>Category: Poor Maintainability</th>
</tr>
</thead>
<tbody>
<tr>
<td>C5.0/See5</td>
<td>C5-M-2</td>
<td>C5-M-4</td>
</tr>
<tr>
<td>C5-M-3</td>
<td>C5-M-6</td>
<td>C5-M-5</td>
</tr>
<tr>
<td>4cRuleBuilder</td>
<td>4C-M-1</td>
<td></td>
</tr>
<tr>
<td>4C-M-2</td>
<td>4C-M-4</td>
<td></td>
</tr>
<tr>
<td>4C-M-3</td>
<td>4C-M-5</td>
<td></td>
</tr>
<tr>
<td>4C-M-6</td>
<td>4C-M-6</td>
<td></td>
</tr>
</tbody>
</table>

8.3. Combined Analysis of Rules generated by C5.0/See5 and 4cRuleBuilder

The SSACR is able to perform one more important task – comparison of rules generated by different tools. In the presented work, there are two different tools C5.0/See5 and 4cRuleBuilder. Till now, we have been comparing rules generated by the same tool for each category. Once the relevant and important rules generated by each tool are found, they can be compared with relevant rules generated by the other tools. The comparison is performed only for categories Excellent Maintainability and Fair Maintainability. The category Poor Maintainability has only two rules generated by C5.0/See5, and the rules have been already analyzed.

Let us take a look at the rules generated for the category Excellent Maintainability by C5.0/See5 and 4C Rule Builder. We have C5-M-2 and C5-M-3 generated by C5.0/See5, and 4C-M-1, 4C-M-2 and 4C-M-3 generated by 4cRuleBuilder, Table 16. All of these rules have high Laplace ratio values and are quite unique (all highly inclusive rules are ignored). The results of comparison (Comparisons is done on test dataset (121 data point)) are presented in Table 17.
Table 17: Contingency table for Excellent_Maintainability rules (the software engineer V)

<table>
<thead>
<tr>
<th>Pair of rules (R1, R2)</th>
<th>Contingency table entries</th>
<th>Similarity measure</th>
<th>Inclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a (R1&amp;R2)</td>
<td>b (R1&amp;nR2)</td>
<td>c (nR1&amp;R2)</td>
</tr>
<tr>
<td>(C5-M-2, 4C-M-1)</td>
<td>23</td>
<td>20</td>
<td>1</td>
</tr>
<tr>
<td>(C5-M-2, 4C-M-2)</td>
<td>19</td>
<td>24</td>
<td>1</td>
</tr>
<tr>
<td>(C5-M-2, 4C-M-3)</td>
<td>12</td>
<td>31</td>
<td>2</td>
</tr>
<tr>
<td>(C5-M-3, 4C-M-1)</td>
<td>0</td>
<td>15</td>
<td>24</td>
</tr>
<tr>
<td>(C5-M-3, 4C-M-2)</td>
<td>6</td>
<td>9</td>
<td>14</td>
</tr>
<tr>
<td>(C5-M-3, 4C-M-3)</td>
<td>0</td>
<td>15</td>
<td>14</td>
</tr>
</tbody>
</table>

Figure 9 shows the comparison chart of inclusions of C5.0/See5 and 4cRuleBuilder rules.

Figure 9 does not show inclusions among all rules to ease readability. Clearly, we see that the rule C5-M-2, generated by C5, dominates as it covers a good percentage of data points covered by other rules.

There are two possible ways of looking at the results. We already know that the rule C5-M-2 dominates all the rules generated by both tools and thus, can be used alone to predict Excellent_Maintainability. Alternatively, domain experts can use the set of other rules, i.e., C5-M-3, 4C-M-1, 4C-M-2, and 4C-M-3. Comparisons of these rules clearly illustrates that this set of rules is very distinctive and the rules are dissimilar to each other. Thus, this set of four rules can be used instead of using the rule C5-M-2.

These four rules are very different syntactically when compared with the rule C5-M-2 what means that we have a chance to obtain a very different description (in the form of different software attributes) of software objects that belong to the category Excellent_Maintainability.

Now, we show the comparison of rules for the category Fair Maintainability. From Table 16, we have two rules generated by C5.0/See5: C5-M-6 and C5-M-7, and 4C-M-4, 4C-M-5, 4C-M-6 – the rules generated by 4cRuleBuilder. The values of similarity and inclusion between these rules are shown in Table 18.
Table 18: Contingency table for Fair Maintainability rules representing the software engineer V

<table>
<thead>
<tr>
<th>Pair of rules (R1, R2)</th>
<th>Contingency table entries</th>
<th>Similarity measure</th>
<th>Inclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a (R1&amp;R2)</td>
<td>b (R1&amp;nR2)</td>
<td>c (nR1&amp;R2)</td>
</tr>
<tr>
<td>(C5-M-6, 4C-M-4)</td>
<td>9</td>
<td>7</td>
<td>52</td>
</tr>
<tr>
<td>(C5-M-6, 4C-M-5)</td>
<td>2</td>
<td>14</td>
<td>13</td>
</tr>
<tr>
<td>(C5-M-6, 4C-M-6)</td>
<td>1</td>
<td>15</td>
<td>3</td>
</tr>
<tr>
<td>(C5-M-7, 4C-M-4)</td>
<td>38</td>
<td>33</td>
<td>23</td>
</tr>
<tr>
<td>(C5-M-7, 4C-M-5)</td>
<td>9</td>
<td>62</td>
<td>6</td>
</tr>
<tr>
<td>(C5-M-7, 4C-M-6)</td>
<td>2</td>
<td>69</td>
<td>2</td>
</tr>
</tbody>
</table>

The analysis clearly indicates that all these rules are quite different. Only a single case of high similarity and inclusion is observed. The rules C5-M-7 and 4C-M-4 have 70.4% similarity. They also display high levels of inclusion – 95% of data points covered by the rule 4C-M-4 are also covered by the rule C5-M-7. Therefore, 4C-M-4 can be discarded. This leaves behind four best rules – C5-M-6, C5-M-7, 4C-M-5, and 4C-M-6.

9 Conclusions

The paper presents a methodology for semantic-based comparison of rules using measures of similarity and inclusion. Application of these measures gives a chance to look ‘inside’ rules and creates possibility of making a selection of good and unique rules representing analyzed data. Such sets of rules represented the most suitable rules for constructing rule-based decision support systems.

A system for analysis of rules based on the proposed methodology has been designed and implemented – it is called the System for Semantic-based Analysis and Comparison of Rules (SSACR). The SSACR offers a mechanism for finding generalized rules based on their semantic comparison. This mechanism provides a better understating of the rules’ performance and thus enables the domain experts to choose rules that interest them the most. The system presented considers the Laplace ratio as a distinguishable measure of goodness of rules, and identifies a set of rules that can be considered for removal.

The proposed methodology is applied to software maintenance data that represent evaluation of maintainability levels of software objects that was performed independently by three software engineers. The paper presents rules that have been identified as the best and most unique ones. Sets of rules representing software attributes that each software engineer implicitly considered as the most important for a given level of maintainability are shown. The results of the comparison of these rules are also included. The comparison of rules constructed by different rule-generating tools is discussed.

10 Appendix

One of important ways of evaluating a rule is to numerically represent its classification capability. In other words, this means calculating "goodness" of rules. The "goodness" can be calculated in a number of ways using variety of measures. Some of these measures are directly obtained from the classification process: a number of correctly classified data points, a number of misclassified points; while some could be obtained via simple arithmetic calculations, for example: accuracy, recall, and precision, as well as confidence and support. There are a number of ways how correctness of the rule can be evaluated.
The values of accuracy, recall and precisions are calculated based on the confusion matrix. In the confusion matrix for the case of two-class classification process, let $a$ represents TruePositive, $b$ – FalsePositive, $c$ – FalseNegative, and $d$ – TrueNegative. In such case the following formulas can be used:

$$accuracy = \frac{a + b}{a + b + c + d} \quad (10.8)$$

$$recall = \frac{a}{a + c} \quad (10.9)$$

$$precision = \frac{a}{a + b} \quad (10.10)$$

The support of the rule is the relative frequency of rules that contain $A$ and $C$:

$$support(A \rightarrow C) = support(A \cup C) \quad (10.11)$$

$$support(A \rightarrow C) = \frac{support(A \cup C)}{support(A)} \quad (10.12)$$

The method used in the paper is based on application of the Laplace ratio. This ratio is very easily applicable to evaluate “goodness” of a single rule. In such case, the direct measures: number of properly classified data points and number of misclassified data points, are used. The formula used for calculating value of the ratio is presented below:

$$LaplaceRatio = \frac{Properly\_Classified\_Points + 1}{Properly\_Classified\_Points + Misclassified\_Points + 2} \quad (10.13)$$

This formula provides a balance between rule's ability to properly classify data points, and its popularity. The popularity relates to a number of data points that satisfy the rule. It is assumed that such behavior of the Laplace ratio is very much related to human's perception of "goodness" of the rule that is a combination of classification ability of the rule and its generality.

References

http://dx.doi.org/10.1109/rams.2002.981648

http://dx.doi.org/10.1145/170035.170072

http://dx.doi.org/10.1109/icsm.1994.336779

http://dx.doi.org/10.2307/1418869
http://dx.doi.org/10.1016/j.knosys.2013.12.008

http://dx.doi.org/10.1109/TKDE.2007.190610

http://dx.doi.org/10.1016/j.neucom.2010.02.016


http://dx.doi.org/10.1109/2.303623

http://dx.doi.org/10.1016/0164-1212(94)00125-7


http://dx.doi.org/10.1109/METRIC.1999.809739

http://dx.doi.org/10.1109/csmr.2001.914975


http://dx.doi.org/10.1007/978-3-540-48625-1_7

http://dx.doi.org/10.1145/74311.74325

http://dx.doi.org/10.1109/csmr.2004.1281427

http://dx.doi.org/10.1016/j.ijar.2009.03.002

http://dx.doi.org/10.1109/32.403791

http://dx.doi.org/10.1016/0164-1212(94)00130-F

http://dx.doi.org/10.1007/978-3-540-75390-2_2


http://dx.doi.org/10.1109/72.846746

http://dx.doi.org/10.1109/icsm.1992.242525


http://dx.doi.org/10.1016/j.fss.2003.10.007


    http://dx.doi.org/10.1007/978-1-4612-1500-4

    http://dx.doi.org/10.1037/0033-295X.84.4.327


[34] Rule Quest Research Data Mining Tools, [http://www.rulequest.com].

[35] 4cData - Data Mining and Knowledge Discovery Tools and Services, [http://www.aboutus.com/4cdata.net].