Using software metrics to estimate the impact of maintenance in the performance of embedded software

Andrws Vieira, Pedro Faustini, Érika Cota
PPGC - Informatics Institute - Federal University of Rio Grande do Sul (UFRGS)
Porto Alegre, Brazil
{andrwsvieira, phafaustini, erika}@inf.ufrgs.br

***Abstract*—This paper proposes a strategy to assist the designer in evaluating the impact of a design choice with respect to the non-functional requirements in embedded systems. We use several regression models to predict physical metrics from design metrics in order to estimate the impact on performance of software changes in the early development stages. This prediction can be used during maintenance to compare alternative design changes before implementation. Such an early estimation allows an efficient design space exploration with no penalty in the development time, which are crucial aspects for an embedded system.**

Keywords— Software Metrics; Maintenance; Data Mining; Regression Analysis; Embedded Systems.

I. INTRODUCTION

In the embedded systems domain, an important challenge is to meet non-functional requirements such as execution time and memory and energy consumption in a constrained hardware. It is common for an embedded system designer to evaluate different solutions (not only algorithms, but also data structures, module decomposition, distinct libraries, etc.) to optimize the performance of the system in a given hardware platform.

The complexity of embedded systems grows constantly, making the use of more abstract design strategies a necessity also in this domain. However, achieving tight performance constraints while taking the benefits of high-level design and programming tools is a challenge since the more abstract is the design the less predictable and more costly is the run-time behavior [1]. The evaluation of the software performance in the target platform is a time-consuming activity. Thus, another challenge in the embedded domain is to perform a thorough design space exploration within a stringent time-to-market.

Many core embedded functions such as GPS or image processing functions can be designed with modeling frameworks such as Matlab which generate dedicated low-level code for a given platform. On the other hand, user interfaces and other functionalities that support the core embedded functions have the same design challenges of non-embedded software, although they must also execute in a constrained hardware. For those components, different module decompositions and interactions lead to distinct performance levels in terms of both internal quality metrics and also hardware resource consumption.

However, a less experienced software designer may not know or recognize the solution that leads to the best performance for a given hardware and spend too much time completing the evaluation cycle in the target platform. This is especially serious during software maintenance, when the target platform is already fixed and design time is even shorter.

The main goal of our research is to define a strategy to support a less experienced embedded software developer in designing a higher quality system. The strategy is based on providing feedback, as soon as possible and before system deployment, about design decisions and how they may impact the quality of the final system in the target platform. This paper presents the first part of such a supporting framework and provides feedback about the correlation between design decisions and non-functional requirements in terms of physical metrics. This feedback can be used mainly during maintenance to evaluate how a design change affects the non-functional requirements (e.g. energy consumption) in the hardware platform.

In this paper, we analyze software metrics extracted from class diagrams (design metrics) and code execution (dynamic metrics or hardware metrics) and apply techniques of Knowledge Discovery in Database (KDD) to implement a prediction method that receives as input a set of design metrics and try to estimate the hardware metrics. The proposed technique is evaluated with 21 different implementations of a banking system representing actual distinct implementations of the same specification made by less experienced designers. Results show that the proposed technique can correlate design and hardware metrics and assist the designer with simple and actionable information.

The paper is organized as follows: Section II discusses some related works. In Section III, we briefly describe the background information about software metrics and regression analysis. Section IV presents our approach based on regression analysis. Section V discusses experimental results and Section VI concludes the paper.

II. RELATED WORKS

A great deal of effort has been spent in the last few years to create useful and practical software metrics [2] [3] [4] [5] [6] [7] as well as cost-effective measurement tools. Comparatively few works have studied the relationship between metrics [8] [9] mainly extracted at different design levels. This might be related to the difficult to establish such a correlation, since metrics defined at different abstraction levels may actually be measuring different things. Still, it is intuitive that good (or bad) design metrics indicate that metrics extracted from the next design level will present similar good (or bad) evaluation.

1063-6773/14 $31.00 © 2014 IEEE
DOI 10.1109/ICSME.2014.86

A major issue with software metrics though is that they must be used with care [10] [11]. This is due to several reasons, such as the imprecision of the extraction method, the definition of the metric itself, the intrinsic characteristics of the artifact used or even the context of the design process and organization developing the system. Even though a large number of metrics exists, use them to extract strong conclusions about the software is indeed a non-trivial task [12]. Hence, methods capable of extracting knowledge from a set of metrics are still needed.

A few works have tried to establish a relationship between code metrics and physical performance of embedded systems. They try to characterize the cost of object-orientation in the embedded platform [1] [13] or modify an OO code to make it more efficient in a single platform [4]. Previous work has shown that OO code impacts the embedded system performance, but, to the best of our knowledge, one still has to define the exact correlation between code and (mainly) design decisions and physical metrics.

III. BACKGROUND

In this section, we review the concepts used in this work about software metrics, and regression analysis.

A. Software Metrics

The great importance of software metrics extraction is in fact to provide a quantitative view of software and its development [14]. One classification widely adopted nowadays, divides software metrics into three types: process metrics, project metrics and product metrics. In this work, we are interested in the product metrics, that is, metrics that assess internal quality attributes of the software product.

Product metrics describe the attributes of the software product at any phase of its development. Product metrics may measure the size of the program, complexity of the software design, performance, portability, maintainability, and product scale. They are used to assess the quality of the product, measure the medium or the final product, among others features. Product metrics can be further divided into two categories: *Dynamic Metrics* and *Static Metrics*.

In this work, we are interested in static metrics at design level (class diagram), to allow us to estimate the dynamic metrics (execution and physical behavior) in embedded systems when they undergo maintenance. The set of metrics used in this work is detailed below.

1) Class Diagrams Metrics

The metrics of class diagrams were extracted by SDMetrics tool [15]. The description and category of all metrics extracted are shown in Table I.

TABLE I. LIST OF CLASS DIAGRAM METRICS

Metric	Category	Description
NumAttr	Size	The number of attributes in the class.
NumOps	Size	The number of operations in a class. Also known as Number of Methods (NM) [5].
NumPub Ops	Size	The number of public operations in a class. Aka Number of Public Methods (NPM) [5].
Setters	Size	The number of operations with a name starting with 'set'.
Getters	Size	The number of operations with a name starting with 'get', 'is', or 'has'.
Nesting	Inheritance	The nesting level of the class (for inner classes). Classes not defined in the context of another class have nesting level 0. [10]
IFImpl	Inheritance	The number of interfaces the class implements.
NOC	Inheritance	The number of children of the class [6].
Num Desc	Inheritance	The number of descendants of the class. Counts the number of children of the class, their children, and so on [8].
DIT	Inheritance	The depth of the class in the inheritance hierarchy [6].
CLD	Inheritance	Class to leaf depth. The longest path from the class to a leaf node in the inheritance hierarchy below the class [11].
OpsInh	Inheritance	The number of inherited operations. Also known as Number of Methods Inherited (NMI) [7].
AttrInh	Inheritance	The number of inherited attributes. This is calculated as the sum of metric NumAttr taken over all ancestor classes of the class.
Dep_Out	Coupling	The number of elements on which this class depends. This metric counts outgoing plain UML dependencies and usage dependencies.
Dep_In	Coupling	The number of elements that depend on this class. This metric counts incoming plain UML dependencies and usage dependencies.
NumAssEl _ssc	Coupling	The number of associated elements in the same scope (namespace) as the class.
NumAssEl _sb	Coupling	The number of associated elements in the same scope branch as the class.
NumAssEl _nsb	Coupling	The number of associated elements not in the same scope branch as the class.
EC_Par	Coupling	The number of times the class is externally used as parameter type. This is the number of parameters defined outside this class, that have this class as their type [9].
IC_Par	Coupling	The number of parameters in the class having another class or interface as their type [9].

2) Dynamic (Hardware) Metrics

Performance in embedded systems is measured in terms of hardware resource consumption: memory consumption, CPU usage and communication between CPU and memory. Memory consumption is divided in the use of the main memory (RAM), and caches. Cache memories accelerate the access to a given data or code, but consume more energy whereas RAM memory consumes less energy but requires more execution time. Caches can be divided in two levels, L1 and L2, according to its size and proximity to the CPU. When the information is not found in the closest cache we have a *cache miss* and the information must be sought in the next level cache or main memory. This process increases communication ratio, execution time, and energy consumption. Thus, the less cache misses the better.

The CPU implements some strategies to increase its operation ratio and avoid extra communication as well. One strategy is to try to predict (and fetch from memory) the next instruction to be executed. If the prediction is correct, execution time is reduced. This is implemented by a digital circuit called branch predictor. Another strategy is to dispatch as many instructions as possible in a pipeline, in such a way that at each cycle one instruction is completed. If the pipeline flow is kept, execution time reduces. The lower the execution time of an application, the lower is the energy consumption of the embedded device.

In this work, we use the gem5 simulator [16] configured to a specific platform (described in Section V) to run the system

and extract the execution metrics. The hardware metrics collected in this work are shown in Table II.

TABLE II. LIST OF HARDWARE METRICS

Metric	Description
Predicted Branches	The number of branches that were predicted correctly.
Missed Branches	The number of branches that were not predicted correctly.
Instructions per Cycles (IPC)	The ratio of total instructions executed by the total number of cycles for an application. Measures the efficiency of the pipeline mechanism.
Instruction Cache Misses	Number of misses occurring in the L1instruction cache.
Data Cache Misses	Number of misses occurring in the L1 data cache.
L2 Cache Misses	Number of misses occurring in the L2 cache (data cache).

B. Regression Analysis

The objective of regression analysis is to predict a single dependent variable (criterion) from the knowledge of one or more independent variable (predictor) [17]. When the problem involves a single independent variable, the statistical technique is called simple regression. When the problem involves two or more independent variables, it is termed multiple regression.

There is more than one way to make a regression analysis. In this work, we used six different algorithms for regression analysis, chosen due to familiarity and ease of use:

1. **Linear least squares regression (LM)** – the simplest linear regression technique. Target function is estimated by minimizing the sum of squares differences between actual and estimated values, called waste. The algorithm assumes that there is a linear relationship between the predictor attributes and the target attribute to be estimated. To apply this technique we used the method "lm" from package "Stats" [18] from R Project [19].
2. **Multivariate Adaptive Regression Splines (MARS)** – it is a non-parametric regression technique and can be seen as an extension of linear models that automatically models nonlinearities and interactions between variables. To apply this technique we used the method "earth" from package "earth" [20] from R Project [19].
3. **Support vector machine (SVM)** – supervised learning models with associated learning algorithms that analyze data and recognize patterns, used for regression analysis and classification. To apply this technique we used the method "svm" from package "e1071" [21] from R Project [19].
4. **Regression tree (CART)** – uses a decision tree as a predictive model which maps observations about an item to conclusions about the item's target value. Regression tree analysis is used when the predicted outcome can be considered a real number (e.g. a software metric). To apply this technique we used the method "rpart" from package "rpart" [22] from R Project [19].
5. **Neural networks (MLP)** – can be used for regression and classification, which can solve nonlinear problems. In this work, we used a Multilayer Perceptron (MLP) with backpropagation algorithm. To minimize the error, the BFGS algorithm is used, with weight decay of 0.0001. It is configured so that the hidden layer had 20 neurons, and the network training is done until 1000 iterations. To apply this technique we used the method "nnet" from package "nnet" [23] from R Project [19].
6. **Random Forest (RF)** – an ensemble learning method for classification and regression that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes output by individual trees. In this work, we used an implementation of Breiman's random forest algorithm contained in package "randomForest" [24] from R Project [19].

IV. PROPOSED APPROACH

As described, we intend to estimate a set of hardware metrics (hwM1, hwM2, ..., hwM*i*) based on a set of design metrics (dM1, dM2, ...,dM*j*) using a dataset where all values are numeric. Our objective of mining is to predict the hardware metrics (dependent variable) singularly (one at a time) from design metrics (independent variables).

We propose the use of KDD strategy to correlate design and hardware metrics. Once this correlation is established, the designer can provide a set of design metrics to the supporting system and receive the expected values for the corresponding code execution metrics. Thus, the designer can have a simple and fast feedback about his/her design and evaluates the effect of a design change in the software performance in the target platform

Our approach follows the framework proposed by Fayyad [25] with some adaptations to the problem in hand. Summarily, we use several regression algorithms to create different predictive models for each physical metric, and then select the best predictive model for each metric of interest. After this process, we have one predictive model for each hardware metric present in the initial dataset as illustrated in Fig. 1.

Among the many regression techniques available in the literature, we initially use the six techniques described in Section II: LM, SVM, CART, MLP, MARS and RF. We decided to choose more than one algorithm, because depending on the data, an algorithm can achieve better results than others.

To perform the data mining, data needs to be stored and formatted appropriately. In this stage, we include all metrics of all implementations in a single dataset. Thus, independent variables (design metrics) and dependent variables (physical

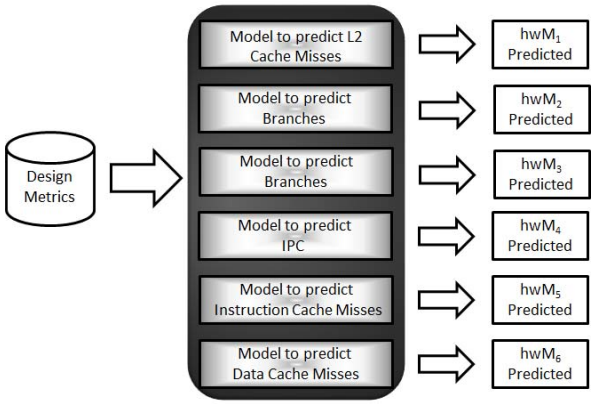

Fig. 1. Proposed Approach

Fig. 2. Input Pattern

metrics) are joined into a single line (registry) for each application, as shown in Fig. 2. In the figure, dM´s are class diagrams metrics shown in Table I and hwM represents physical metrics shown in Table II. Thus, each line of our dataset represents one set of metrics of one implementation of the target application.

To select the best algorithm for each metric, we use the ten-fold cross-validation technique [26]. In this technique, the original sample is randomly partitioned into ten equal size subsamples. Then a single subsample (out of the ten defined) is retained as the validation data for testing the model, and the remaining nine subsamples are used as training data. The cross-validation process is then repeated ten times, with each one of the 10 subsamples used exactly once as the validation data. Then, results from the folds can be averaged (or otherwise combined) to produce a single estimation.

Our approach runs a cross-validation for each algorithm and chooses the model with the lowest error. Root-Mean-Square Error (RMSE) - the square root of the variance of the residuals - is used to measure the error. RMSE indicates the absolute fit of the model to the data or, in other words, how close the observed data points are to the models predicted values. If the main purpose of the model is prediction, the RMSE is the most important criterion for fitness.

V. PRELIMINARY RESULTS

A. Experimental Setup

We use a set of 21 implementations of a basic banking system as case study. This set of implementations was chosen for three reasons. First, they represent different designs choices for a single specification. Second, we had access to both, class diagrams and source code for all implementations, to extract metrics and simulate in the target platform. Finally, the implementations were made by students with different levels of knowledge in OO design and programming thus presenting distinct design quality attributes. For each implementation, design metrics were extracted from the class diagram. Extracted metrics indeed show the differences among implementations. For instance, coupling metric Dep_out ranges from 2.1 to 8 (380% variation) among all 21 implementations.

The embedded target platform configured in the gem5 simulator is an ARMv7 Cortex-A15 CPU of 1.0 GHz with 64 KB of cache L1 (32 KB I-cache, 32 KB D-cache) and 1MB of cache L2. Each system implementation was executed in the simulated target platform and physical metrics were extracted. Such metrics also put in evidence the differences among implementations: cache misses range from 6.485.546 to 15.257.805 (235% variation) for example.

Following Harrell [27] one needs 10-20 observations per each parameter used as independent variable to be able to detect reasonable-size effects with reasonable power. Our case study currently consists of 21 implementations of one application. Due to this relatively small number of implementations, we can use only one design metric at a time as independent variable or our approach would not have statistical significance.

In our case study, the input data (following Fig. 2) for the regression algorithms is composed of 20 design metrics and 6 physical metrics and there are 21 data registries, one for each implementation of the bank system. In this case, 1,200 predictive models are created for a single hardware metric: for each training step we use 6 different algorithms, 20 different design metrics, and each training is divided into ten folds. As a result, 7,200 predictive models are created, but just six are selected at the end, one for each hardware metric. The model selected for each estimated metric is the one with smallest RMSE.

B. Experimental Results

Fig. 3 shows how all design metrics correlate to a single physical metric (branch prediction in this case). In the figure, the error rate obtained for the best algorithm selected for each design metric is shown. One can notice that only a sub-set of design metrics indeed correlates to a physical metric. We have also observed that the correlations between design and physical metrics vary with the system. Thus, for another system, different design metrics correlate to branch prediction. Thus, during maintenance, developers can focus on specific design metrics to evaluate and compare different design options without the burden of deployment or simulation on the target platform.

Table III shows the best combination of one design metric (first column) with one regression algorithm (second column) that results in the best predict model for one specific hardware metric (third column). The last column shows the normalized RMSE, obtained after the ten-fold cross validation. One can observe that, for the banking system and the defined target platform, Coupling Metrics (Dep_Out and Dep_In) and the number of operations (NumOps) are the best design metrics to predict cache and branch misses. Thus, during maintenance of this system, coupling metrics of the modified design are extracted and used to predict those specific physical metrics. If the predicted number of cache and/or branch misses does not seem reasonable, the developer can rapidly explore other design options.

TABLE III. RESULTS

Design Metric	Algorithm	Hardware Metric	NRMSE
Dep_In	SVM	L2 Misses	0.32%
Dep_Out	MARS	Predicted Branches	1.66%
NumOps	LM	Missed Branches	0.34%
NumOps	MARS	IPC	0.37%
Dep_Out	LM	Icache Misses	1.5%
NumOps	SVM	Dcache Misses	1.04%

VI. CONCLUSIONS AND FUTURE WORK

We have presented a KDD-based technique to assist a less experienced designer in implementing a higher quality embedded system within a stringent design time. The technique defines predictive models to estimate physical metrics from design metrics. Then, the impact on hardware platform of any alternative design can be quickly analyzed by extracting the new design metrics, feeding the predict model with them and checking the expected hardware metrics.

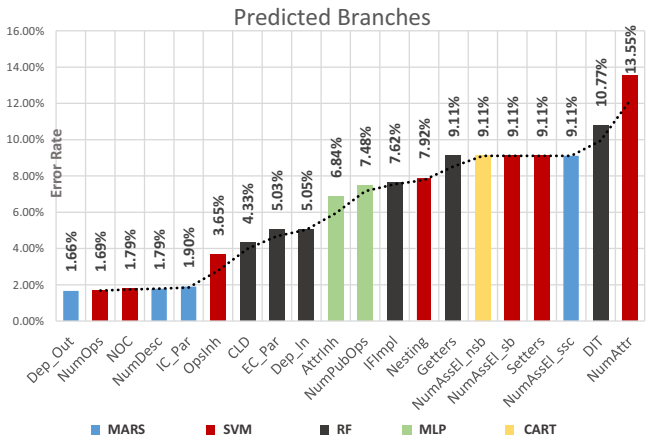

Fig. 3. Results of Predicted Branches

Our approach currently fits better the evolution stage in the SLCD because versions of similar applications provide the best prediction errors. The use of the proposed approach during the first design is now under investigation. Using this technique during the first design implies generating the predictive model with one application and using it with another system. Error rates in this situation are higher since correlations between metrics may differ from one system to another.

Current work also involves the inclusion of other applications in the training set as well as additional regression algorithms. The validation of the predictive models in a real design maintenance situation is also under development. At last, we are envisioning the construction of a complete design space exploration framework that includes a rich set of training algorithms and a user-friendly interface with automatic data (metrics) extraction and prediction. Such framework could be the foundation of a semi-automatic design space exploration tool for embedded software.

ACKNOWLEDGMENT

We thank Prof. Luigi Carro (UFRGS) for proposing this topic in the group and for his valuable inputs on a preliminary version of this paper.This work was supported in part by CNPq and PRONEM/FAPERGS.

REFERENCES

[1] A. Chatzigeorgiou and G. Stephanides, "Evaluating Performance and Power of Object-Oriented Vs. Procedural Programming in Embedded Processors," in *7th Ada-Europe International Conference on Reliable Software Technologies*, London, 2002.

[2] M. L. Cook, "Software Metrics: An Introduction and Annotated Bibliografy," *Software Engineering Notes,* vol. 7, no. 2, pp. 41-60, 1982.

[3] S. Ragab and H. Ammar, "Object oriented design metrics and tools a survey," in *nformatics and Systems (INFOS), 2010 The 7th International Conference on*, Cairo, 2010.

[4] J. Mattos, E. Specht, B. Neves and L. Carro, "Making Object Oriented Efficient for Embedded System Applications," in *Symposium on Integrated Circuits and Systems Design*, Florianopolis, 2005.

[5] A. Lake and C. Cook, "Use of Factor Analysis to Develop OOP Software Complexity Metrics," Corvallis, 1994.

[6] S. Chidamber and C. Kemerer, "A Metrics Suite for Object-Oriented Design," *IEEE Transactions on Software Engineering,* vol. 20, no. 6, pp. 476-493, Jun 1990.

[7] M. Lorenz and J. Kidd, Object-Oriented Software Metrics, Prentice Hall, 1994.

[8] W. Li and S. Henry, "Object-oriented metrics that predict maintainability," *Journal of Systems and Software,* vol. 23, no. 2, pp. 111-122, Nov 1993.

[9] L. Briand, P. Devanbu and W. Melo, "An Investigation into Coupling Measures for C++," Boston, 1997.

[10] R. K. Lind and K. Vairavan, "Effort, An Experimental Investigation of Software Metrics and Their Relationship to Software Development," *IEEE Transactions on Software Engineering,* vol. 15, no. 5, pp. 649-653, May 1989.

[11] D. Tegarden, S. Sheetz and D. Monarchi, "Effectiveness of traditional software metrics for object-oriented systems," Jan, 1992.

[12] N. E. Fenton and M. Neil, "Software Metrics: Roadmap," in *Proceedings of the Conference on The Future of Software Engineering*, Limerick, 2000.

[13] M. Oliveira, R. Redin, L. Carro, L. da Cunha Lamb and F. Wagner, "Software Quality Metrics and their Impact on Embedded Software," in *5th International Workshop on Model-based Methodologies for Pervasive and Embedded Software*, Budapest, 2008.

[14] R. S. Pressman, Software engineering : a practitioner's approach, 7th ed., New York: McGraw-Hill, 2010.

[15] *SDMetrics,* Zellertal, 2014. Available from http://www.sdmetrics.com/.

[16] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill and Wood, "{The Gem5 Simulator," *IGARCH Comput. Archit. News,* vol. 39, no. 2, pp. 1-7, 2011.

[17] J. F. Hair, W. C. Black, B. J. Babin and R. E. Anderson, Multivariate Data Analysis, Upper Saddle River: Prentice Hall, 2009.

[18] "The R Stats Package," [Online]. Available: http://stat.ethz.ch/R-manual/R-patched/library/stats/html/00Index.html. [Accessed 2014].

[19] R. Team, "R: A Language and Environment for Statistical Computing," R Foundation for Statistical Computing, 2008. [Online]. Available: http://www.r-project.org/. [Accessed 2014].

[20] S. Milborrow, "Package 'earth'," January 2014. [Online]. Available: http://cran.r-project.org/web/packages/earth/earth.pdf. [Accessed April 2014].

[21] D. Meyer and e. al., "Package 'e1071'," March 2014. [Online]. Available: http://cran.r-project.org/web/packages/e1071/e1071.pdf. [Accessed April 2014].

[22] T. Therneau, B. Atkinson and B. Ripley, "Package 'rpart'," March 2014. [Online]. Available: http://cran.r-project.org/web/packages/rpart/rpart.pdf. [Accessed April 2014].

[23] B. Ripley and W. Venables, "Package 'nnet'," March 2014. [Online]. Available: http://cran.r-project.org/web/packages/nnet/nnet.pdf. [Accessed April 2014].

[24] Liaw and MatthewWiener, "Package 'randomForest'," August 2013. [Online]. Available: http://cran.r-project.org/web/packages/randomForest/randomForest.pdf. [Accessed April 2014].

[25] U. Fayyad, G. Piatetsky-shapiro and P. Smyth, "From Data Mining to Knowledge Discovery in Databases," *AI Magazine,* vol. 17, pp. 37-54, 1996.

[26] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *International Joint Conference on Artificial intelligence*, San Francisco, 1995.

[27] F. E. Harrell, Regression Modeling Strategies, New York: Springer, 2002.