2015 IEEE/ACM 4th International Workshop on Realizing AI Synergies in Software Engineering

Towards Interpretable Defect-Prone Component Analysis using Genetic Fuzzy Systems

Themistoklis Diamantopoulos and Andreas Symeonidis
Electrical and Computer Engineering Dept., Aristotle University of Thessaloniki
Thessaloniki, Greece
thdiaman@issel.ee.auth.gr, asymeon@eng.auth.gr

***Abstract*—The problem of Software Reliability Prediction is attracting the attention of several researchers during the last few years. Various classification techniques are proposed in current literature which involve the use of metrics drawn from version control systems in order to classify software components as defect-prone or defect-free. In this paper, we create a novel genetic fuzzy rule-based system to efficiently model the defect-proneness of each component. The system uses a Mamdani-Assilian inference engine and models the problem as a one-class classification task. System rules are constructed using a genetic algorithm, where each chromosome represents a rule base (Pittsburgh approach). The parameters of our fuzzy system and the operators of the genetic algorithm are designed with regard to producing interpretable output. Thus, the output offers not only effective classification, but also a comprehensive set of rules that can be easily visualized to extract useful conclusions about the metrics of the software.**

***Index Terms*—Software Reliability Prediction; defect-prone components; software fault prediction; genetic fuzzy systems;**

I. Introduction

Lately, several researchers focus on *Software Reliability Prediction (SRP)*. Although the area is quite broad, the main line of work is similar. The problem at hand focuses on discovering defect-prone components in a software project either pre-release or post-release. Thus, the main aim is to predict whether a software component contains code that either has defects or could potentially evolve such. The term "component" may refer to a class or a package (in an object-oriented perspective), however higher-level components or even "inter-project" level components (e.g. software projects or web services) are also possible.

In the context of Software Reliability, current approaches focus on predicting defect-prone components using several quantitative metrics that are known to be satisfactory predictors of reliability [1]. These metrics may concern method-level and class-level representations of software structure [2], [3], or change metrics [4], [5]. The former include metrics such as the number of lines of code per file/class, the depth of the inheritance tree for a class, etc., whereas the latter employ metrics drawn from revisions as kept by version control systems, thus including the number of times a file has been refactored, the number of authors that committed a file, the number of lines that were changed for each file, etc.

In this paper, we focus on an interesting type of metrics, introduced by Nagappan et al. [6], that fall in the latter category. The authors define the notion of "change bursts", i.e. consecutive changes in files, in order to effectively associate the procedure of writing fast (and perhaps "sloppy") code for a component with the possibility that the component is defect-prone. Their approach indicates that change bursts are indeed effective predictors of reliability.

Apart from the metrics involved, another important aspect of SRP involves the algorithms employed in order to determine defect-prone components. Several algorithms have been proposed in this area, and they have proven effective in identifying defect-prone components [1], [6]. However, they produce complex models that fail to describe the effect of the metrics on the class attribute in a comprehensive manner.

In this paper, we design an interpretable system that reflects the effect of different metrics in the defect-proneness of each component. Given a change bursts dataset (which is an effective way of representing the data [6]), we design a fuzzy rule-based system that describes the contribution of the metrics of the dataset, in an interpretable manner. We design our classifier using a *Genetic Algorithm (GA)*, in order to learn an optimal and at the same time comprehensive set of rules. In addition, we visualize the rule base of our classifier illustrating its potential when a comprehensive set of rules is required.

Section II provides background knowledge on classification using fuzzy systems and using GAs to learn fuzzy rules. Section III reviews related work in the area of SRP, focusing on the interpretability of the techniques used in this area. Section IV focuses on the problem confronted and presents the dataset used. Our methodology for building a novel genetic fuzzy system is illustrated in V and is evaluated in Section VI, both in terms of efficiency and in terms of interpretability. Finally, Section VII concludes this paper providing useful remarks and insight for future research.

II. Background

This section provides essential background information on the fields of *Fuzzy Logic (FL)* theory and GAs. Subsection II-A focuses on the design of fuzzy rule-based systems in order to solve the problem of classification, and subsection II-B describes how GAs can be applied to the problem in order to devise/optimize the rules of the classifier.

A. Fuzzy Rule-Based Systems for Classification

According to *Fuzzy Logic (FL)*, problems are modeled and solved with regard to their vagueness or *fuzziness*. Variables

978-1-4673-7064-6/15 $31.00 © 2015 IEEE
DOI 10.1109/RAISE.2015.13

are described using fuzzy sets and are assigned *linguistic* terms (e.g. low, average) according to a *membership function*, that denotes the degree to which an element belongs to the set.

Apart from describing imprecise (and sometimes uncertain) data, FL theory has been used for control, or even regression and classification problems. This is accomplished using a so called *fuzzy rule-based system*[1]. Designing a fuzzy system comes down to determining a *rule base* and a *fuzzy inference engine*. Concerning the rule base, one can devise appropriate rules by means of expert help, or by learning rules from data (more about this in the next subsection). An example rule is:

IF x_1 is A AND x_2 is B THEN y is C

where x_1 is A, x_2 is B are the *antecedents* of the rule, denoting that the input variables x_1 and x_2 belong to the fuzzy sets A and B respectively. The *consequent* of the rule, y is C denotes that the output variable y belongs to the set C. Fuzzy rules are usually preferred when the interpretability of the final system is important. Intuitively, a rule is more interpetable when it has fewer antecedents.

Fuzzy inference is performed according to the desired output and can lead to: a) membership on an output fuzzy set or b) an arithmetic value. The former are implemented with *Mamdani-Assilian (MA)* controllers [7] and the latter with *Takagi-Sugeno (TS)* controllers [8].

B. Learning Rules using Genetic Algorithms

Designing a rule base can be quite a tedious task, possibly requiring human expert input. Since expert help may not be available, several researchers have focused on deriving a rule base given a known set of inputs and outputs [9]. In the case of a fuzzy classifier, this can be seen as training the system.

GAs are inspired by the evolution theory of biology. When designing a GA, one must decide on the form of each *individual*, i.e. its *chromosome*, and the form of the operations performed on the *population* of individuals, i.e. *crossover*, *mutation*, and *selection*. In the case of learning rules, there are three approaches, the Michigan approach [10], the Pittsburgh approach [11], and the iterative rule learning approach [12]. The Michigan approach models each rule as an individual, whereas in a Pittsburgh-style classifier, each individual is a complete rule base. The iterative rule learning approach is similar to the Michigan approach, however the algorithm iteratively extracts the best individual/rule of the rule base and determines the final rule base by reasoning among the selected individuals.

Upon following one of the aforementioned approaches, the problem is reduced to defining the population as well as the evolution and fitness operators. Note that GAs can be used to optimize a variety of parameters i.e. to evolve the rule base or optimize both the rules and the membership functions.

[1]Also known as *fuzzy control system*, or *fuzzy controller*, or even simply as *fuzzy system*. The terms are used interchangeably in this paper.

III. Related Work

As noted in Section I, several metrics have been designed in order to estimate the reliability of software components. The problem can be formulated either as classification, i.e. classifying each component as defect-prone or defect-free, or as regression, where the desired outcome is a ranked list of components based on the defect-proneness of each component [1]. Although ranking methods can also be used, they deviate from the scope of this work, thus the reader is referred to [13] for more information. Concerning regression, early approaches in the field involved regression systems that model the correlation between the metrics and the defects of a component [3]. Currently, researchers use more complex models, such as Linear Regression [5]. Concerning classification, which is more relevant to our work, several algorithms have been applied to the problem. Zimmermann et al. [14] use Logistic Regression to classify files as defect-prone or defect-free given class-level metrics, while Menzies et al. [15] showed that Naïve Bayes is quite effective for detecting complex components, which as they claim, are also the most defect-prone.

Although the aforementioned approaches are effective, they result in black-box models, that are hard to interpret. Other approaches involve the use of decision trees [4], [16] for defect prediction, since decision trees are generally considered easy to interpret. However, trees are interpretable only under certain circumstances; when there are numerous metrics, scalar rule antecedents of the form $x_1 \leq 45.5$ or $x_2 \geq 28.7$ are almost as interpretable as a regression model. An interesting approach by Knab et al. [16] involves generating trees so that the distance of each metric from the root denotes its importance. Although the derived trees are interpretable, most information, such as the value ranges of these metrics or the correlation among them is absent from this representation.

A paradigm descriptive enough to retain much information without compromising effectiveness is the use of fuzzy classifiers. One of the most indicative approaches is the one by Khalsa [17]. The author uses a MA controller on the C&K metrics [2]; this approach includes a rather large rule base, however with only three metric variables. Thus, although the rule base is indeed comprehensive, the system cannot easily scale to e.g. 14 rules as in this paper.

Other approaches include also the TS fuzzy model designed by Aljahdali and Sheta [18]. However, the system described by the authors is applicable on predicting faults given fault databases from multiple projects, thus the scenario largely differs from ours both in terms of input variables and in terms of the desired output, which is a reliability growth model. Pandey and Goyal [19] also attempted to design a comprehensive fuzzy model, however for a high-level set of metrics set by experts (e.g. design team experience). Although these metrics are usually used for cost and effort estimation[2], we refer to it in this section since its purpose is fault prediction.

[2]Software cost or effort estimation is a similar area to SRP. Although several efforts have been made towards designing fuzzy cost estimation models, those are omitted here since they deviate from the main area of this paper.

Finally, although there are efforts towards reliability prediction using GAs, the latter are used in order to optimize the parameters of other (non-fuzzy) models, such as the architecture of neural networks [20] or the parameters of auto-regressive models [21]. To the best of our knowledge, there is no other research effort towards designing a comprehensive fuzzy rule-based system, either using GA or not, for the version-level metrics and, especially the burst metrics defined in [6].

IV. Metrics and Dataset Employed

Given that software is developed in a number of consecutive *builds* (commits), i.e. one build immediately after another, Nagappan et al. [6] define a consecutive number of changes in a component of the software as a *change burst*. The authors define the minimum size of a burst, *burst size*, and the maximum gap within a burst, *gap size*. Small gap sizes result in more and smaller bursts, as long as the burst size is large enough (more about the effect of gap and burst sizes in [6]). Thus, Nagappan et al. [6] define several metrics for each component. These metrics are summarized in Table I.

TABLE I
Change bursts metrics

Metric	Description
Change Metrics	
NumberOfChanges	Number of builds that a component changed
NumConsecChanges	Consecutive builds that a component changed
NumChangeBursts	Total number of change bursts
TotalBurstSize	Aggregate number of builds of all bursts
MaxChangeBurst	Number of builds of the largest change burst
Temporal Metrics	
TimeFirstBurst	The time that the first burst occurred
TimeLastBurst	The time that the last burst occurred
TimeMaxBurst	The time that the maximum burst occurred
People Metrics	
PeopleTotal	Number of people that changed the component
TotalPeopleInBurst	Number of people involved across all burst
MaxPeopleInBurst	Maximum number of people over all bursts
Churn Metrics	
ChurnTotal	Total number of modified lines of the component
TotalChurnInBurst	Total number of modified lines across all burst
MaxChurnInBurst	Maximum number of modified lines over all bursts

The metrics lie in four categories: (a) change metrics that refer to the changes of files within bursts, (b) temporal metrics that refer to the time of the bursts (early or late), (c) people metrics for the people involved in changes/bursts, and (d) churn metrics that measure modified lines per file. There is also a binary metric for the presence or absence of bugs in each component.

In this paper, we use the dataset of Eclipse 2.0 and evaluate our algorithms against the class-granularity version of the dataset. In total, Eclipse has 6728 classes, 626 of them are defect-prone and the rest (5753) are defect-free. Thus, our dataset consists of 6728 samples, while each of them includes 14 integer values for the attributes of Table I and a binary value for the class attribute denoting if this class has any bugs.

V. A Genetic Fuzzy System for Detecting Defect-Prone Components

This section describes a fuzzy rule base system for detecting defect-prone components as well as a GA for learning the fuzzy rules.

A. Fuzzy Rule-Based System

In our classification scenario, each metric of Table I is an independent attribute, and the class attribute is the presence or absence of bug(s) in the respective component. Although there are multiple ways to define such a problem, we decided to model it as a "one-class" classification, i.e. the class defect-prone. This has an important advantage: the final model that shall sufficiently model the data will be highly comprehensible since it will only have one consequent. The architecture of our system is shown in Figure 1:

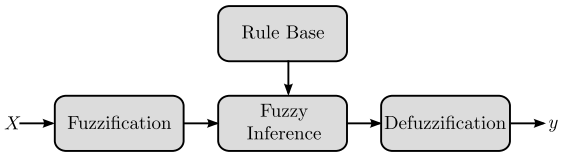


Fig. 1. Fuzzy rule-based system

The system has three stages. At first, the input variables (X) are fuzzified by finding their membership degree in a set of rules. After that, the fuzzy inference component determines the membership degree for the output given the rules and the input membership, and finally the output variable is defuzzified to produce a numeric value.

Upon normalizing all attributes values in the range $[0, 100]$, we define three triangular fuzzy sets for each attribute, given the triangular membership function:

$$\mu_{TMF}(x) = \begin{cases} 0 & \text{if } x < a \\ (x-a)\,/\,(m-a) & \text{if } a \le x \le m \\ (b-x)\,/\,(b-m) & \text{if } m < x \le b \\ 0 & \text{if } x > b \end{cases} \quad (1)$$

These sets are the fuzzy set `Low` with $a = -50$, $m = 0$, and $b = 50$, the fuzzy set `Average` with $a = 0$, $m = 50$, and $b = 100$, and the fuzzy set `High` with $a = 50$, $b = 100$, and $c = 150$. The output is in a triangular fuzzy set named `Large` (for large defect-proneness) with $a = 0$, $m = 1$, and $b = 2$.

We design a Mamdani-Assilian rule-based system to confront the classification problem as a one-class classification problem. An example rule of the system has the form:

`IF` x_1 `is Low AND` x_2 `is Average AND ... AND` x_n `is High THEN` y `is Large WITH` d

where d is the confidence degree for class `Large`. An example of our system for two variables is shown in Figure 2. The `AND` operator between two antecedents is defined as the minimum of their membership values. Each rule outputs the membership for the class `Large` of the `DefectProneness` variable. After that, the final membership for the fuzzy set of `Large`

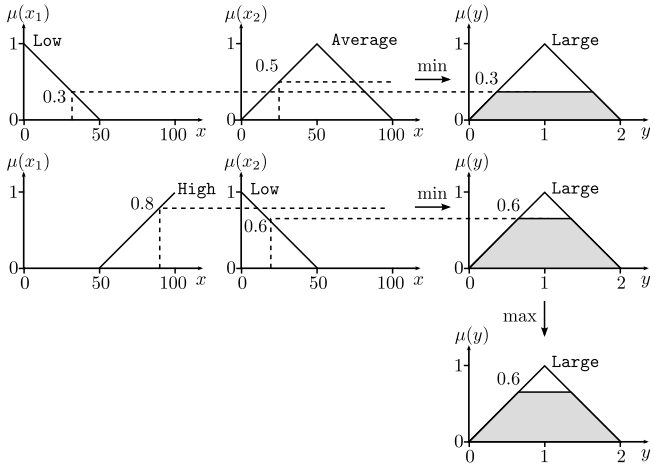


Fig. 2. An example with two rules of the fuzzy system. x_1 belongs to the set `High` with degree 0.3 and to `Average` with degree 0.7. x_2 belongs to `Average` and `Low` with degrees 0.4 and 0.6 respectively. The output of each rule is the minimum of its antecedents' degrees (0.3 for the 1st and 0.6 for the 2nd rule). The final output is their maximum 0.6.

is defined as the maximum among all membership values of all rules. Finally, the system uses the max defuzzifier operator. E.g. the component of Figure 2 is defect-prone with probability 0.6, and defect-free with probability 0.4.

B. Genetic Algorithm for Learning Fuzzy Rules

Upon defining the fuzzy system, we design a GA in order to determine an optimal rule base. Our GA is shown in Figure 3.

```
t = 0
Initialize P(t)
Evaluate P(t)
while (continue)
  t = t + 1
  Select P(t) from P(t − 1) with Tournament Selection
  Evolve P(t)
  Evaluate P(t)
```

Fig. 3. A genetic algorithm for evolving fuzzy rules

In Figure 3, t represents the current generation. In each step of the algorithm, the population $P(t)$ is evaluated according to a *fitness function* and the *fittest* individuals are selected to form the new population. These individuals evolve using the operators of crossover (which is equivalent to reproduction) and mutation (which is adaptation of the individual itself to the environment) in order to adapt to the environment. The following paragraphs define the population, the crossover and mutation operators, and the fitness function of our method.

1) Population: Our algorithm follows the Pittsburgh approach, with every individual being a complete rule base. The main advantage of the Pittsburgh approach over other approaches is that it takes into account the interactions between the rules. If we represented each rule as an individual, as in the Michigan approach or the iterative rule learning approach, then the final rule base may not cover effectively the whole dataset. Thus, we define a subchromosome which represents one rule of the system. Each gene of the subchromosome refers to the linguistic value of the respective variable; the possible values for a gene are 0, 1, 2, and 3, corresponding to the variables `Low`, `Average`, `High`, and `DontCare`. The `DontCare` operand has a membership function that is always equal to 1, hence it is not taken into account for the formulation of the rule. An example subchromosome and the respective rule is:

0	2	3	2	3	1	3	3	3	1	3	3	3	3

IF x_1 is Low AND x_2 is High AND x_4 is High AND x_6 is Average AND x_{10} IS Low

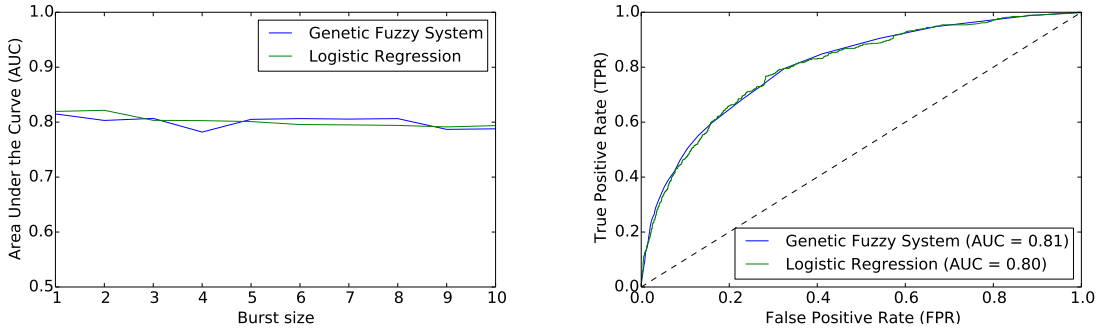
Thus, the chromosome of the system is the concatenation of several subchromosomes following the above pattern.

The initial individual is created similarly to the *learning from examples* methodology [9] that uses a training set to extract a rule base. We iterate over all defect-prone instances and create a rule for each instance. This rule is added to the rule base if it is not *similar* to the already added rules. Two subchromosomes/rules are similar if $v\%$ (in our case 50%) of their genes are the same. Thus, if the newly derived rule is not similar to any of the existing ones, it is added to the rule base. If, however, it is similar to an existing subchromosome, then the latter is adapted to account for the information of both subchromosomes. This adaptation is done by adding `DontCare` operands in the place of the genes that the two subchromosomes differ. For example, let 02120210112120 be the newly derived subchromosome; assuming that the most similar subchromosome is 02120210120020, the second subchromosome will change to 02120210133320 (switching the tenth, eleventh and twelfth genes to 3). Upon creating the initial individual, we create a population by applying mutation over this individual.

2) Crossover and Mutation: Custom crossover and mutation operators are defined to ensure that the iteration of the different rule bases is rational.

Crossover involves two individuals and produces two new (offspring) individuals by mixing the genes of the chromosomes. Known operators, such as one-point, two-point or uniform crossover, are not a good fit for the chromosomes of our scenario, since each chromosome is not only a sequence of genes, but also a sequence of rules, i.e. subchromosomes. Thus, we define a uniform crossover operator that only selects crossover points that do not split the subchromosomes. In other words, we allow swapping rules between rule bases, but we do not allow swaps within the rules themselves.

Mutation refers to changing the genes of a chromosome; the new value of each gene can be either uniformly selected from the possible values, or selected so that it is close to its old value. We use a two-step mutation function. In our case, we used a two-step mutation function for each gene. For each gene, we first assign a 50% probability of mutating the gene to the value `DontCare`; this ensures that new rules are smaller than the old ones. Next (if the first step was not activated),



(a) (b)

Fig. 4. Diagram (a) depicts the AUC for the Genetic Fuzzy System and the Logistic Regression algorithm for different burst sizes given gap size equal to 2, and diagram (b) provides the Receiver Operator Characteristic (ROC) curves of the two algorithms for gap size equal to 2 and burst size equal to 3.

we assign 50% probability of mutating the gene towards the upper value and 50% probability of mutating it towards the lower value. If there is no lower or upper value, we select again the value 3. For example, if the value of a gene is 0, it may either change to 1 or to 3, both with a 50% probability.

3) Fitness Function: The fitness function determines which individuals survive in each generation; thus it represents the desired features of our final chromosome. Our function receives as input an individual and outputs a score based on two criteria. The first criterion is the *Area Under the Curve (AUC)*. The AUC is computed by applying the rule base to the training set (a random sample of it, equal to a quarter, to avoid overfitting) and creating the *Receiver Operator Characteristic (ROC)* curve. The AUC is preferred over other metrics, such as recall or precision, since it models well the accuracy of the result both in terms of maximizing true positives and in terms of minimizing false positives.

Though effective, the AUC does not account for interpretability. As a result, we use a second criterion: the average length of each rule of the rule base. The length of each rule depends on the number of `DontCare` operands. Finally, the fitness function for a rule base RB is:

$$Fitness(RB) = AUC(RB) \cdot \left(MaxL - \sum_{R \in RB} \frac{L(R)}{|RB|} \right) \quad (2)$$

where $L(R)$ is the length of rule R (i.e. the count of non-`DontCare` operands), and $MaxL$ is the maximum rule length (in our case equal to 14).

Finally, we refrain from selecting the fittest individuals for the next generation since then the GA might converge to suboptimal solutions. Instead, tournament selection with size equal to 3 is applied, so that 3 randomly selected individuals participate in a tournament and the fittest one proceeds to the next generation. In addition. we keep a *Hall-of-Fame (HoF)*, to ensure that after the execution of the GA, the optimal solution is saved.

VI. Evaluation

In this section, we provide the results of our evaluation on the dataset described in Section IV.

A. Experimental Setup

We developed our method using the *Distributed Evolutionary Algorithms in Python (DEAP)* library [22] since it allows finegraining the GA (e.g. defining custom crossover, mutation, and fitness functions). The library also allowed us to execute our method in 8 cores, thus resulting in low execution time.

Two thirds of the data compose the training set and the remaining one third is the test set. We tested our *Genetic Fuzzy System (GFS)* against a *Logistic Regression (LR)* classifier for 100 different combinations of gap and burst sizes, i.e. gap size ranging from 1 to 10, and burst size ranging from 1 to 10. The population has size 40 and the number of generations for the GA is 40. We also defined a 0.2 probability that two individuals crossover and a 0.5 that an individual mutates. Although this configuration seems sensitive, in fact it is almost arbitrary; any configuration can lead to an optimal solution as long as the number of generations is large enough.

B. Experimental Results

In the Eclipse dataset the burst size is proportional to the gap size [6], thus bursts are isolated and cannot be merged to each other. Hence, our results for all combinations of gap and burst sizes were similar, with AUC around 0.8. Since both our GFS and the LR classifier yielded optimal results for gap size equal to 2, we provide the AUC for the two classifiers for this gap size and for burst size ranging from 1 to 10 in Figure 4a.

The burst size does not affect the performance of either classifier. Our GFS performs equally well to the LR classifier, achieving better results for half of the burst size measurements. In any case, though, this plot indicates that the AUC for the two classifiers is within the limits of statistical insignificance. This conclusion can be further explored in the ROC curves of Figure 4b, which are drawn for gap size equal to 2 and burst size equal to 3; the two curves are almost identical.

However, the main hypothesis of this paper involves not only creating an effective classifier, which as shown in Figures 4a and 4b is achieved, but also producing interpretable output. Therefore, given the same configuration of gap and burst sizes (2 and 3 respectively) which proved optimal for our GFS, we examine the rule base of the system to determine

```
 1. IF NumberOfChanges IS Average
 2. IF NumberOfChanges IS High AND NumConsecChanges IS Low
 3. IF NumberOfChanges IS Average
 4. IF TotalBurstSize IS Average AND TimeFirstBurst IS High AND TimeMaxBurst IS Low AND PeopleTotal IS Low
 5. IF NumberOfChanges IS High AND NumConsecChanges IS High AND MaxChangeBurst IS Low AND TimeMaxBurst IS
                  High AND PeopleTotal IS Low AND TotalPeopleInBurst IS Average AND ChurnTotal IS Average
 6. IF MaxChangeBurst IS Low AND MaxPeopleInBurst IS High AND MaxChurnInBurst IS Low
 7. IF NumConsecChanges IS High AND MaxChangeBurst IS Average AND TotalPeopleInBurst IS High AND ChurnTotal
                                                                IS Average AND MaxChurnInBurst IS Average
 8. IF TotalBurstSize IS High
 9. IF NumChangeBursts IS Average AND PeopleTotal IS Average
10. IF MaxChangeBurst IS High AND TotalChurnInBurst IS Average
11. IF NumberOfChanges IS Average AND NumChangeBursts IS High
12. IF NumConsecChanges IS Average AND PeopleTotal IS Low AND ChurnTotal IS Average
```

Fig. 5. Fuzzy rule base of the Genetic Fuzzy System. The consequent is always the same (`THEN DefectProneness IS Large`), so it is omitted.

whether it is comprehensive. The fuzzy rules of our GFS are shown in Figure 5. It is interesting to note that rules 1 and 3 are identical. This is not totally unexpected since similar rules are indeed penalized when creating the initial population, yet using mutation and crossover may result in similar (or in this case identical) rules. In this case, we may assume that having identical rules implies that no further training is required since any new rules will not add more information.

We can comment on the interpretability of the rule base shown in Figure 5 using a variety of factors. At first, the size of the rule base is quite small; we succeeded in extracting the minimal amount of 12 rules that successfully describe 6728 instances. Furthermore, the length of each rule is also quite lower than expected. Notably, the rule with the maximum length, rule 5, has only 7 antecedents, so it includes only half of the metrics defined in Table I. In addition, 9 out of 12 rules, rules 1-3, 6, and 8-12, have at most 3 antecedents. Finally, the average rule length is lower than 3 (the exact value is 2.75).

Although useful conclusions can be drawn by simply examining this rule base, we also visualize the rules in order to better illustrate how our system can provide a comprehensive reliability analysis. We visualize each rule using blocks when a variable belongs to a set. Figure 6a visualizes the 7th rule, allowing us to draw conclusions. For example, we could say that components are defect-prone when they change consecutively (`NumConsecChanges IS High`), or when there is a fair amount of lines modified (`ChurnTotal IS Average`).

Although reading and visualizing these rules one by one is acceptable due to their small size and length, we would like to go one step further by visualizing the whole rule base. This can be done in a variety of ways [23]. The visualization method to be used depends on several factors, such as the number of antecedents per rule, the number of fuzzy sets, etc. In our case, that the rule base is quite small, using an overly complex method to visualize it would be an overkill. As a result, we visualize the rule base in Figure 6b using transparent blocks.

In Figure 6b, each rule is a layer of the diagram. Given that each layer has transparency equal to 0.33 (which is enough since no more than 3 rules have a common antecedent), we can start from the first rule and plot it on the diagram, and then plot the second one, etc. until we plot the final (twelfth) rule. Darker areas represent high correlations between the metric values and the defect-proneness of the component, whereas lighter ones denote smaller effect on its defect-proneness.

Given this diagram, one can conclude that a component is defect-prone when it changes often (`NumberOfChanges IS Average OR High`), especially if the changes are consecutive (`NumConsecChanges IS High`), or when few people work on the component (`PeopleTotal IS Low`) but most of them during bursts (`TotalPeopleInBurst IS Average OR High`). Other metrics may have little or no effect to defect-proneness. Thus, a component may or may not be defect-prone, regardless of when the last burst occurred (`TimeLastBurst`), while the time of the first burst does not strongly affect the consequent; it is somewhat possible if it is late (`TimeFirstBurst IS High`), but this block is light.

VII. Conclusion

Although several research efforts on SRP are effective, they usually lack interpretability. In this paper, we designed a novel fuzzy rule-based system which is equally effective to standard classifiers, yet also yields comprehensive results. The reduction of the problem to one-class classification and the use of a rule-length criterion to the GA fitness function resulted in small comprehensive rules that can be visualized in an intuitive manner. In addition, the proper design of the population of the GA and the crossover and mutation operators ensured that the algorithm achieves satisfactory results.

Despite evaluating our method on a single dataset, the Eclipse IDE is a complex software project indicating that our GFS may also be applicable to other projects, which is a fair assumption to be explored further in future work. Furthermore, even if we do not claim that the burst metrics are effective on all datasets, our methodology is metric-agnostic, i.e. it can be adapted to other types of metrics. Concerning performance, our GFS is stable since the resulting rule bases from multiple runs are very similar (omitted due to space limitations). Finally, since effectiveness is not our main hypothesis, we refrain from discussing the statistical significance of our approach compared to LS or comparing the methods in a cross-validation scheme, and focus on the interpretability of the rule base. In any case, we argue that the

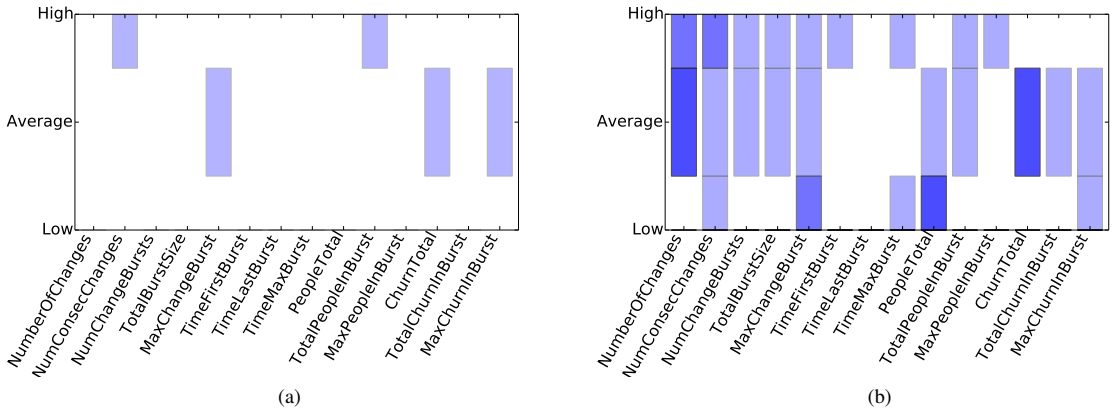


Fig. 6. Visualization of fuzzy rules using semi-transparent blocks. Diagram (a) depicts the 7th rule of the rule base, and diagram (b) depicts the rule base. Since each rule is a layer, the transparency degree of an antecedent of diagram (b) indicates the support of this antecedent in the rule base.

outcome of an SRP method should not be a classification, but rather a *reliability analysis*, with a comprehensive set of rules.

Our approach can provide several ideas for future research. At first, concerning the fuzzy rules, one can evaluate different configurations including two-class and one-class rules in order to effectively solve the problem, while the effect of GAs can also be evaluated with regard to other differently derived fuzzy systems. In addition, interesting comparisons can be made by evaluating rule bases derived using the Michigan and Pittsburgh approaches. Finally, the interpretability versus accuracy trade-off of our GFS can be explored, based on different rule lengths and rule base sizes.

ACKNOWLEDGMENT

Parts of this work have been supported by the FP7 Collaborative Project S-CASE (Grant Agreement No 610717), funded by the European Commission.

REFERENCES

[1] Marco D'Ambros, Michele Lanza, and Romain Robbes. Evaluating Defect Prediction Approaches: A Benchmark and an Extensive Comparison. *Empirical Software Engineering*, 17(4–5):531–577, August 2012.

[2] S. R. Chidamber and C. F. Kemerer. A Metrics Suite for Object Oriented Design. *IEEE Transactions on Software Engineering*, 20(6):476–493, June 1994.

[3] F. Brito e Abreu and W. Melo. Evaluating the Impact of Object-Oriented Design on Software Quality. In *Proceedings of the 3rd International Software Metrics Symposium*, pages 90–99, Mar 1996.

[4] Raimund Moser, Witold Pedrycz, and Giancarlo Succi. A Comparative Analysis of the Efficiency of Change Metrics and Static Code Attributes for Defect Prediction. In *Proceedings of the 30th International Conference on Software Engineering*, pages 181–190, New York, USA, 2008.

[5] A.E. Hassan. Predicting Faults Using the Complexity of Code Changes. In *Proceedings of the IEEE 31st International Conference on Software Engineering*, pages 78–88, May 2009.

[6] N. Nagappan, A. Zeller, T. Zimmermann, K. Herzig, and B. Murphy. Change Bursts as Defect Predictors. In *Proceedings of the 2010 IEEE 21st International Symposium on Software Reliability Engineering (ISSRE)*, pages 309–318, Nov 2010.

[7] E. H. Mamdani and S. Assilian. An Experiment in Linguistic Synthesis with a Fuzzy Logic Controller. *International Journal of Man-Machine Studies*, 7(1):1–13, 1975.

[8] T. Takagi and M. Sugeno. Fuzzy Identification of Systems and Its Applications to Modeling and Control. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-15(1):116–132, Jan 1985.

[9] L.-X. Wang and J.M. Mendel. Generating Fuzzy Rules by Learning from Examples. *IEEE Transactions on Systems, Man and Cybernetics*, 22(6):1414–1427, Nov 1992.

[10] John H. Holland and Judith S. Reitman. Cognitive Systems Based on Adaptive Algorithms. In *Pattern directed inference systems*, pages 313–329. Academic Press, New York, USA, 1978.

[11] Stephen Frederick Smith. *A Learning System Based on Genetic Adaptive Algorithms*. PhD thesis, University of Pittsburgh, PA, USA, 1980.

[12] Gilles Venturini. SIA: A Supervised Inductive Algorithm with Genetic Search for Learning Attributes Based Concepts. In *Proceedings of the European Conference on Machine Learning*, pages 280–296, London, UK, 1993. Springer-Verlag.

[13] X. Yang, K. Tang, and X. Yao. A learning-to-rank approach to software defect prediction. *Reliability, IEEE Transactions on*, PP(99):1–13, 2014.

[14] Thomas Zimmermann, Rahul Premraj, and Andreas Zeller. Predicting Defects for Eclipse. In *Proceedings of the Third International Workshop on Predictor Models in Software Engineering*, pages 9–19, Washington, DC, USA, 2007. IEEE Computer Society.

[15] Tim Menzies, Jeremy Greenwald, and Art Frank. Data Mining Static Code Attributes to Learn Defect Predictors. *IEEE Transactions on Software Engineering*, 33(1):2–13, January 2007.

[16] Patrick Knab, Martin Pinzger, and Abraham Bernstein. Predicting Defect Densities in Source Code Files with Decision Tree Learners. In *Proceedings of the 2006 International Workshop on Mining Software Repositories*, pages 119–125, New York, NY, USA, 2006. ACM.

[17] Sunint K. Khalsa. A Fuzzified Approach for the Prediction of Fault Proneness and Defect Density. In *Proceedings of the World Congress on Engineering*, volume 1, pages 218–223, 2009.

[18] S. Aljahdali and A.F. Sheta. Predicting the Reliability of Software Systems Using Fuzzy Logic. In *Proceedings of the Eighth International Conference on Information Technology*, pages 36–40, April 2011.

[19] Ajeet Kumar Pandey and Neeraj Kumar Goyal. Fault Prediction Model by Fuzzy Profile Development of Reliability Relevant Software Metrics. *International Journal of Computer Applications*, 11(6):34–41, December 2010.

[20] Q.P. Hu, M. Xie, S.H. Ng, and G. Levitin. Robust Recurrent Neural Network Modeling for Software Fault Detection and Correction Prediction. *Reliability Engineering & System Safety*, 92(3):332–340, 2007.

[21] S.H. Aljahdali and M.E. El-Telbany. Software reliability prediction using multi-objective genetic algorithm. In *Proceedings of the 2009 IEEE/ACS International Conference on Computer Systems and Applications (AICCSA)*, pages 293–300, 2009.

[22] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. DEAP: Evolutionary Algorithms Made Easy. *Journal of Machine Learning Research*, 13:2171–2175, Jul 2012.

[23] Binh Pham and Ross Brown. Analysis of Visualisation Requirements for Fuzzy Systems. In *Proceedings of the 1st International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia*, pages 181–187, New York, NY, USA, 2003. ACM.