NFRs Early Estimation Through Software 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*— We propose the use of regression analysis to generate accurate predictive models for physical metrics using design metrics as input. We validate our approach with 40+ implementations of three systems in two development scenarios: system evolution and first design. Results show maximum prediction errors of 1.66% during system evolution. In a first design scenario, the average error is 15% with the maximum error still below 20% for all physical metrics. This approach provides a fast and accurate strategy to boost embedded software productivity and quality, by estimating Non-Functional Requirements (NFRs) during the first design stages.**

***Keywords*— Embedded Systems; Performance Estimation; Software Metrics; Regression Analysis.**

I. Introduction

As complexity of embedded software grows, the use of more abstract design strategies becomes crucial in this domain. However, achieving tight performance constraints while taking the benefits of high-level design and programming tools, is still a challenge [1].

It is known that design decisions taken at higher abstraction levels can lead to substantially superior improvements on issues such as performance and power. On the other hand, the evaluation of Non-Functional Requirements (NFRs) like energy and execution time in the target platform, although fundamental for the embedded software design, requires the completion of the whole design and deployment cycle thus becoming a time-consuming activity. Due to the short time-to-market, this situation precludes a wider exploration of the design space and inhibits the construction of a solution that represents the best design strategy in terms of both software quality and physical requirements. Thus, an important challenge in the embedded domain is to perform a thorough design space exploration within a stringent time-to-market.

Early estimation approaches have been trying to establish the relationship between high-level design indicators and the system performance in a target platform [1] [2] [3]. However, current approaches still cannot provide the developer with an actionable feedback about the impact of a design decision in the overall system performance.

In this paper, we analyze software metrics extracted from class diagrams, source code and real system execution to establish a relationship between them and help the embedded software developer in designing a higher quality system. The goal is to provide feedback, as soon as possible, and before system deployment, about design decisions and how they may affect the quality of the final system in the target platform.

The contributions of this paper are threefold: i) we propose the use of data mining techniques to explore the relationships between metrics of different levels of abstraction; ii) we show that such relationships are application-dependent; iii) we propose an early NFRs estimation framework that can be used to accelerate embedded software development and evolution.

The paper is organized as follows: Section II reviews related work and presents the necessary background for the proposed technique. Section III presents our approach based on regression analysis. Section IV discusses experimental results and the use of the proposed approach during software design and evolution. Section V concludes the paper.

II. Background

A. Related Work

A few works have tried to establish a relationship between the quality of the software and its overall performance in embedded platforms. Chatzigeorgiou and Stephanides [1] show that object-oriented (OO) programming can incur more execution time and power consumption in an embedded processor. Mattos et al. [4] propose the automatic transformation of OO code to make it more efficient in an embedded platform. Redin et al. [5] analyzed traditional software metrics in the context of embedded software, trying to achieve better software quality with lower impact on hardware costs. Corrêa et al. [3] propose the use of a neural network to find cross-correlations between code and physical metrics in a maintenance scenario. Still, no specific relationship between design, code and physical metrics has been defined yet. As a result, the impact of design decisions in the NFRs can only be evaluated after deployment in the target hardware.

B. Software Metrics

In this work, we use static metrics at design (class diagram) and code levels, to estimate dynamic (hardware) metrics. Design metrics are extracted from class diagrams by SDMetrics tool [6] while code metrics are extracted from source code by Understand tool [7]. Examples of such metrics are shown in Table I. The estimation framework proposed in Section III uses all metrics (from design and code levels) extracted by the tools. However, due to the lack of space, and without compromising the conclusions, only the design metrics that achieved the best results are presented here. We use the gem5 simulator [8] configured to a specific platform (described in Section IV) to run the applications and extract execution metrics related to the performance of the application in the target hardware. Examples of such metrics are energy consumption, IPC, number of cache misses, among others.

C. 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) [9]. In this work, we use six different algorithms for regression analysis: Linear Least Squares Regression (LM); Multivariate Adaptive Regression Splines (MARS); Support Vector Machine (SVM); Regression Tree (CART); Neural Networks (MLP); Random Forest (RF). These algorithms were chosen due to familiarity and ease of use. Detailed information about those algorithms can be found at [10].

III. Proposed Approach

We propose the use of regression analysis to relate static (design) metrics with dynamic (hardware) metrics in such a way that the developer can have a simple and fast feedback about his/her design. A framework is proposed to devise prediction models for each hardware metric of interest, using design metrics as inputs. Prediction models are defined during the training step using regression analysis with historical data. Once defined, the prediction models can be used to estimate each execution metric using one specific design metric (also defined by the proposed framework), as shown in Fig. 1. This way, in future designs or during system evolution, the developer can work at the design level, extract design metrics and have an estimative, before deployment, about the effect of a design decision on the overall performance of the software in the target platform.

Our approach follows the framework proposed by Fayyad [11] which refers to the overall process of discovering useful knowledge from data (Knowledge Discovery in Databases – KDD). In our case, all values in the dataset are numeric and our objective of mining is to use one static metric (independent variable) to predict one hardware metric (dependent variable) at a time. We use six regression algorithms in an attempt to detect the possible distinct relationships between metrics. Indeed, results will show that, depending on the data, an algorithm can achieve better results than others.

By using several regression algorithms we create different predictive models for each dependent variable. Then, we can select the model with smallest error for each metric of interest.

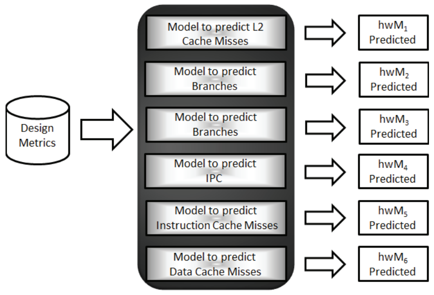
A. KDD applied to the estimation of physical metrics

Regression analysis is applied to historical data, i.e., metrics extracted from previous projects using a given hardware platform. As mentioned, we explain the proposed framework

TABLE I. LIST OF DESIGN METRICS

Metric	Cat.	Description
NA	S	Number of attributes in the class.
NM	S	Number of Methods. Aka number of operations in a class [15]
NPM	S	Number of Public Methods (or operations) in a class [15]
Setters	S	Number of operations with a name starting with 'set'.
ND	I	Total number of descendants of the class. [18]
DIT	I	Depth of the class in the inheritance hierarchy [17]
CLD	I	Class to leaf depth. The longest path from the class to a leaf node in the inheritance hierarchy below the class [16]
NIA	I	Number of inherited attributes.
D_Out	C	Number of elements on which this class depends.
D_In	C	Number of elements that depend on this class.
NAE_b	C	Nº of associated elements in the same scope branch as the class.
NAE_nb	C	Nº of associated el. not in the same scope branch as the class.

* S = Size; I = Inheritance; C = Coupling.


Design Metrics
Model to predict L2 Cache Misses
Model to predict Branches
Model to predict Branches
Model to predict IPC
Model to predict Instruction Cache Misses
Model to predict Data Cache Misses
hwM1 Predicted
hwM2 Predicted
hwM3 Predicted
hwM4 Predicted
hwM5 Predicted
hwM6 Predicted
Fig. 1. Proposed Approach (After Training)

using only design and hardware metrics, but experiments using code metrics have presented similar results. The proposed framework is composed of three main steps as detailed next.

Data needs to be stored and formatted appropriately so that data mining algorithms can be applied. Thus, the *first step* in the proposed framework is to include all metrics extracted from previous projects (both static and dynamic ones) in a single dataset. Independent variables (IV´s) and dependent variables (DV´s) are joined into a single line (registry), as shown in Fig. 2. In Fig. 2, IV´s are class diagrams metrics shown in Table I and DV's represent physical metrics. Thus, each line of our dataset represents the complete set of metrics for all abstractions of one particular implementation.

The *second step* in the framework consists in running all regression algorithms over the assembled dataset. Each algorithm generates a model that relates each IV to each DV in the dataset. In this phase, we use the ten-fold cross-validation as model validation technique to avoid bias [12]. In this technique, the original sample is randomly partitioned into ten equal sized 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 to produce a single estimation.

We use the Root-Mean-Square Error (RMSE) to measure the error [13], i.e., to measure how close the observed data points are to the values generated by the prediction models. Thus, at the end of the second step, one has six models relating each design metric to each execution metric in the dataset. Then, for each pair of metrics, only the model with smallest RMSE (the one that best relates that specific pair of metrics) is retained.

Finally, in the *third step*, the framework finds the best design metric one must use to predict each execution metric of interest. This is done by selecting, for each hardware metric in the dataset, the static metric (and corresponding prediction model selected in the previous step) with the smallest RMSE.

In summary, 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 the corresponding predictive models are created. At last, the best combination (lowest RMSE) of these parameters is selected as the predictive model for a given physical metric.

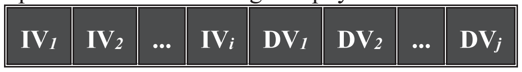


Fig. 2. Input Pattern

IV. Experimental Results

A. Experimental Setup

Our experimental setup is composed by three sets of applications. Each set comprises different Java implementations of the same specification. The first set (App 1) contains 21 implementations of a simple banking system. The second specification (App 2) is a simple ecommerce system for which we have 12 implementations available. The third specification (App 3), also with 12 implementations, is an electronic voting system for collegiate decisions. The banking and ecommerce systems are typical CRUD (create-read-update-delete) data-oriented applications, whereas the voting system is mostly a reactive application (control-oriented). For all applications we had access to different implementations that really represent distinct design and coding decisions over a single specification. The implementations naturally present distinct design quality attributes as well as they were made by students with different levels of knowledge in OO design and programming.

For each implementation, design metrics were extracted from the class diagrams. Extracted metrics indeed showed the differences among implementations. For instance, among all 21 implementations of the banking system, the coupling metric D_out ranges from 2.1 to 8 (380% variation) whereas LOC (lines of code) ranges from 364 to 2,220.

The embedded target platform configured in the gem5 simulator [8] 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. Again, the differences among implementations at the lower level were observed: cache misses ranged from 6,485,546 to 15,257,805 (235% variation) for example. During simulations, all sets of 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.

We implemented our approach using regression analysis libraries available in R Project [14]. We evaluated our framework with two different setups. First (setup A), we applied the technique (training and prediction) for each system individually. In other words, the training dataset is assembled using metrics of all implementations of a single specification. This setup represents the use of the proposed method during system evolution where the design under evaluation, although different, is very similar to a previous version of the system.

In the second setup (setup B) we used the metrics of applications 1 and 3 together to train and create the predictive models. Then, we used the second application (ecommerce) to evaluate the accuracy of our framework when submitted to a totally different application not used during training. In this case, software metrics of one implementation of the ecommerce system are used as input to the predictive models. This last setup simulates the use of the proposed framework for a new project in the organization but using historical data as reference for a first performance estimation.

B. Experimental Data

1) Setup A

Table II presents the results considering the system evolution scenario: for each application, the best combination of one design metric (second column) with one regression algorithm (third column) that results in the best predictive model for one specific hardware metric (first column) is presented. The fourth column shows the normalized RMSE, obtained after the ten-fold cross validation in each case.

From Table II three main observations should be noted: i) the error rate is very low in all cases (maximum of 2%), showing that in case of system evolution a very accurate estimation of final performance can be achieved; ii) different regression algorithms perform better than others, depending on the application; iii) the software metric that best relates to a physical one also varies depending on the application. These last two observations corroborate the need for a thorough exploration of the dataset that motivated this work.

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 (result of Step 2 in the proposed framework). One can notice that only a sub-set of design metrics indeed relates well to a physical metric and, as shown in Table II, the best relations between design and physical metrics vary with the system.

As an example of use, let us consider two alternative designs (imp02 and imp03) as possible evolutions of the original design (imp01) of the ecommerce system. Let us also assume we want to know the energy consumption of each alternative design. From Table II we know that, for this system, energy consumption best relates to design metric NM. Hence, for each alternative design, we extract this metric and feed it to the corresponding prediction model. Table III shows the data related to this example. The value of the input design metric is given in the second column and the estimated energy is given in the third column. The percentage variations of the extracted metrics with respect to the original design are also shown. With this information, the developer can reason and choose the best solution: lowest energy (imp03) or best trade-off between software and hardware metrics (imp02). In summary, using the proposed method the developer has all information necessary to choose which aspect of the system should be the focus of attention during design space exploration.

Table II – Results For Setup A

*HW Met	App 1 (Banking) Training and Testing			App 2 (Ecommerce) Training and Testing			App3 (Voting) Training and Testing		
	DM	**Alg.**	**NRMSE**	**DM**	**Alg.**	**NRMSE**	**DM**	**Alg.**	**NRMSE**
MB	NM	SVM	0.34%	CLD	RF	0.39%	CLD	SVM	0.04%
PB	D_Out	RF	1.66%	NAE_nb	LM	0.19%	NM	SVM	0.19%
ICM	D_Out	SVM	1.50%	NA	SVM	0.08%	NPM	SVM	0.48%
DCM	NM	SVM	1.04%	NIA	SVM	0.07%	D_Out	SVM	0.00%
L2M	D_In	SVM	0.32%	NPM	CART	0.29%	ND	SVM	0.02%
IPC	NM	SVM	0.37%	D_In	MLP	0.44%	NA	MARS	0.54%
E	DIT	SVM	0.94%	NM	SVM	0.14%	Setters	SVM	0.31%

*Energy (E); Instructions per Cycles (IPC); L2 Cache Misses (L2M); Data Cache Misses (DCM); Instruction Cache Misses (ICM); Predicted Branches (PB); Missed Branches (MB)

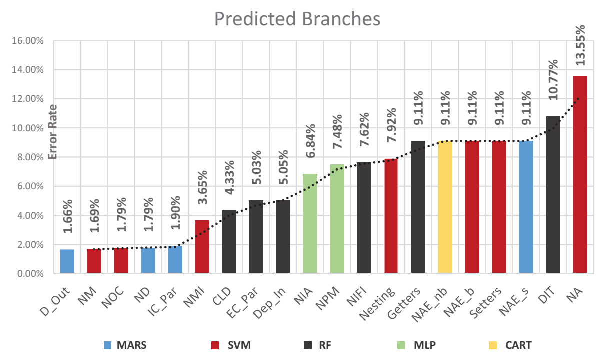


Fig. 3. Results of Predicted Branches for Setup A

TABLE III. USE OF THE PROPOSED METHOD TO PREDICT ENERGY

Design option	Extracted Design Metric (NM)	Estimated Energy (mJ)	Actual Energy (mJ)
imp01	3.5	4294.177	4270.207
imp02	5.79 (+39%)	3506.084 (-18.3%)	3431.016 (-19.6%)
imp03	2.69 (-23%)	3817.796 (-11%)	3743.042 (-12.3%)

2) *Setup B*

In this setup we joined the metrics of applications 1 and 3 (bank and voting systems) in a single dataset. Notice that we joined data-driven and control-driven examples. Then we use the resulting predictive models to estimate physical metrics for the third application (ecommerce, data-driven) thus simulating the scenario of a new project in the organization. The results for this setup can be seen in Table IV.

Results show that increasing the dataset with metrics of different applications (data and control-oriented) indeed reduces the accuracy of the predictive models. Still, error rates remain in an acceptable range, mainly considering its use as a first estimation for a new project. According to Table IV, the developer can, in the first stages of a new project, have a good estimation of the expected performance of the new design (average error rate of 13% when testing with design metrics of App 2 – third column). This early estimative can guide both, the software and the hardware design. Moreover, as the design evolves, the prediction system can be fed with new data to improve estimation accuracy (average error rate of 4% when testing with design metrics of Apps 1 or 3 – second column).

V. CONCLUSIONS AND FUTURE WORK

We have presented a KDD-based technique to define highly accurate NFRs prediction models using design metrics as inputs. Using the proposed approach, the impact of any alternative design in the target platform can be quickly analyzed before deployment. Experimental results show the applicability of the framework not only for system evolution, but also during the first stages of a new project, thus boosting system productivity. Current work includes the study of additional regression algorithms as well as the use of other applications and hardware platforms in the training set. The validation of the technique in a real design environment is also under consideration.

ACKNOWLEDGMENT

We thank the students that implemented the case studies. This work is partially funded by CNPq and FAPERGS grants.

TABLE IV. RESULTS FOR SETUP B

HW Met	Training and Testing with App 1 and 3			Training with App 1 and 3 and Testing with App 2		
	Metric	**Alg.**	**Error**	**Metric**	**Alg.**	**Error**
MB	NAE_b	RF	6.18%	NAE_nb	CART	14.76%
PB	DIT	RF	5.19%	NAE_nb	MLP	16.08%
ICM	NIA	SVM	3.66%	D_Out	SVM	17.85%
DCM	DIT	RF	6.02%	NAE_nb	CART	11.95%
L2M	NAE_b	CART	2.90%	ND	MLP	19.18%
IPC	DIT	MLP	0.91%	NAE_b	CART	2.79%
E	CLD	SVM	6.15%	NOC	CART	12.87%

REFERENCES

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

[2] M. Oliveira and e. al., "Software Quality Metrics and their Impact on Embedded Software," in *5th Int. Workshop Model-based Methodologies for Pervasive and Embedded Software*, Budapest, 2008.

[3] U. Corrêa, L. Lamb, L. Carro, L. Brisolara and J. Mattos, "Towards estimating physical properties of embedded systems using software quality metrics," in *IEEE 10th Int. Conf. Comput. and Inform. Technology (CIT)*, Bradford, 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] R. M. Redin, M. F. Oliveira, L. B. Brisolara, J. C. B. Mattos, L. C. Lamb, F. R. Wagner and L. Carro, "On the Use of Software Quality Metrics to Improve Physical Properties of Embedded Systems," *Distributed Embedded Systems: Design, Middleware and Resources,* vol. 271, pp. 101-110, 2008.

[6] J. Wüst, "SDMetrics," Sept. 2014. [Online]. Available: http://sdmetrics.com/. [Accessed April 2014].

[7] S. Toolworks, "Understand," SciTools, Sept. 2014. [Online]. Available: http://www.scitools.com. [Accessed Jul 2014].

[8] 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.

[9] J. F. Hair, W. C. Black, B. J. Babin and R. E. Anderson, Multivariate Data Analysis, Prentice Hall, 2009.

[10] R. Team, 2014 Nov. [Online]. Available: http://cran.r-project.org/web/packages/available_packages_by_name.html.

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

[12] 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.

[13] F. E. Harrell, Regression Modeling Strategies, NY: Springer, 2002.

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

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

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

[17] S. Chidamber and C. Kemerer, "A Metrics Suite for Object-Oriented Design," *IEEE Trans. Softw. Eng,* vol. 20, no. 6, pp. 476-493, Jun 1990.

[18] W. Li and S. Henry, "Object-oriented metrics that predict maintainability," *J. Syst. Softw.,* vol. 23, no. 2, pp. 111-122, Nov 1993.