Diagnosis of Software Erosion through Fuzzy Logic

Ricardo Pérez-Castillo, Ignacio García Rodríguez de Guzmán and Mario Piattini
Alarcos Research Group, University of Castilla-La Mancha
Paseo de la Universidad, 4 13071, Ciudad Real, Spain
{ricardo.pdelcastillo, ignacio.grodriguez, mario.piattini}@uclm.es

Abstract— Companies have a vast number of existing software systems, which are not immune to software erosion and ageing as a consequence of uncontrolled maintenance over time. Currently, there are several metrics to measure and quantify software erosion, which also recommends some maintenance actions to deal with software erosion. Unfortunately, there are many symptoms at the same time and several possible maintenance actions that could be carried out. As a consequence, this uncertain environment implies that the best set of actions is unknown and cannot be certainly linked to specific detected erosion symptoms. This paper provides a fuzzy rule-based system to address that challenge. The system is divided into two levels: the first one recognizes precise software erosion metrics and provides fuzzy software erosion symptoms; and the second one takes the fuzzy symptoms and finally obtains fuzzy maintenance actions. This system is therefore a decision-making mechanism to select the best set of actions depending on the specific software erosion symptoms. This system has been implemented using the Matlab Fuzzy Logic Toolbox and it was simulated using Simulink.

Keywords. Fuzzy Rule-Based System; Software Erosion; Maintenance; Decision-Making.

I. INTRODUCTION

According to the Lehman’s first law, a software system must continually evolve or it will become progressively less suitable in real-world environments [8]. Indeed, companies count on a vast number of existing software systems which are not immune to software erosion and software ageing, i.e., existing software systems that become progressively less maintainable [14].

The successive changes in a software system degrade its quality, and thus, a new and improved system should replace the previous one. However, the wholesale replacement of these systems from scratch is risky since it has a great impact in technological, human and economic terms [7, 16]. The technological and human point of view is affected since replacement would involve retraining all the users in order to understand the new system and the new technology, or the new system may lack specific functionalities that are missing due to the technological changes. Moreover, the economic point of view is also affected since the replacement of an entire legacy system implies a low Return of Investment (ROI) in that system. In addition, the development or purchase of a new system could exceed a company’s budget.

In order to address the phenomenon of software erosion, the evolutionary maintenance is a better solution to obtain improved systems, without discarding the existing systems, thus minimizing the software erosion effects. Evolutionary maintenance makes it possible to manage controllable costs and preserves the valuable business knowledge embedded in the legacy system, since 78% of maintenance changes are corrective or behavior-preserving [3].

When companies are faced with the phenomenon of software erosion, they have two main challenges. Firstly, they should know how their systems’ erosion levels are, i.e., the erosion symptoms. Secondly, companies should know the best set of maintenance actions to carry out in order to solve, or at least mitigate, those detected symptoms. In addition, the selected actions should be carried out carefully without committing more erosion problems. There are some works in the literature addressing software erosion symptoms detection. A common and widely-used diagnosis framework was proposed by Vissagio [19]. That framework recognizes a set of symptoms for the diagnosis of software erosion, as well as a set of formal metrics to measure those symptoms. In addition, this framework provides some maintenance actions to address each symptom. TABLE I summarizes symptoms, metrics and maintenance actions proposed by Vissagio [19].

The metrics proposed in that framework can be accurately scored by observing software systems and counting the specific elements (e.g., the number of clone programs, number of dead lines of source code, and so forth). However, the recommended maintenance actions usually are carried out in an uncertainly environment, since there are similar actions that are the same for different erosion symptoms (e.g. refactor, reverse engineering, etc). In addition, some actions while solve some symptoms could make another symptoms worse. Establishing certainly relationships between symptoms and actions is therefore a hard task.

This paper proposes a fuzzy rule-based system to address the uncertainty in the decision-making process related to select the most appropriate set of maintenance actions to eradicate software erosion symptoms. The objective of this paper is to propose a fuzzy diagnosis system to detect software erosion symptoms, which makes it possible to select the most appropriate set of maintenance actions. That is, a set of actions that reduce the maintenance effort (and therefore the maintenance cost), and minimize the erosion symptoms in a higher level.
The design of the proposed fuzzy rule-based system follows the Mandami fuzzy controller configuration [9, 10], which is also known as Fuzzy Inference System (FIS). The proposed FIS is justified by the fact that it can deal with the uncertainty [20], which mainly appears in three key parts of the FIS.

- The software erosion symptoms (e.g., pollution, missing knowledge, coupling and anomalous data) are a combination of some metrics and they cannot be accurately established for a particular software system. Each software erosion symptom can be defined by a fuzzy set establishing the level (between 0 and 1) which the symptoms appear in a software system.

- Metrics represent input, certain variables that decide each erosion symptoms. These precise values are taken from the measurement of certain aspects of the software (e.g., number of clone programs, number of dead data, etc). Nevertheless, each precise variable can become a fuzzy set in order to achieve fuzzy values of each erosion symptoms.

- Output variables of the proposed FIS system are a set of maintenance actions. These actions are carried out in a fuzzy way. This is due to the fact that there is an uncertainty derived by the search of an agreement between cost and benefit (in terms of software erosion reduction) of the maintenance actions. As a consequence, the fuzzy definition of the output variables can establish fuzzy rules between inputs and outputs in the proposed FIS system.

The remaining of this paper is organized as follows. Section II briefly show related work with this paper. Section III presents in detail the proposed FIS system. Section IV provides an implementation of the systems and Section V simulates the FIS system with a real-life software system. Finally, Section VI discusses conclusions and future work.

II. RELATED WORK

Software maintenance is a time-consuming and hard task, which requires most effort than software development throughout the software lifecycle [6]. The detection of software erosion in the maintenance activity is a key task to know if new maintenance actions are (or are not) necessary. For this reason, maintenance levels measurement has been widely studied in literature for many years.

Hall et al. [4] provided a set of relations between some metrics and specific demands in different maintenance areas. Basili et al. [2] presented a study to deal with the prediction of maintenance process, although that work does not focus on the software erosion symptoms. Lehman et al. [8] also take into account the evolution of some metrics related to the maintenance activity. Hayes et al. [5] provide a recent model to estimate the human maintenance effort related to some maintenance metrics. However, that work does not consider the software erosion metrics and its relation with specific maintenance actions. Vissagio [19] provides a framework focusing on the relationship between software erosion metrics and the needed maintenance.

All this work does not take the uncertain maintenance environments into account. For this reason, some works try to solve this problem through the fuzzy logic. For instance, Ning et al. [13] provide a learning system to predict software erosion. However, that work ignores the recommended maintenance actions, and in addition it focuses on application server. Mittal et al. [11, 12] provide a fuzzy logic technique to measure the maintainability level of software systems, but they do not find out the best set of maintenance actions either.

In contrast, this paper proposes a fuzzy rule-based system to detect the level of a set of software erosion symptoms, as well as to recommend the most appropriate set of actions depending on the recognized symptoms. The main advantage of our proposal is that it not only employs fuzzy logic in the input (erosion symptoms), but also the output (maintenance actions) are treated through fuzzy logic. The uncertainty level of maintenance environments is therefore reduced by means of our proposal.

III. FUZZY INFERENCE SYSTEM

The proposed FIS system consists of a fuzzy rule-based system. This system considers metrics related to software erosion symptoms as inputs and provides a set of recommended actions as outputs. The architecture of the proposed FIS system (see Figure 1) consists of five sub-systems organized in three levels:

<table>
<thead>
<tr>
<th>Sint.</th>
<th>Metric</th>
<th>Description</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clone Programs</td>
<td>There are duplicate programs from a functional viewpoint</td>
<td>Indentity most update version and remove the remaining clones</td>
<td></td>
</tr>
<tr>
<td>Obsolete Programs</td>
<td>There are source code files without its corresponding executable file</td>
<td>Remove obsolete programs</td>
<td></td>
</tr>
<tr>
<td>Sourceless Programs</td>
<td>There are executables files without its corresponding source file to be maintained</td>
<td>Rewrite source code by means of reverse engineering</td>
<td></td>
</tr>
<tr>
<td>Dead Data</td>
<td>There are created data that are not used by the programs</td>
<td>Remove pieces of source code that create dead data</td>
<td></td>
</tr>
<tr>
<td>Dead Code</td>
<td>There are pieces of source code that cannot be reached by the control flow</td>
<td>Remove dead code</td>
<td></td>
</tr>
<tr>
<td>Missing Documentation</td>
<td>There are pieces of source code without documentation</td>
<td>Re-document through reverse engineering</td>
<td></td>
</tr>
<tr>
<td>Missing Functionalities</td>
<td>There are some functionalities that are not met for any program</td>
<td>Create, split or modify programs to support those functionalities</td>
<td></td>
</tr>
<tr>
<td>Poor Lexicon</td>
<td>There are data and programs with inconsistent names</td>
<td>Rename and refactor</td>
<td></td>
</tr>
<tr>
<td>Pathological Files</td>
<td>There are files that can be created or modified by several programs</td>
<td>Refactor by means of reverse engineering</td>
<td></td>
</tr>
<tr>
<td>Control Data</td>
<td>There are data that control the execution flow of several programs</td>
<td>Refactor by removing control data</td>
<td></td>
</tr>
<tr>
<td>Useless Data</td>
<td>There are external data (e.g., databases) that are not used for any program</td>
<td>Remove programs that create obsolete data</td>
<td></td>
</tr>
<tr>
<td>Obsolete Data</td>
<td>There are external data files created by a program that are not updated by any program</td>
<td>Remove programs that create obsolete data</td>
<td></td>
</tr>
<tr>
<td>Semantic Redundant Data</td>
<td>There are external data semantically equal or contained in other one</td>
<td>Remove synonym data</td>
<td></td>
</tr>
<tr>
<td>Computational Redundant Data</td>
<td>There are derived data that are calculated with the same value in database</td>
<td>Remove equivalent external data</td>
<td></td>
</tr>
</tbody>
</table>
• **Metric level** offers the input of the system, and is defined by the precise values measured from software systems according to the metrics presented in TABLE I. This level does not organize any subsystem.

• **Symptom level** adapts each precise metric value to a specific fuzzy set. These fuzzy variables are the inputs of four fuzzy rule-based subsystems, i.e., one subsystem for each software erosion symptom (see TABLE I). Subsystems establish fuzzy rules to obtain the four fuzzy values for each symptom, i.e., pollution, missing knowledge, coupling and anomalous data (see Figure 1).

• **Diagnosis level** defines a finite set of six maintenance actions according to the Vissagio framework (see Figure 1). This level contains the last subsystem, which takes the symptoms variables (obtained in the previous level) as input and generates fuzzy values concerning the maintenance actions as output (see Figure 1).

The following subsections presents in detail the five subsystems grouped into the symptom level and diagnosis level (see Figure 1). Firstly, Subsection III.A provides the fuzzy set inputs for the four first levels related to erosion symptoms. Secondly, Subsection III.B presents the fuzzy set definition for the maintenance actions, the output of the last subsystem. Finally, Subsection 0 specifies all the rules concerning the five subsystems.

### A. Software Erosion Symptoms and Input Variables

The software erosion symptoms level (see Figure 1) consists of four FIS subsystems. Each subsystem is in charge of evaluation of a symptom. Our proposal is based in the framework proposed by Vissagio [19], which characterizes the erosion of software systems by four symptoms: pollution, missing knowledge, coupling and anomalous data.

Each subsystem takes certain values since according to a set of metrics, which are directly quantifiable by observing software systems. Both specific metrics (as input) and erosion symptoms (as outputs) are defined as a fuzzy set. TABLE II shows the fuzzy set definition of all the metrics. Each fuzzy sets defines its domain between 0 and 1, provides a set of linguistic labels, and specifies the specific trapezoids in numerical and graphical way for each label. TABLE III specifies the fuzzy sets for the subsystem outputs related to each software erosion symptom.

The identified fuzzy sets were defined in a heuristic manner by taking into account the opinion of several maintenance experts. After collect information from different experts, that information was merged using the Delphi technique [15], which allows achieving a workable consensus within time limits. Despite this proposal, different fuzzy sets and linguistic labels could be proposed for the FIS system. Indeed, the mechanism employed in this paper to obtain the fuzzy sets is not the most appropriate. For instance, a better definition could imply a massive collection of information about real-life maintenance and development projects. Thereby, the fuzzy set could be obtained from this information through statistical analyses, or by means of learning systems based on fuzzy logic [1]. However, the definition of the optimal fuzzy sets is outside of the scope of this paper.

Moreover, domains of all the fuzzy sets are defined between 0 and 1 considering the density ratio of each metric, since software systems can have different sizes. For instance, the cloned program metric is defined as the number of duplicate programs divided into the total number of programs...
of the software system (i.e., it represents the percentage of cloned programs). The remaining of metric values are accurately calculated in a similar way.

TABLE II. Fuzzy Sets for the Input Metrics

<table>
<thead>
<tr>
<th>St</th>
<th>Metric</th>
<th>Linguistic Label</th>
<th>Trapezoids</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Clone Programs</td>
<td>Low, Medium, High</td>
<td>([-0.5 0.0 0.5], [0.5 1 1.5])</td>
</tr>
<tr>
<td></td>
<td>Obsolete Programs</td>
<td>[Low, Medium, High]</td>
<td>([-0.5 0.0 0.5], [0.5 1 1.5])</td>
</tr>
<tr>
<td></td>
<td>Sourceless Programs</td>
<td>[Low, Medium, High]</td>
<td>([-0.5 0.0 0.5], [0.5 1 1.5])</td>
</tr>
<tr>
<td></td>
<td>Dead Data</td>
<td>[Null, Low, Medium, High, Maximum]</td>
<td>([-0.25 0.0 0.25], [0.5 0.75 1], [0.75 1 1.25])</td>
</tr>
<tr>
<td></td>
<td>Dead Code</td>
<td>[Null, Low, Medium, High, Maximum]</td>
<td>([-0.25 0.0 0.25], [0.5 0.75 1], [0.75 1 1.25])</td>
</tr>
<tr>
<td></td>
<td>Missing Documentation</td>
<td>Low, Medium, High</td>
<td>([-0.5 0.0 0.5], [0.5 1 1.5])</td>
</tr>
<tr>
<td></td>
<td>Missing Functionalities</td>
<td>Low, Medium, High</td>
<td>([-0.5 0.0 0.5], [0.5 1 1.5])</td>
</tr>
<tr>
<td></td>
<td>Poor Lexicon</td>
<td>Low, Medium, High</td>
<td>([-0.5 0.0 0.5], [0.5 1 1.5])</td>
</tr>
<tr>
<td></td>
<td>Pathological Files</td>
<td>[Null, Low, Medium, High, Maximum]</td>
<td>([-0.25 0.0 0.25], [0.5 0.75 1], [0.75 1 1.25])</td>
</tr>
<tr>
<td></td>
<td>Control Data</td>
<td>[Null, Low, Medium, High, Maximum]</td>
<td>([-0.25 0.0 0.25], [0.5 0.75 1], [0.75 1 1.25])</td>
</tr>
<tr>
<td></td>
<td>Useless Data</td>
<td>Low, Medium, High</td>
<td>([-0.5 0.0 0.5], [0.5 1 1.5])</td>
</tr>
<tr>
<td></td>
<td>Obsolete Data</td>
<td>Low, Medium, High</td>
<td>([-0.5 0.0 0.5], [0.5 1 1.5])</td>
</tr>
<tr>
<td></td>
<td>Semantic Redundant Data</td>
<td>Low, Medium, High</td>
<td>([-0.5 0.0 0.5], [0.5 1 1.5])</td>
</tr>
<tr>
<td></td>
<td>Computational Redundant Data</td>
<td>Low, Medium, High</td>
<td>([-0.5 0.0 0.5], [0.5 1 1.5])</td>
</tr>
</tbody>
</table>

B. Maintenance Actions and Output Variables

The diagnosis level depicted in Figure 1 contains a sole FIS subsystem. This subsystem takes as input the fuzzy outputs of symptoms subsystems of the previous level, and it generates as final output the fuzzy values for each maintenance action. For this purpose, six predefined actions (identified from A1 to A6) have been selected according to the Vissagio framework (see TABLE I).

- **A1. Rewrite source code through reverse engineering.** Reverse engineering techniques are used to discover embedded or missing knowledge. After that, new executable source code is generated. This action mainly addresses the pollution and missing knowledge symptom.
- **A2. Re-document through reverse engineering.** This action is very similar than previous one, however the objective of reverse engineering is to extract meaningful information about the software system instead source code generation. This action deals with the missing knowledge symptom.
- **A3. Remove dead code and dead data.** This action analyses and removes lines of dead code as well as data that are not reached. This action mainly deals with the pollution symptom.
- **A4. Refactoring / restructuring.** This action reorganizes and restructures the software system in order to deal with any erosion symptom.
- **A5. Remove anomalous programs.** This action remove programs in order to eradicate erroneous programs, i.e., duplicated programs, obsolete programs, programs that generate pathological files, and so on. It mainly addresses the pollution and anomalous data symptom.
- **A6. Remove redundancies.** This action removes those redundant data from the semantic and computational viewpoint, thus it deals with the anomalous data symptom.

TABLE III. Fuzzy Sets for the Erosion Symptoms

<table>
<thead>
<tr>
<th>Erosion Symptom</th>
<th>Linguistic Label</th>
<th>Trapezoids</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pollution</td>
<td>Low, Medium, High, Maximum</td>
<td>([-0.25 0.25 0.25], [0.25 0.5 0.75], [0.5 0.75 1], [0.75 1 1.25])</td>
</tr>
<tr>
<td>Missing Knowledge</td>
<td>Low, Medium, High</td>
<td>([-0.5 0.0 0.5], [0.5 1 1.5])</td>
</tr>
<tr>
<td>Coupling</td>
<td>Low, Medium, High, Maximum</td>
<td>([-0.25 0.25 0.25], [0.25 0.5 0.75], [0.5 0.75 1], [0.75 1 1.25])</td>
</tr>
<tr>
<td>Anomalous Data</td>
<td>Low, Medium, High</td>
<td>([-0.5 0.0 0.5], [0.5 1 1.5])</td>
</tr>
</tbody>
</table>

TABLE IV. Fuzzy Sets for the Maintenance Actions

<table>
<thead>
<tr>
<th>Maintenance Action</th>
<th>Linguistic Label</th>
<th>Trapezoids</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1. Rewrite source code through reverse engineering.</td>
<td>Low, Medium, High</td>
<td>([-0.5 0.0 0.5], [0.5 0.5 1], [0.5 1 1.5])</td>
</tr>
<tr>
<td>A2. Re-document through reverse engineering.</td>
<td>Low, Medium, High</td>
<td>([-0.5 0.0 0.5], [0.5 0.5 1], [0.5 1 1.5])</td>
</tr>
<tr>
<td>A3. Remove dead code and dead data.</td>
<td>Low, Medium, High</td>
<td>([-0.5 0.0 0.5], [0.5 0.5 1], [0.5 1 1.5])</td>
</tr>
<tr>
<td>A4. Refactoring / restructuring.</td>
<td>Low, Medium, High</td>
<td>([-0.5 0.0 0.5], [0.5 0.5 1], [0.5 1 1.5])</td>
</tr>
<tr>
<td>A5. Remove anomalous programs.</td>
<td>Low, Medium, High</td>
<td>([-0.5 0.0 0.5], [0.5 0.5 1], [0.5 1 1.5])</td>
</tr>
<tr>
<td>A6. Remove redundancies.</td>
<td>Low, Medium, High</td>
<td>([-0.5 0.0 0.5], [0.5 0.5 1], [0.5 1 1.5])</td>
</tr>
</tbody>
</table>
These actions can fully or partially address one or more erosion symptoms. The output of the last subsystem provides a membership value for each action. In order to establish fuzzy rules between symptoms and the set of actions, these actions must be defined as fuzzy sets. TABLE IV shows the fuzzy sets defined in the proposed FIS system. These fuzzy sets were also established using the Delphi technique by involving information provided by maintenance experts.

C. Fuzzy Rules

Finally, to complete the definition of the FIS system, a set of fuzzy rules must be established to able the inference of an entire diagnosis of a software system. Fuzzy rules are if-then rules with a three-part process: Firstly, the precise metric values are fuzzified with the metric fuzzy sets (see TABLE II). Thus, all fuzzy inputs in the antecedent are resolved as a membership degree between 0 and 1. If the antecedent involves only one fuzzy set, then this is the degree of support for the rule. Secondly, fuzzy logic operators (e.g. and, or) are applied to multiple part antecedents. These operators resolve the antecedent to a single number between 0 and 1, which is the degree of support for the rule. Thirdly, the degree of support for the entire rule is used to shape the output fuzzy set, i.e., the consequent of the fuzzy rule. If the antecedent is partially true (i.e., its membership value is lower than 1), then the output fuzzy set is truncated according to the implication method.

On the one hand, a set of fuzzy rules are established to define the outputs related to the four symptoms subsystems: pollution, missing knowledge, Coupling and Anomalous Data.

Pollution fuzzy rules:

1. If (CloneProg is low) and (SourcelessProg is low) then (Pollution is null)
2. If (CloneProg is medium) and (SourcelessProg is low) then (Pollution is low)
3. If (CloneProg is high) and (SourcelessProg is low) then (Pollution is high)
4. If (CloneProg is medium) and (SourcelessProg is medium) then (Pollution is high)
5. If (CloneProg is medium) and (SourcelessProg is high) then (Pollution is maximum)
6. If (CloneProg is high) and (SourcelessProg is high) then (Pollution is maximum)
7. If (ObsoleteProg is low) and (DeadData is null) and (DeadCode is null) then (Pollution is null)
8. If (ObsoleteProg is medium) and (DeadData is low) and (DeadCode is low) then (Pollution is low)
9. If (ObsoleteProg is medium) and (DeadData is medium) and (DeadCode is medium) then (Pollution is medium)
10. If (ObsoleteProg is medium) and (DeadData is high) and (DeadCode is high) then (Pollution is high)
11. If (ObsoleteProg is medium) and (DeadData is maximum) and (DeadCode is maximum) then (Pollution is high)
12. If (ObsoleteProg is high) and (DeadData is maximum) and (DeadCode is maximum) then (Pollution is maximum)
13. If (Obsoleteprogs is high) and (DeadData is high) and (DeadCode is high) then (Pollution is high)
14. If (Obsoleteprogs is high) and (DeadData is medium) and (DeadCode is medium) then (Pollution is medium)
15. If (CloneProg is high) and (Obsoleteprogs is high) and (SourcelessProg is high) then (Pollution is maximum)
16. If (CloneProg is low) and (Obsoleteprogs is low) and (SourcelessProg is low) and (DeadData is null) and (DeadCode is null) then (Pollution is high)

Missing Knowledge fuzzy rules:

1. If (MissingDocument is low) and (MissingFunc is low) and (PoorLexicon is low) then (MissingKnowledge is low)
2. If (MissingDocument is low) and (MissingFunc is low) and (PoorLexicon is low) then (MissingKnowledge is low)
3. If (MissingDocument is low) and (MissingFunc is medium) and (PoorLexicon is low) then (MissingKnowledge is medium)
4. If (MissingDocument is medium) and (MissingFunc is medium) and (PoorLexicon is low) then (MissingKnowledge is medium)
5. If (MissingDocument is medium) and (MissingFunc is medium) and (PoorLexicon is medium) then (MissingKnowledge is medium)
6. If (MissingDocument is high) and (MissingFunc is medium) and (PoorLexicon is low) then (MissingKnowledge is medium)
7. If (MissingDocument is not low) and (MissingFunc is high) and (PoorLexicon is not low) then (MissingKnowledge is high)
8. If (MissingDocument is low) and (MissingFunc is high) and (PoorLexicon is not low) then (MissingKnowledge is medium)

Coupling fuzzy rules:

1. If (PathologicalFiles is not maximum) and (ControlData is maximum) then (Coupling is maximum)
2. If (PathologicalFiles is high) and (ControlData is maximum) then (Coupling is maximum)
3. If (PathologicalFiles is high) and (ControlData is high) then (Coupling is maximum)
4. If (PathologicalFiles is medium) and (ControlData is high) then (Coupling is high)
5. If (PathologicalFiles is medium) and (ControlData is medium) then (Coupling is medium)
6. If (PathologicalFiles is low) and (ControlData is medium) then (Coupling is medium)
7. If (PathologicalFiles is low) and (ControlData is low) then (Coupling is low)
8. If (PathologicalFiles is null) and (ControlData is low) then (Coupling is low)
9. If (PathologicalFiles is null) and (ControlData is null) then (Coupling is null)

Anomalous Data fuzzy rules:

1. If (SemRedundant is not high) and (CompRedundant is not high) then (AnomalousData is not high)
2. If (SemRedundant is high) and (CompRedundant is high) then (AnomalousData is medium)
3. If (SemRedundant is medium) and (CompRedundant is not medium) then (AnomalousData is medium)
4. If (SemRedundant is high) and (CompRedundant is not high) then (AnomalousData is high)
5. If (UselessData is not low) and (ObsoleteData is not low) and (SemRedundant is not low) and (CompRedundant is not low) then (AnomalousData is not low)
6. If (UselessData is high) and (ObsoleteData is high) and (SemRedundant is not low) and (CompRedundant is not low) then (AnomalousData is high)
7. If (UselessData is medium) and (SemRedundant is high) then (AnomalousData is high)
8. If (ObsoleteData is medium) and (SemRedundant is high) then (AnomalousData is high)
9. If (ObsoleteData is medium) and (CompRedundant is high) then (AnomalousData is high)

On the other hand, after establishing fuzzy rules concerning the FIS subsystems of symptom level, the set of rules of the action subsystem must be also defined. These rules were established in the similar way than the previous sets.

Maintenance Actions fuzzy rules:

1. If (Pollution is not low) then (A1 is high) and (A3 is medium) and (A5 is medium)
2. If (MissingKnowledge is high) then (A1 is medium) and (A2 is medium) and (A4 is low)
3. If (MissingKnowledge is not low) and (AnomalousData is not low) then (A4 is medium)
4. If (AnomalousData is high) then (A5 is medium) and (A6 is high)
5. If (AnomalousData is not high) then (A5 is low) and (A6 is medium)
6. If (Pollution is medium) and (AnomalousData is not low) then (A4 is low) and (A6 is high)
7. If (Coupling is not low) then (A4 is alto) and (A5 is medium)
8. If (Coupling is high) then (A4 is alto) and (A5 is low)
9. If (Pollution is low) and (MissingKnowledge is low) then (A1 is low) and (A2 is low)
10. If (Pollution is low) then (A1 is low)

IV. IMPLEMENTATION

This paper also provides an implementation of the proposed FIS system in order to demonstrate its feasibility and facilitate its adoption. The FIS system has been implemented using the
The FIS system was implemented using the Fuzzy Logic Toolbox of Matlab [17]. In addition, the FIS’s real performance has been also simulated by means of Simulink [18], another Matlab module.

The implementation of the FIS system using the Fuzzy Logic Toolbox was carried out following a set of steps for each aforementioned FIS subsystem. Due to space limitations, it focuses in the Coupling subsystem to illustrate the entire implementation process.

1. The inputs and output of the FIS system are firstly selected. Relationships between the input fuzzy variables and output fuzzy variables are established. This task can be easily carried out by means of the FIS Editor of Matlab (see Figure 2). The type of the FIS subsystem can be also selected by means of that editor. All subsystems are based on the Mandami controller configuration [9, 10].

2. In the second step, input and output fuzzy variables are defined by means of fuzzy sets. Each fuzzy variable has several linguistic labels and each label is defined as a fuzzy set. This task is also performed through the Matlab FIS editor according to the fuzzy variables presented in Section III. For instance, Figure 3 shows the definition of the fuzzy set for the ‘null’ linguistic label within ‘PathologicalFiles’ input variable in the Coupling subsystem.

3. The third step involves the fuzzy rules definition. This task establishes specific relationship through the linguistic labels of both input and output fuzzy variables. Matlab also provides a graphical editor to easily generate fuzzy rules.

4. Finally, the logic operators as well as the implication and defuzzification operations must be established through the Matlab FIS editor (see Figure 2). The ‘and’ and ‘or’ logic operators are used in antecedents of fuzzy rules to combine several fuzzy sets. In this case we respectively select the minimum and maximum operators to combine several fuzzy sets. These operators are well-known and commonly used to represent the intersection (and) and union (or) of fuzzy sets. Moreover the implication function is selected in this step. The implication function is used to obtain a membership value in the consequent fuzzy set from the membership value of the antecedent. This FIS system uses the minimum operation. Another operation that must be established is the aggregation function. This function merges all membership values obtained in a particular fuzzy set that appears in consequents of several rules. The FIS system uses the maximum operation since it guarantees to obtain the higher value obtained for a particular fuzzy set. Finally, a defuzzification function must be selected to obtain a real value between 0 and 1 from the aggregated area obtained in a particular consequent fuzzy set. The proposed FIS system uses the centroid function, which use the weighted average of a few data points in the aggregated area.

To simulate the FIS system implemented though the Matlab Fuzzy Logic Toolbox, we use Simulink [18]. Simulink is an environment for multi-domain simulation and model-based design for dynamic and embedded systems. It makes it possible to simulate, implement, and test a variety of time-varying systems.

Figure 4 shows the simulation model of the proposed FIS system, which is built in Simulink as a fuzzy logic controller following the architecture presented in Figure 1. Firstly, the fuzzy logic controller will receive the precise values of the all maintenance metrics, and it then will trigger the respective fuzzy rules of the four subsystems in the symptom level. Outputs of the symptom subsystems are used as the input for the maintenance subsystem, which generates the membership degree (as a value between 0 and 1) to each of the six possible maintenance actions. In addition, the controller filters out
actions under the aforementioned maintenance threshold, which can be established by maintenance experts in each company. As a consequence, the controller indicates by means a light what actions should be carried out.

In order to simulate the performance of the FIS system, a real-life software system was evaluated. The software system named VillasanteLab manages the operation of a Spanish company in the water and waste industry. VillasanteLab system manages information related to chemical laboratories, customers and products such as chemical analysis, dilutions, and chemical calibrations. The analyses supported by the application examined different parameters, including a large number of physical, chemical and microbiological parameters according to current regulations and laws for controlling water quality. From a technological point of view, VillasanteLab system is a traditional web application developed using Java platform and consists of 28,800 lines of source code. This software system was released for four years ago, and it has had three major modifications with seven medium modifications in total. Therefore, the VillasanteLab system probably has been eroded over time.

Firstly, all the maintenance metrics used in our FIS system were accurately evaluated. TABLE V shows the fourteen metric values obtained after evaluating the VillasanteLab system. These values were introduced in the simulation model and the model was then executed in Simulink (see Figure 4). The obtained results for the erosion symptoms were respectively 0.2, 0.5, 0.5 and 0.4; and the final results concerning maintenance actions were 0.80, 0.50, 0.43, 0.55, 0.41 and 0.5 (see TABLE V). In addition, the fuzzy rules executed as well as the fuzzy sets values were obtained during simulation (see Figure 5).

The configuration of simulation considers a maintenance threshold of 0.5. Therefore, the recommended maintenance actions for the VillasanteLab system were A1. Rewrite source code through reverse engineering as well as A4. Refactoring / restructuring.

VI. CONCLUSIONS

This paper presented a fuzzy rule-based system (also known as Fuzzy Inference System) to find out the software erosion symptoms of a concrete software system. The proposed FIS system does not only diagnose the kind of software erosion, but also recommend the best treatment for the eroded system. We refer to “the best treatment” as a set of maintenance actions that must be carried out to reduce or mitigate the software those erosion symptoms without incurring in more erosion itself.

The design of the FIS system was mainly divided into two levels. The first level evaluates the erosion symptoms, which consists of four subsystems (one for each symptom). Each subsystem accepts as input metrics accurately measured from the software system. These precise values are then fuzzified and a set of fuzzy rules establish as output the membership degrees for each symptom subsystem. The second level has only one subsystem, which takes the fuzzy values of each symptom subsystem and it obtains the membership degrees for each candidate maintenance action.
In order to validate the feasibility of our proposal and its adoption, the proposed FIS system was also implemented in Matlab using the fuzzy logic toolbox. Furthermore, the implemented system was also simulated through Simulink. The advantage of the implemented simulation system is that maintainers can know the best set of maintenance actions to be carried out. The set of actions is prioritized, which helps maintainers to distribute maintenance resources and efforts. In addition, maintainers can estimate the reduction of erosion metrics in the hypothetical case to carry out the recommended actions. As a consequence, they might simulate a set of iterations of incremental maintenance and predict by means of our system the expected reduction of erosion symptoms. If a simulation shows that the software system is much eroded and a lot of maintenance effort will be necessary, the system could be discarded without starting the maintenance process. In conclusion, the proposed FIS system supports the decision-making process in the maintenance stage, and it can save maintenance cost by means of the proposed simulation model.

Moreover, the simulation of our system allowed us to detect potential improvements for the FIS system. Those improvements were progressively applied to adjust the definition of fuzzy sets. In addition, the rules were established by experts according to the Delphi technique and they were progressively improved through the simulation. However, the Delphi technique adopted to establish the rules is not the best approach, since it has a heuristic nature. As a consequence, we propose as future work the improvement of the fuzzy rules by using statistical techniques from real-life maintenance data or learning systems to find out the optimum fuzzy rules.

ACKNOWLEDGMENT

This work was supported by the FPU Spanish Program; by the R+D projects funded by JCCM: ALTAMIRA (PII2109-0106-2463) and PRALIN (PAC08-0121-1374); and the PEGASO/MAGO project (TIN2009-13718-C02-01) funded by MICINN and FEDER.

REFERENCES


