World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:4, No:3, 2010

Artificial Intelligence for Software Quality Improvement

Martín Agüero, Franco Madou, Gabriela Esperón, Daniela López De Luise

***Abstract*—**This paper presents a software quality support tool, a Java source code evaluator and a code profiler based on computational intelligence techniques. It is Java prototype software developed by AI Group [1] from the Research Laboratories at Universidad de Palermo: an Intelligent Java Analyzer (in Spanish: Analizador Java Inteligente, AJI). It represents a new approach to evaluate and identify inaccurate source code usage and transitively, the software product itself.

The aim of this project is to provide the software development industry with a new tool to increase software quality by extending the value of source code metrics through computational intelligence.

***Keywords*—**Software metrics, artificial intelligence, neural networks, clustering algorithms, expert systems

I. Introduction

IN order to show that a computer program is mature and free of bugs, and that Software Requirements Specifications have been met (SRS), it will be necessary to have a strategy to support this process. The goal for any software project is to accomplish the above mentioned requirements, which means to get the best quality. Historically, the word "quality" has been adapted and has evolved together with the different technologies to which it has been applied. In the thirties, the metallurgical industry defined quality as a compliance to requirements; any deviation from such requirements meant loss of quality or limited trust in product quality. The consequence of this was lower costs and less rework [2]. In the fifties, quality costs increased exponentially. Therefore, specifications including tolerance (i.e., a deviation from perfection) were proposed. Inspections ensured that the product fell within a predefined tolerance. The goal of such inspections was to avoid corrections through the identification of product deviations from the original specification [3]. The creation of software does not imply serial production costs, but it is an intensive activity [4]. It requires the interaction and coordination of several specialists during all development stages. In the following subsections, different perspectives of software quality are presented.

Martín Agüero is with the Universidad de Palermo, Argentina.
e-mail: aguero.martin@gmail.com

A. Software Quality

It can be said that, as an adaptation and extension of classical definitions, the software industry focuses on the following principles: 1. Software requirements are the quality metric fundamental. Lack of compliance with requirements is a quality failure. 2. Standards establish development criteria. Absence of standards means, in many cases, low quality [5]. 3. Indirect measures (e.g. usability, maintainability, etc.) and direct measures (e.g. lines of code). Software Quality Assurance (SQA) are a way of encompassing the software engineering processes. It mainly consists of monitoring and developing information and administration tasks [6]. Inspection and metrics make software projects successful due to their excellent quality control results. Even though intensive software quality control increases costs, it is an activity with high Return On Investment (ROI). Empiric verification without data indicators and measures make theories and propositions remain abstract [7].

B. Strategies for Software Quality Assurance

Estimation Metrics: Emerged as a reaction to omissions and deficiencies in the Lines of Code (LOC), an estimation technique used at the beginning of the eighties. Albrecht presented through IEEE a concept called Function Points (FP), showing that it was not technically reliable to measure LOC. The following are some useful approaches that were devised afterward to improve QA.

OO Languages: The development of languages in object paradigms reduced the bug levels in procedural languages [8]. But software always have defects such as: 1. A very ambitious scheduling. 2. Complex models. 3. Unbalanced module sizes. 4. Subtle code errors even when testing is over.

Strategic Methodologies: Total Quality Management (TQM) success is only possible when there is a strong management commitment. It depends on the application of effective quality programs and technical revisions. They must be implemented before the application of Total Quality Management to the business model. Companies that do not apply quality metrics and decide to obtain a marginal profit increase by using TQM as a slogan will rarely achieve a successful outcome [8].

Test Case Tools: Test case tools are popular. They are able to identify and isolate missed and failed tracks of code. Nevertheless, a level of program execution of 90% does not mean a 90% bug–free program. It has been stated that less than 30% of defects are found by means of unit testing. Furthermore, test cases do not guarantee correctness since

there could be mistakes in the coding process. It has been proved that: 1. Test cases have more bugs than the products for which they have been created. 2. Typically, one third of test cases are duplicated. This increases costs and therefore, does not optimize the use of resources [8].

C. Economic impact of inadequate infrastructure for software testing [9]

Today the complexity of software is increasing at an alarming rate. Quality is defined as a bundle of attributes and the level of those attributes holds a positive value. Few companies are interested in advanced testing techniques as a way of forecasting field reliability based on test data and of calculating defect density to benchmark quality. Standardized automated testing scripts along with standard metrics would also provide a more consistent method to determine when to stop testing. Most developers prefer early bug detection, at the same developmental stage. Based on the software community and user surveys, the US annual costs of an inadequate infrastructure for software testing is estimated to range from $22.2 to $59.5 billion[1]. Over half of these costs are borne by software users in the form of error avoidance and mitigation activities. The remaining costs are borne by software developers and reflect the additional test resources that are consumed due to inadequate testing tools and methods.

D. Intelligent Java Analyzer (IJA)

This new software tool prototype employs traditional and new source code metrics to model its content in context. Metrics results are discretized depending on deviations from the programming language specification. Thresholds are set to obtain distances from the preferred code style. A dataset built by Expectation Maximization (EM) data mining algorithm is the reference data source to train a neural network. A Multi Layer Perceptron (MLP) artificial neural network (NN) classifies the source code instances on clusters formerly established by the training set. In a programmed self-tuning process, the prototype can adjust each cluster profile, determining a dynamic distinctive identity for every classification output (NN). The classification phase groups source code instances that share common attributes of their syntaxes. In those cases where the attribute reveals a sign of erroneous language handling, a recommendation phase is activated. An expert system pre-loaded with rules analyzes the classification results and identifies every inaccurate source code usage. The rule engine also builds a set of recommendations based on key features detected in the code. The analysis process is completed with a report-style output advising the author on convenient procedures to improve the source code.

II. Prototype Architecture

IJA prototype is divided into 9 modules: 6 for core functions and 3 for support services Fig. 1 shows component relations, storage units and external interfaces. The analysis is executed in two main steps: classification and recommendation, both of which are coordinated by a Services Manager module.

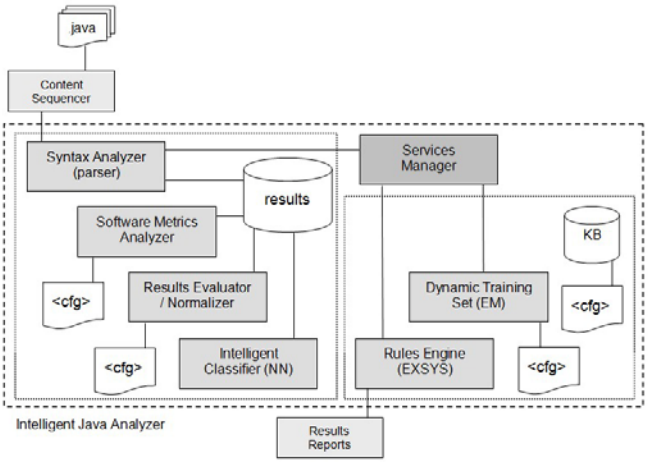

Fig. 1 Intelligent Java Analyzer (IJA) architecture

It is a flexible and adaptable tool due to its design and XML configuration files. It could be used not only for scientific requirements but also to suit the needs of any firm. The whole tool is built on Enterprise Java technology platform; it runs on any JEE 1.5 servlet container. Its design maximizes CPU load and reduces memory requirements, enabling the quick analysis of large datasets. The following sections describe each module design and their main features.

A. Content Sequencer

Content Sequencer Module is an IJA extension of Java Collection API interface (Application Programming Interface). It standardizes, encapsulates, and serializes source code files (SCF) content synchronically in order to process it in a transparent way. It can be configured to define specific word separator tokens.

B. Syntax Analyzer (Parser)

The first component of the classification phase is the Syntax Analyzer. It processes and extracts some context features. It also implements a parser algorithm that translates Java syntax into the model proposed by W. A. Woods: Augmented Transition Networks (ATN) [10].This applies State and Memento design patterns [11]. The algorithm detects some reserved words, symbols and structures and changes to a specific state.

The IJA modular design makes it possible to extend all its functionalities to other languages just by selecting a specific Syntax Analyzer module implementation for each programming language.

C. Software Metrics Analyzer

It is the calculus component and the analyst's chosen algebra operations translator. According to the actual XML configuration, it generates software metrics by executing

[1]The impact estimates do not reflect the "costs" associated to mission critical software where any problem may lead to extremely high costs such as loss of life or catastrophic failure.

mathematical operations and using the results obtained from the previous module.

For example, a metric named v_3 is the division between the quantity of methods and the quantity of methods with names starting with lowercase[2]. Besides some new metrics, the software also implements classical metrics, e.g. v_{11} assesses the rate between the number of comments collected in a SCF and the number of Javadocs in it.

At this point, the analyst can set a weighting value to every system operator. The set of metrics defined is important to model SCF quality [13].

D. Results Evaluator / Normalizer

The Results Evaluator module normalizes the specific numeric values for each metric between -1 and 1. There are predefined thresholds —or quality class bounds— to help evaluate the result quality.

Each SCF is classified using the metric numeric results and the thresholds. Floating labels provide a code identification reference (ID) of the SCF accumulated in each sequence. Finally, the data is exported into an *arff* format so that the results (attribute-relation file format) are compatible with the Intelligent Classifier Module.

E. Intelligent Classifier (Neural Network)

This is the last step into the classification phase analysis process. It has a Multi Layer Perceptron (MLP). This Neural Network with backpropagation, was trained using the clusters derived automatically by an Expectation Maximization clustering algorithm (EM) [14].

Fig. 2 represents the MLP architecture. The input layer is loaded with metric values (V_1, V_2, V_3…V_{14}) for each SCF. There is just one hidden layer and an output layer defined as each of the five (5) EM clusters.

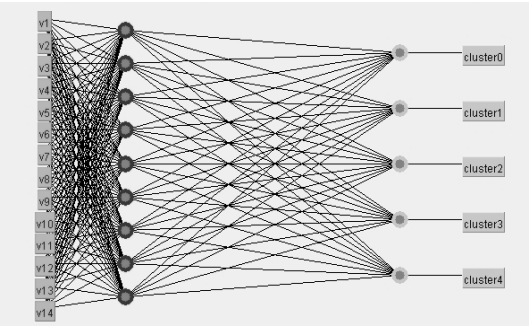

Fig. 2 IJA Neural Network

Data Mining[3]*:* In order to find out the best cluster number for the MLP, the Expectation Maximization (EM) algorithm was applied [15]. Parameters were established so as to detect clusters automatically by using cross validation. Results with log likelihood (-1.79183) are shown in Table 1.

[2]A violation to the Java Language Specification [12]
[3]Exploratory analysis to extract hidden information from large datasets.

TABLE 1
EM CLUSTERS

cluster #	*instances*	*%*
0	9	1
1	127	13
2	102	10
3	149	15
4	572	57
5	4	4

The EM algorithm stops when there is not a significant quality increase. The quality is measured with:

$$r_1.P(a)+r_2.P(b)+r_3.P(c)+r_4.P(d)+r_5.P(e) \qquad (1)$$

Being a, b, c, d and e clusters and r_1, r_2, r_3, r_4 and r_5 the parameters, the algorithm uses the probability register of true parameters. Log likelihood stands for the willingness or credibility measure of these probabilities. It is obtained as the product of the conditional probabilities upon every instance *i* in the sample:

$$r_1.P\left(\frac{x_i}{a}\right)+r_2.P\left(\frac{x_i}{b}\right)+r_3.P\left(\frac{x_i}{c}\right)+r_4.P\left(\frac{x_i}{d}\right)+r_5.P\left(\frac{x_i}{e}\right) \qquad (2)$$

As anoter test, EM algorithm was reconfigured with the following parameters:
Maximum interactions: 100
Deviation from minimum standard: 1.0×10^{-6}
Number of clusters: 5
Seeds: 200
The final results are shown in Table for log likelihood: -9.06605.

TABLE II
CLUSTERS

cluster #	*instances*	*%*
0	326	33
1	50	5
2	190	19
3	246	25
4	188	19

A. Neural Network Setup: A multilayer perceptron neural network was selected to classify the SCF. It was trained by a backpropagation algorithm and configured with the following parameters:
Learning Rate: 0.3
Momentum: 0.2
Training Time: 500 epochs
Validation threshold: 20
Training Source: set training and cross validation

B. Neural Network Classification Accuracy: A general average, over 90%, confirms the strong correlation between SCF grouped in clusters by EM and the NN capability of distinguishing differences between attributes and of classifying them correctly [18].

C. Cluster Profiling: An automated algorithm to evaluate the meaning of each cluster was implemented. It depends on the instance SCF metric values. In this process each metric value of the cluster is compared to the total average (as a reference), and depending on a distance factor, a level of proximity is determined. This algorithm takes outputs from clustering and gets results from the following procedure:

1. Inputs: the clusters characterized by the metrics: $V_1 ... V_n$ (discrete values [-1,0,1])

The algorithm:

a. Acquire an average for every metric and value, and select it as a reference (see Table).
b. Select the value V_x with minimum distance to reference and categorize it comparing it with the same V_x of the other clusters (using relative values) (see Table).
c. Set up a position (ranking) for the cluster, depending on the distance results (see Table).

2. Output clusters characterized by metrics. Using discrete numbers and classifications according to the proximity to reference (see Table).

Demonstration for V_1 (lines of code / number of classes)

TABLE III
REFERENCE AVERAGE VALUES

metric	*value*	*result*	*meaning*
v1	-1	0.250503	*A 25% of cluster classes have less than 100 LOC/class.*
v1	0	0.26760563	*Between 100 and 200 LOC/class.*
v1	1	0.48189133	*More than 200 LOC/clases*

TABLE IV
DISTANCE TO REFERENCE[4]

cluster #	*metric*	*value*	*result*	*distance*
0	v1	0	0.4326241	0.16501847
1	v1	-1	0.6369863	0.3864833
2	v1	0	0.4871795	0.21957386
3	v1	1	0.6551094 7	0.17321813
4	v1	0	0.4691358	0.20153016

TABLE V
POSITIONS DEPENDING ON DISTANCE RESULTS

cluster #	*metric*	*value*	*distance*	*position*
0	v1	0	0.16501847	0
1	v1	-1	0.3864833	4
2	v1	0	0.21957386	3
3	v1	1	0.17321813	1
4	v1	0	0.20153016	2

Then, every numeric result with a high-grade position (0 or 1) is converted into a qualitative label where the profile that determines the identity of every cluster is obtained, for example the Cluster 0 profile:

[4]In all cases the value series with the closer result to the reference are selected.

TABLE IV
CLUSTER 0 PROFILE

metric	*value*	*position*	*meaning*
v1	0	0	*Between 100 and 200 LOC each class.*
v2	1	1	*Methods with more than 40 LOC on each block.*
v4	1	1	*Uppercased variable names.*
v6	1	1	*High usage of constant variables.*
v10	1	0	*High usage of inner and nested classes.*
v11	1	1	*Disproportionate usage of comments type.*
v22	1	1	*Most classes having just the default constructor.*

The next section describes recommendation phase modules.

III. EXPERT SYSTEM

An expert system or a system based on knowledge is a computer system that makes decisions or solves problems in a particular field by means of knowledge and analytical rules defined by experts. It is made up of a knowledge base —the rules of the EXSYS, that is to say, the codified expert knowledge—, a working memory —stocks the data received at the beginning in order to solve a problem, then the intermediate conclusions and the final results— and an inference engine, which models the human reasoning process. The diagram in Fig. 3 basically represents the structure of an expert system. Three examples of very well-known expert systems are CLIPS [19], JESS [20] and DROOLS [21].

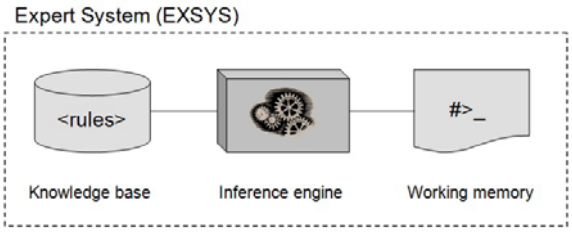

Fig. 3 Basic architecture of an expert system

A. Clips

In the mid-eighties, NASA[5] required the support of expert systems for developing projects. Therefore, a number of prototypes emerge but their results are not good enough to fulfill internal requirements. Then, a prototype of an expert system was developed; it called CLIPS (C Language Integrated Production System) whose main characteristic is its interoperability with other existing systems. Subsequent improvements and enlargements have turned CLIPS into a point of reference for the development of other expert systems. Even though CLIPS has shown successfully its productive capacity, as regards expert systems, and it is now in the public domain, its interface with Java through JNI (Java Native Interface) is going through a 0.2 beta experimental phase.

[5] National Aeronautics and Space Administration.

B. Jess

The rule engine JESS is a project that had its origin in CLIPS but which was written entirely in Java. It was developed during the nineties in Sandia National Laboratories and it shares with CLIPS several design concepts and similarities regarding syntax.

C. Drools

As in the case of CLIPS and JESS, DROOLS is the implementation and extension of Rete algorithm [16], designed by Dr. Charles L. Forgy at the Carnegie Mellon University. Basically, its algorithm consists in a network of interconnected nodes with different characteristics — according to rules that define them— that evaluate inputs by propagating results to the next node when there are coincidences (see Fig. 4).

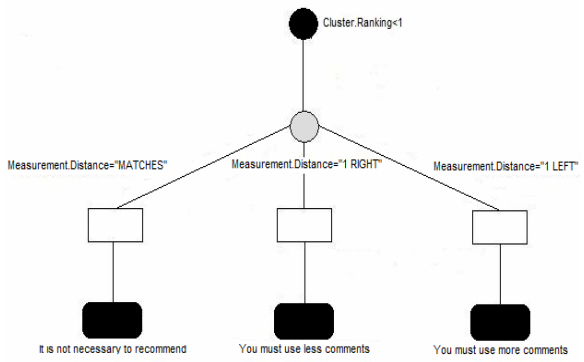

Fig. 4 Rete algorithm

DROOLS offers integration tools with Java, capacity of scalability and a clear division between data and logic domain.

The IJA project incorporates DROOLS Expert and defines rules in MVEL scripting language.

Considering the information obtained through metrics and indicators, IJA uses an expert system which, according to the results of previous processes, proposes recommendations for the correction of any deficiencies found. The solution is generated by an expert system with pre-loaded rules in the knowledge base. Basically, each rule analyzes the classification created by a neuronal network and then makes a recommendation based on the cut values for each metric [17] (see Table).

TABLE VII
RECOMMENDATIONS AS METRICS AND CLASSIFICATION

metrics	*meaning*	*intervals*	*recommendations*
v1	Lines per class	0-100	*No suggestion.*
		101-200	*It is suggested to decrement the number of LOC per class.*
		201 or more	*It is necessary to decrement the number of LOC per class.*
v2	Methods per class	0-20	*No suggestion.*
		21-40	*It is suggested to decrement the number of methods per class.*
		40 or more	*It is necessary to decrement the number of methods per class.*
v22	Classes with default constructor	0	*No suggestion.*
		1	*It is suggested to write constructors.*
		More than 1	*It is recommended to write the constructor for each class.*

Knowledge Base: The knowledge base that is part of the IJA expert system establishes a bijective function between metrics and rules, that is to say, each metric that was evaluated has a specific rule associated to it. Every rule analyzes every classification result and metric value according to the following algorithm:

a. For the cluster where the SCF was classified, is that metric significant (ranking > 1)?
b. Does the value obtained agree with the one that was expected?

YES: No recommendation is necessary
NO: Suggest a correction

The text for that suggestion is also different, considering the distance with respect to the expected value [22]. Significant deviations mean more relevant recommendations.

IV. Future Work

The goal of the next stage of research is the intensive testing and subsequent tuning of the prototype by means of a statistical analysis that may validate the system or help make the required adjustments.

In the same manner, more research will be done, taking into account the theory-practice framework that the results of the present work represent, in order to define new quality criteria for the evaluation of software.

V. Conclusion

This paper proposes a new approach to evaluate source code quality and source code profiling. In order to do so, it defines source code metrics and analyzes its internal behavior with data mining and machine learning procedures. In the process, a dataset created by a data mining algorithm is the reference classified data source to set up a neural network. The trained multilayer perceptron has shown excellent precision to classify this sort of data. An expert system analyzes the classification results and identifies every inaccurate source code usage.

In order to test the prototype, a web user interface is being developed. Therefore, this feature will make community

feedback available, enabling future self-tuning capabilities.

This tool represents a new approach to automatically evaluate and provide recommendations for programmers in order to improve the source code quality, and consequently, the software product itself.

REFERENCES

[1] AI Group: http://www.palermo.edu/ingenieria/it_lab.html
[2] Roe and Lytle, pp. 99, 1935.
[3] Moore, pp. 652, 1958.
[4] James D. Arthur, "Managing Software Quality: A Measurement Framework for Assessments and Prediction", Springer, 2002.
[5] ISO/IEC 9126: http://www.cse.dcu.ie/essiscope/sm2/9126ref.html
[6] Roger S. Pressman, "Ingeniería del Software: Un Enfoque Práctico", Mc Graw Hill, 1998.
[7] Stephen H. Kan, "Metrics and Models in Software Quality Engineering", Addison-Wesley Professional, 2002.
[8] Capers Jones, "Applied software measurement: assuring productivity and quality", Mc Graw Hill, 1996.
[9] National Institute of Standards and Technology, "The Economic Impacts of Inadequate Infrastructure for Software Testing", RTI, 2002.
[10] W.A. Woods, "Transition Network Grammars for Natural Language Analysis", pp. 591-606, Communications of the ACM, 1970.
[11] Bruce Eckel, "Thinking in Patterns", 2003.
[12] James Gosling, Bill Joy, Guy Steele, Gilad Bracha "The Java Language Specification 3rd Edition ", Pretience Hall, 2005.
[13] Daniela López De Luise, Martín Agüero, "Aplicación de Métricas Categóricas en Sistemas con Lógica Difusa", Revista IEEE América Latina, 2007.
[14] Patrick H. Winston, "Inteligencia Artificial, tercera edición", Addison Wesley Iberoamericana, 1992.
[15] Ian H. Witten, Eibe Frank "Data Mining: Practical Machine Learning Tools and Techniques", pp. 265, Morgan Kaufmann, 2005.
[16] Charles Forgy, "Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem", Artificial Intelligence, 19, pp 17-37, 1982.
[17] Madou F, Agüero M., Esperón G., López De Luise D., "Sistemas Expertos en Evaluación de Calidad Java", CONESCAPAN, 2009.
[18] Agüero M., Esperón G., Madou F, López De Luise D., "Intelligent Java Analyzer", IEEE CERMA, 2008.
[19] CLIPS http://clipsrules.sourceforge.net/
[20] JESS http://www.jessrules.com
[21] DROOLS http://www.jboss.org/drools/drools-expert.html
[22] Madou F, Agüero M., Esperón G., López De Luise D., "Evaluador Inteligente de Código Java", CICA, 2009.

Martín Agüero: Argentinian. Graduate in Informatics (Universidad de Palermo). Sun Certified Java Programmer (SCJP). Consultant, specialized in Java Enterprise Edition (JEE). Researcher from AIGroup at the Engineering School since 2005, for the IJA project.

Gabriela Esperón: Argentinian. Mathematics professor. Degree at Universidad de Buenos Aires. Specialized in numerical analysis. Staff of teachers from the Department of Exact Sciences, Engineering School, Universidad de Palermo. Researcher and coordinator of AIGroup since 2007.

Franco Madou: Argentinian. Advanced Informatics student from Universidad de Palermo. Consultant, specialized in Java Enterprise Edition (JEE). Researcher from AIGroup at the Engineering School since 2007, for the IJA project.

M. Daniela López De Luise: Argentinian. Doctoral degree in Computing Sciences from Universidad nacional de La Plata. Senior analyst for many enterprises. Researcher on applied computational intelligence in linguistics. Senior member of IEEE and founder of the Computational Intelligence Society in Argentina. Director of AI Group at Universidad de Palermo.