Early estimation of NFRs for Embedded System Using Design Metrics

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

***Abstract*—In this paper, an approach is proposed to assist the designer in evaluating the impact of a design choice with respect to the Non-Functional Requirements (NFRs) in embedded systems. We use design metrics extracted from class diagrams and regression models to estimate, in the early development stages, the impact of software changes on NFRs. This prediction can be used during the first design 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— Embedded Systems, Design Space Exploration; Software Metrics; Regression Analysis.

I. INTRODUCTION

Many core embedded functionalities such as GPS or image processing functions can be designed with high-level modeling frameworks such as Matlab which generate dedicated low-level code for a given platform. On the other hand, several components of the embedded system (including user interfaces and other functionalities that support the core embedded functions) have the same characteristics and 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 not only in terms of internal software quality metrics but mainly hardware resource consumption.

The complexity of embedded software 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 non-functional requirements (NFRs) of a given software in the target platform is a time-consuming activity. Thus, an important challenge in the embedded domain is to perform a thorough design space exploration within a stringent time-to-market.

Experienced developers reuse knowledge from previous projects as a guide to decision-making during design. However, a less experienced software designer may not know or recognize the solution that leads to the best NFRs performance for a given hardware and spend too much time completing the evaluation cycle in the target platform.

Within this context, 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 early part of our work that provides feedback about the correlation between design decisions and NFRs. This feedback can be used in the modeling stage to evaluate how a design choice affects the NFRs (e.g. energy consumption) thus helping the developer in building a better software and reducing refactoring during implementation. A usage example of our approach is illustrated in Fig. 1. The developer can explore the design space (without implementing and deploying actual code) until NFRs figures reach satisfactory levels. Then, code is implemented and implementation space can also be further explored, using the same principle, without deployment. Finally, when code is deployed in the target platform, a final tuning can be performed with less effort.

In this paper we focus only on the correlation between design and physical metrics. We analyze software metrics extracted from class diagrams (design metrics), and code execution (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. In other words, we try to extract information from previous projects (or earlier versions of the same project) that can help us making better design decisions.

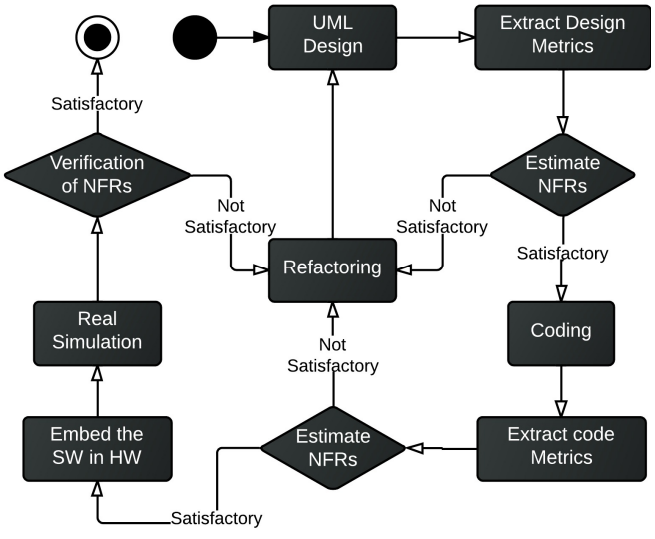

Fig. 1. Usage Flow of Our Approach

The proposed technique is evaluated with 21 different implementations of a banking system and 12 different implementations of a voting system. The case studies represent actual distinct implementations made from the same specification 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 as well as cost-effective measurement tools [2] [3] [4]. Comparatively few works have studied the relationship between metrics 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.

A major issue with software metrics though is that they must be used with care. This is due to several reasons such as the imprecision of the extraction methods, 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 [5]. 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] [6] 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 has still to define what is 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 [7]. One classification widely adopted nowadays, divides the 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 estimating the dynamic metrics (physical behavior) in embedded systems. The metrics used in this work are detailed below.

1) Class Diagrams Metrics

The metrics of class diagrams were extracted by SDMetrics tool [8]. 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) [9].
NumPubOps	Size	The number of public operations in a class. Aka Number of Public Methods (NPM) [9].
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 [11].
Num Desc	Inheritance	The number of descendants of the class. Counts the number of children of the class, their children, and so on [12].
DIT	Inheritance	The depth of the class in the inheritance hierarchy [11].
CLD	Inheritance	Class to leaf depth. The longest path from the class to a leaf node in the inheritance hierarchy below the class [13].
OpsInh	Inheritance	The number of inherited operations. Also known as Number of Methods Inherited (NMI) [14].
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 [15].
IC_Par	Coupling	The number of parameters in the class having another class or interface as their type [15].

2) Dynamic (Hardware) Metrics

Even though the term performance is usually related to execution and response time, it is reasonable to say that, in the mobile domain, the overall performance of an application includes not only the notion of time, but the efficient usage of hardware resources, including memory, battery, communication between CPU and memory, and so on.

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).
Energy	Total of energy consumed by the application.

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.

In this work, we used six different algorithms for regression analysis, all algorithms used are from R Project [18], as follows:

1. **Linear least squares regression (LM)** – This is the simplest linear regression technique where the target function is estimated by minimizing the sum of squares differences between actual and estimated values, called waste. Like all linear methods, 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" [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].
3. **Support vector machine (SVM)** – Is 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].
4. **Regression tree (CART)** – A regression tree 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 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].
5. **Neural networks (MLP)** – is a technique that 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].
6. **Random Forest (RF)** – Random forests are 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].

IV. PROPOSED APPROACH

As described, we intend to estimate a set of hardware metrics based on a set of design metrics. In this situation, we will be using a dataset where all values are numeric and our objective of mining is to predict the hardware metrics (dependent variable) but singularly (one at a time) from one or more design metrics (independent variables).

For this aim, we propose the use of KDD strategy to correlate design metrics with hardware metrics in such a way that the designer can have a simple and fast feedback about his/her design. 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 hardware metrics and evaluates the effect of a decision made in modeling stage on the overall performance of the software 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 dependent variable (in this case, physical metric) and then select the best predictive model for each metric of interest. After this process, we have one predictive model for each dependent metric. Those selected predictive models can be used then to evaluate a new design version, as shown in Fig. 1.

Among the many regression techniques available in the literature, we initially chose 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.

Data needs to be stored and formatted appropriately so that data mining algorithms can be applied. In this stage, we include all metrics of all implementations in a single dataset. Thus, independent variables and dependent variables are joined into a single line (registry), as shown in Fig. 2. In our case, each row in the dataset contains the metrics of each application in its entirety. In Fig. 2, IV′s are the independent variables (class

Fig. 2. Input Pattern

diagrams metrics) shown in Table I and DV's represent dependent variables (hardware metrics) shown in Table II, which we want to predict. Thus, each line of our dataset represents one set of metrics.

To select the best algorithm for each metric, we use the ten-fold cross-validation technique [26]. In this technique (depicted in Fig.3), 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.

The 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. In Equation 1 $X_{obs[i]}$ is observed value and $X_{pred[i]}$ is predicted value at time i.If the main purpose of the model is prediction, the RMSE is the most important criterion for fitness [27].

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(X_{obs[i]} - X_{pred[i]})^2}{n}} \quad (1)$$

Therefore, our approach runs a cross-validation for each hardware metric of interest (DV), with each design metric (IV). For all combinations, all regression algorithms are attempted and create corresponding predictive models. The predictive model representing the best combination (lowest RMSE) of these parameters is then selected, as depicted in Fig. 4. Only one (the best) predictive model for each hardware metric is saved in the end.

V. Preliminary Results

A. Experimental Setup

As an experimental setup, we used two sets of applications. Each set comprises different Java implementations of the same specification. The first specification is a simple banking system, which is composed of 21 implementations. The second specification is an electronic voting system for collegiate, which is composed of 12 implementations..

These applications were chosen for three reasons. First, they represent different types of implementations. The banking system is a typical CRUD application, whereas the voting system is mostly a reactive application. Second, for both applications we had access to the different implementations that represent distinct design and coding decisions over a single specification. Finally, the implementations were made by students with different levels of knowledge in OO design and programming. Thus, the available implementations really present distinct design quality attributes.

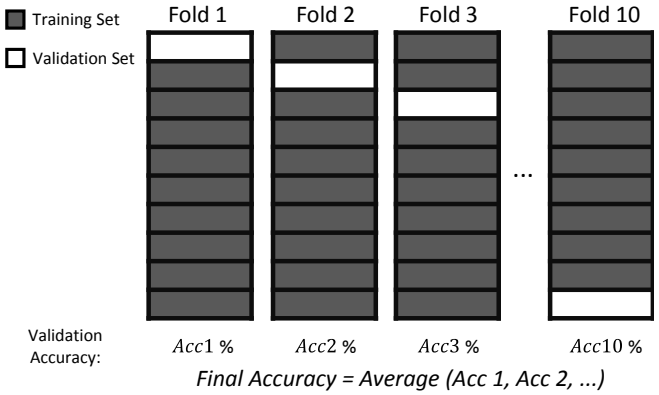

Fig. 3. 10-fold Cross Validation

For each implementation, design metrics are extracted from the class diagram and physical metrics are extracted from the simulation of the application in the target platform. 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. During simulation, all implementations are executed using the same set of input stimuli, that is, all implementations perform the same operations (from the user point of view) when hardware metrics are collected. Input stimuli were defined in such a way that more than 90% of code coverage is achieved in all implementations, meaning that the whole code is in fact executed.

After all data collected and stored in data registries as shown in Fig.2, we evaluated our framework with two different setups.

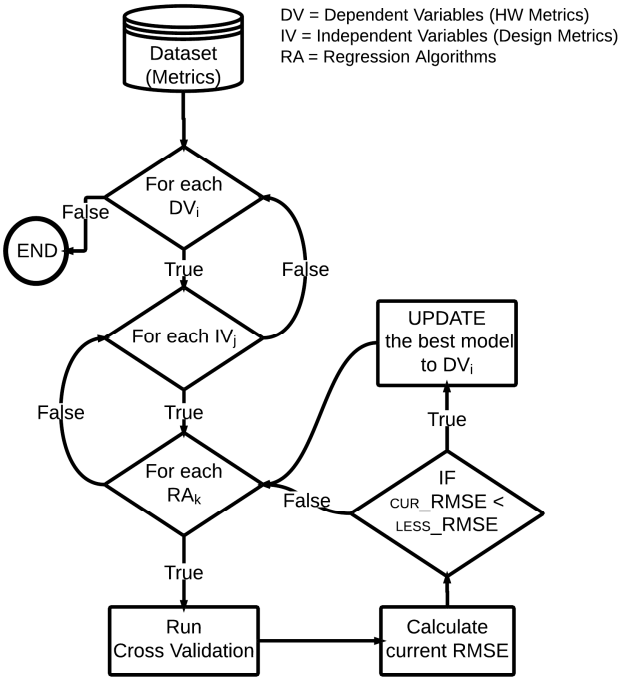

Fig. 4. Flowchart of Proposed Approach

First (setup A), we considered only the banking system for training and prediction (simulate evolutions of this application). In the second setup (setup B) we used the metrics of both applications together to evaluate the accuracy of our framework when submitted to different types of application.

B. Experimental Results

In this section, we present the preliminary results of our study with two case studies.

1) Setup A

Following Harrell [27] one needs 10-20 observations per parameter (independent variable) estimated 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 this case study, the input data for the regression algorithms is composed of 20 design metrics and 7 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. i.e., 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 seven are selected at the end, one for each hardware metric. As mentioned, the model selected for each estimated metric is the one with smallest RMSE.

Table III presents the results for this setup: 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. From Table III , one can observe that Coupling Metrics (Dep_Out and Dep_In) and the number of operations (NumOps) have a large influence on non-functional requirements in this case.

Fig. 5 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. The figure shows that not all design metrics correlate well to the physical metrics. Note also that the error displayed for each selected metric could be even worse if it had been another algorithm used. However, one can notice that a set of design metrics indeed correlates to physical ones and this information can be analyzed by the designer to choose the best strategy without the burden of deployment or simulation on the target platform.

2) Setup B

In this setup, both applications described before were used in the experiment, i.e., 21 implementations of the banking system plus 12 implementations of the voting system. The idea behind this setup is to analyze the performance of our framework for different applications. Table IV shows the corresponding results.

As expected, the error rate increased compared to the results shown in Table III. However, the error has not increased much considering that we are estimating a non-functional requirement from a UML abstraction. For example, estimate the

TABLE III. RESULTS FOR SETUP A

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%
DIT	SVM	Energy	0.94%

TABLE IV. RESULTS FOR SETUP B

Design Metric	Algorithm	Hardware Metric	NRMSE
NumAssEl_sb	CART	L2 Misses	2.90%
NumAnc	RF	Predicted Branches	5.19%
NumAssEl_sb	RF	Missed Branches	0.57%
NumAnc	MLP	IPC	0.9%
AttrInh	SVM	Icache Misses	3.7%
NumAnc	RF	Dcache Misses	6.02%
CLD	SVM	Energy	6.15%

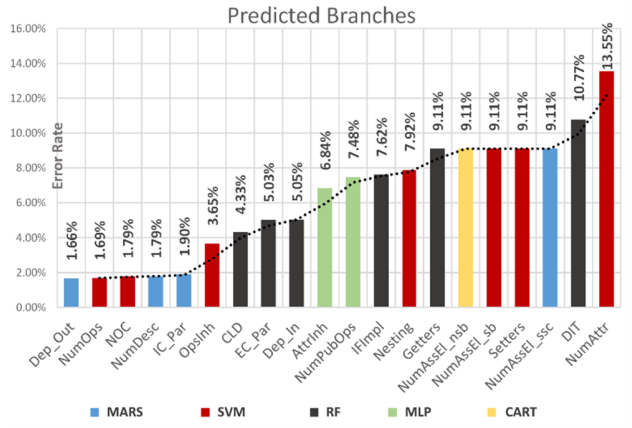

Fig. 5. Results of Predicted Branches for Setup A

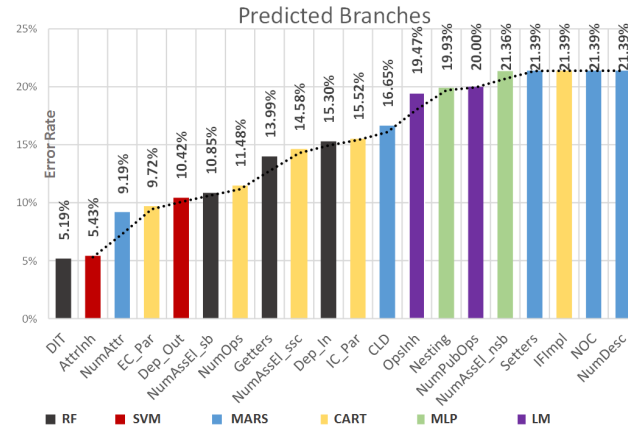

Fig. 6. Results of Predicted Branches for Setup B

power consumption of an application from a class diagram with an error rate of 6%, for many cases can be of great significance, resulting in a shorter design.

Fig. 6 shows the relation and error rates between all design metrics and one hardware metric (branch prediction in this case). It demonstrates that for different datasets there will be different metrics which will relate better with specific NFRs. For this reason, it is very unlikely that a static metric will correlate always (directly or inversely) with a hardware metric that is dynamic.

These results are supported by our initial study where we tried to find correlations between design metrics and hardware that were applicable for different applications and was not always possible. However, with the framework presented in this paper is possible to find the best metrics that correlate with each other and create predictive with a low error rate models.

VI. Conclusions And Future Work

We have presented a strategy to assist a less experienced designer in implementing a higher quality embedded system. The proposed approach is based on the use of KDD process to estimate physical metrics from design metrics. Based on a set of design and hardware metrics, the predict model is trained using regression analysis. 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. The process was implemented and validated with two case study for 21 distinct implementations of a banking system and 12 distinct implementations of a voting system.

It is important to note that the framework is not limited only to the prediction of hardware metrics from design metrics, it is possible to estimate code metrics from design metrics or estimate hardware metrics from code metrics, etc.

Current work involves the inclusion of other applications in the training set as well as additional regression algorithms. 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

This work was supported in part by FAPERGS and CNPq.

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

[3] S.R. Ragab and H.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.C.B. 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] Norman E. Fenton and Martin Neil, "Software Metrics: Roadmap," in *Proceedings of the Conference on The Future of Software Engineering*, Limerick, 2000.

[6] M.F.S. Oliveira, R.M. Redin, L. Carro, L. da Cunha Lamb, and F.R 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.

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

[8] SDMetrics, 2014. Available from http://www.sdmetrics.com/.

[9] A. Lake and C.R. Cook, "Use of Factor Analysis to Develop OOP Software Complexity Metrics," Oregon State Univ., Corvallis, Tech. Report 1994.

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

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

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

[14] Mark Lorenz and Jeff Kidd, *Object-Oriented Software Metrics*.: Prentice Hall, 1994.

[15] Lionel Briand, Prem Devanbu, and Walcelio Melo, "An Investigation into Coupling Measures for C++," , Boston, 1997.

[16] Nathan Binkert et al., "

[17] Joseph F. Hair, William C. Black, Barry J. Babin, and Rolph E. Anderson, *Multivariate Data Analysis*. Upper Saddle River: Prentice Hall, 2009.

[18] R Team. (2008) R: A Language and Environment for Statistical Computing. [Online]. http://www.r-project.org/

[19] The R Stats Package. [Online]. http://stat.ethz.ch/R-manual/R-patched/library/stats/html/00Index.html

[20] Stephen Milborrow. (2014, January) Package 'earth'. [Online]. http://cran.r-project.org/web/packages/earth/earth.pdf

[21] David Meyer and et al. (2014, March) Package 'e1071'. [Online]. http://cran.r-project.org/web/packages/e1071/e1071.pdf

[22] Terry Therneau, Beth Atkinson, and Brian Ripley. (2014, March) Package 'rpart'. [Online]. http://cran.r-project.org/web/packages/rpart/rpart.pdf

[23] Brian Ripley and William Venables. (2014, March) Package 'nnet'. [Online]. http://cran.r-project.org/web/packages/nnet/nnet.pdf

[24] Liaw and MatthewWiener. (2013, August) Package 'randomForest'. [Online]. http://cran.r-project.org/web/packages/randomForest/randomForest.pdf

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

[26] Ron 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] Frank E. Harrell, *Regression Modeling Strategies*. New York: Springer, 2002.