On the Use of Software Quality Metrics to Improve Physical Properties of Embedded Systems

Ricardo M. Redin, Marcio F. S. Oliveira, Lisane B. Brisolara, Julio C. B. Mattos, Luis C. Lamb, Flávio R. Wagner, and Luigi Carro

Abstract As software production achieves a growing importance in the embedded systems world, quality evaluation of embedded software and its impact on physical properties of embedded systems becomes increasingly relevant. Although there are tools for embedded software design that improve software specification and verification, we are still short of a tool that supports the designer's decisions on the best design strategy regarding low level, physical characteristics like performance, energy, and memory footprint, which are critical in the embedded domain. In this paper, we provide an analysis of the correlation between software quality metrics and physical metrics for embedded software. By means of experiments, we investigate the impact of software engineering best practices on embedded software and show that software quality metrics can be used to guide design decisions toward improving physical properties of embedded systems.

Key words: embedded software, software engineering, measurement, quality metrics

1 Introduction

Software engineers have been improving the software design process, and new methods have been proposed for all software development steps, from requirements specification to testing. New programming paradigms have arisen, such as Object-Orientation (OO) and Aspect-Orientation (AO), as well as new development methods such as Model-Driven Engineering. A key factor of any engineering process is the measurement and assessment of its characteristics; thus different metrics to gauge and improve the quality of software products have been also proposed. Such

Ricardo M. Redin · Marcio F. S. Oliveira · Lisane B. Brisolara · Julio C. B. Mattos · Luis C. Lamb · Flávio R. Wagner · Luigi Carro
Institute of Informatics, Federal University of Rio Grande do Sul (UFRGS), Brazil

metrics have been designed to evaluate concepts such as reuse, abstraction, cohesion, coupling, and other software attributes.

In the case of embedded systems, differently from the traditional software domain, the main metrics currently in use are the physical ones, such as performance, memory, energy, power, size, and weight, guided by design constraints. Other important and related metrics are reuse, time-to-market, and price. Although many methodologies extract the physical metrics by proposing estimation or simulation tools, the reuse and time-to-market factors are approached only through design methods without direct or indirect evaluation.

However, the hard constraints typically found in embedded systems do not allow the embedded system community to benefit from the advances in traditional software methodologies As a result, the most critical challenge for Systems-on-Chip (SoC) design is the software development process, which now accounts for 80% of the cost of embedded system development. Notwithstanding the growth in the use and application of software engineering methodologies in embedded software development, the current practice of embedded software development is still unsatisfactory, in particular in industry .

Traditional (or classical) quality metrics provided by software engineers have been successfully applied to improve the software quality for general-purpose systems, leading to improvements in reuse and time-to-market. Traditionally, these metrics help designers to increase properties such as abstraction and reuse, which are good for time-to-market and maintainability. However, some best design practices for conventional software cannot be applied to embedded software because they can cause a negative impact on the physical metrics.

In this work, we investigate the relationship between traditional (classical) software quality metrics and the relevant physical metrics for embedded systems. Different design decisions over the application model influence these metrics, thus we intend to find out which software quality metrics are relevant for embedded software design. Moreover, we show that the best design practices of traditional software can negatively impact the physical properties of embedded systems, which implies that some sacrifices in terms of reuse or maintainability are required to achieve a better performance. Finally, we propose to use the knowledge about the relationship between quality and physical metrics to suggest modifications in the modeling solution that will improve this solution regarding the physical metrics.

The remaining of this paper is organized as follows. Section 2 presents related background work. Section 3 describes the software quality metrics selected for our analysis. Section 4 presents the experiments conducted and our main results. Section 5 concludes the paper and points out directions for future work.

2 Related work

There are several proposals of metric frameworks to evaluate the quality of the software products in the software engineering literature, see e.g.. In the case of object-

oriented software, a well-known survey of quality metrics is. In a more recent empirical study of OO metrics, the authors apply several quality metrics to three different projects and study the relationship between these metrics .

However, only a few works relating software quality measures for embedded software products and physical metrics for embedded systems have been published so far. For instance, in the authors describe the results of an experiment where four different mobile devices running Role Playing Games applications are analyzed in terms of software quality metrics and performance. The study shows that the development effort can be greatly reduced without compromising the performance through the reuse of platform and/or software components.

Our work focuses on measuring the correlation between object-oriented quality metrics for traditional software design and physical metrics for embedded systems design. We aim at finding out how the correlation between these metrics can be used to aid a designer to improve the embedded software quality and still achieve better results in terms of performance, memory, and power consumption. Moreover, we show that one can use specific quality metrics as reliable predictors for the impact of software design decisions on the final system physical metrics.

3 Software metrics

As the key of any engineering process is measurement, research efforts have provided many measures and metrics to evaluate processes, software products, and projects in order to guide design decisions. A set of important metrics was selected from and. Since there is no well-defined or widely accepted metrics classification, we group these metrics by the attribute which the metric refers to, in order to facilitate the presentation. The classification and the used metrics are as follows.

Coupling: It measures the relationship between components, including calls, and number of instances. High values of these metrics lead to an application that is poor in encapsulation, reuse, and maintainability. The following metrics fit into this category: Afferent Coupling (Ca), Efferent Coupling (Ce), and Instability (I).

Cohesion: It measures the degree to which the elements of a scope are functionally related. The recommendation from software engineering is to use strongly cohesive modules, which implement functionality that is related to one feature of the software and requires little or no interaction with other modules. Lack of Cohesion of Methods (LCOM) is the cohesion metric used in this work.

Extensibility and reuse: These metrics evaluate the possible reuse of a scope and the capacity of it to be extended. Abstractness (A), Normalized Distance from Main Sequence (Dn), and Depth of Inheritance Tree (DIT) are metrics used in this work.

Population (or size) metrics: These metrics measure the system in terms of attributes, methods, and classes. They are also associated to complexity. In general, higher values of these metrics mean an increase in memory footprint, lower performance, and a more complex solution. Nevertheless, the distribution of the population metrics has more impact on the dynamic behavior of the application. Almost

all population metrics count the number of a given structure inside the application code. The used population metrics are: Number of Attributes (NOA), Number of Classes (NOC), Number of Methods (NOM), Number of Packages (NOPK), Number of Parameters (NOP), Number of Static Attributes (NOSA), Number of Static Methods (NOSM), and Total Lines of Code (TLOC).

Complexity: These metrics measure the hardness to understand or express the problem/algorithm. They are related to alternative execution flows, element granularity/hierarchy, and nested execution. Metrics in this category are: McCabe Cyclomatic Complexity (VG), Method Lines of Code (MLOC), Nested Block Depth (NBD), and Weighted Methods per Class (WMC).

There is a large number of software metrics. We have selected this set of metrics because they are commonly and widely used in the software engineering domain and there are several tools to automatically extract them from source code or even from UML models. Surely, other metrics could be applied in this study, and other important metrics are planned for future experiments.

4 Experiments

We have carried out experiments aiming at: (1) verifying the correlation between software quality metrics and physical ones; (2) identifying the relevant quality software metrics for embedded software design; (3) measuring the impact of specification strategies by using software quality metrics. Furthermore, we show that these metrics can be used to improve embedded software with respect to its physical properties. We will also show that some sacrifices in terms of reuse or maintainability are required in order to achieve a hard constrained performance.

The analyzed applications are a wheelchair control and an MP3 player. The wheelchair control application consists of a real-time embedded system dedicated to the automation and control of an intelligent wheelchair that helps people with special needs. For this experiment, we have implemented only the wheelchair movement control, which is an essential use case of the system. The MP3 player is an application that is usually embedded in many consumer electronics systems, used to play music in a compressed data format. This application presents a dataflow processing channel in which many algorithms must be executed until the compressed data can be played.

Different solutions were developed for both applications. All solutions were implemented using Java for the target platform. For every alternative implementation, a synthesis tool was used to obtain the hardware description and the Java byte codes for the application. From the final implementation, in Java byte codes, we extracted the physical metrics by using a cycle-accurate simulation , while from the Java source code we extracted the software quality metrics by using the Eclipse Metrics plug-in.

The physical data from cycle-accurate simulation and software metrics from Eclipse Metrics were matched using the cross-correlation formula that measures

the similarity of two arrays of data. Results obtained for both quality and physical metrics, including the cross-correlation between them, are presented in the following.

4.1 Experimental results

Firstly, we have analyzed the cross-correlation between these different metrics for a given application and afterwards among different applications to observe whether the achieved correlation is similar for all applications or not. A positive cross-correlation means that an increase on a given quality metric results in an increase on the related physical property. On the other hand, negative values translate to an inverse relationship.

In the MP3 player experiment three different solutions were analyzed. Sol-1 is object-oriented and follows as much as possible the recommendations of software engineering. Sol-2 is OO too, but much more concerned with physical proprieties of the final system. Sol-3 is entirely targeted at good values for physical proprieties of the resulting product and thus entirely static. Table 1 shows the physical properties obtained for each MP3 solution. Analyzing these results, one can observe that the best solution considering traditional software engineering paradigms (Sol-1) is the worst one regarding physical metrics.

Table 2 presents the quality metrics and the correlation between them and the physical properties. For all software metrics the maximum value or total is showed, except the metrics marked with an asterisk (*), for which we have used the average value because their total values just tell us where to look for bad code constructs. An average value, in turn, shows us how much a good or bad behavior is distributed across the entire application.

Table 1 Extracted physical metrics from the MP3 player.

Property	Sol. 1	Sol. 2	Sol. 3
Program memory	238,484	237,192	242,688
Data memory	146,812	117,756	324,733
Cycles	1,830,675,876	830,365,894	239,748,559
Energy (J)	79.8575	36.2221	21.9624

For the wheelchair experiment four solutions were analyzed. Sol-1 is the most concerned about performance, energy, and memory of the final system. All operating system services were implemented by the application, and only the required services are implemented. Sol-2, in turn, is the most concerned with the quality of the software product. It uses threads and an underlying platform that supports multithreading, among other features. Sol-3 and Sol-4 use the same platform as Sol-2 and differ from each other in design strategies. Table 3 summarizes the physical proprieties of the wheelchair solutions, and Table 4 shows the software metrics val-

Table 2 Extracted software quality metrics and its cross-correlation to physical metrics on MP3 Player.

Property	Sol. 1	Sol. 2	Sol. 3	Prog. Mem.	Data Mem.	Cycles	Energy
Abstractness	0.143	0	0.2	0.858	0.804	-0.132	0.005
Afferent Coupling	3	2	5	0.994	0.979	-0.536	-0.416
Depth of Inheritance Tree	2	1	2	0.682	0.608	0.147	0.281
Efferent Coupling	4	2	3	0.225	0.130	0.622	0.723
Instability	0.75	1	1	0.293	0.384	-0.930	-0.972
Lack of Cohesion of Methods*	0.655	0.42	0.245	-0.671	-0.740	0.998	0.980
McCabe Cyclomatic Complexity*	1.832	7.448	6.492	0.137	0.232	-0.860	-0.922
Method Lines of Code	4101	4618	5675	0.850	0.897	-0.942	-0.887
Nested Block Depth*	1.188	2.23	2.305	0.349	0.438	-0.951	-0.984
Normalized Distance*	0.707	0.556	0.626	0.184	0.088	0.654	0.752
N. of Attributes	186	112	6	-0.797	-0.852	0.969	0.926
N. of Children	106	22	4	-0.447	-0.531	0.978	0.997
N. of Classes	27	26	64	0.979	0.994	-0.769	-0.674
N. of Methods	463	85	47	-0.371	-0.458	0.957	0.988
N. of Packages	5	3	6	0.884	0.834	-0.185	-0.048
N. of Parameters	7	14	6	-0.761	-0.695	-0.033	-0.169
N. of Static Attributes	98	58	597	0.987	0.998	-0.740	-0.641
N. of Static Methods	127	2	71	0.283	0.189	0.574	0.681
Total Lines of Code	7891	6853	8423	0.887	0.838	-0.191	-0.055
Weighted methods per Class	1081	648	766	-0.030	-0.127	0.800	0.875

ues and the cross-correlation between software and physical metrics. In Table 3, BC identifies the metric value for the Best Case execution of the controller.

Table 3 Physical metrics obtained from the Wheelchair Movement Controller.

Property	Sol. 1	Sol.2	Sol. 3	Sol. 4
Program memory	2,063	6,248	5,208	5,094
BC Data memory	372	582	431	421
BC Performance	1,898	28,588	9,104	7,776
BC Energy	2,714,132	40,569,570	12,916,022	11,026,748

As one of the applications is dataflow and the other one is control flow, some correlations differ from one experiment to the other. As expected, performance and energy are highly-correlated physical properties. In all experiments these two metrics follow the same tendencies, and correlation between software metrics and each of them hardly differ significantly from the other.

4.2 Experimental results analysis

While some good practices of software engineering cause an overhead in the physical properties of embedded systems, other ones can help to design better products

Table 4 Extracted software quality metrics from the Wheelchair Movement Controller and its cross-correlation to physical ones.

Property	Sol. 1	Sol. 2	Sol. 3	Sol. 4	Prog. Mem.	Data Mem.	Cycles	Energy
Abstractness	0	0	0	0	0.000	0.000	0.000	0.000
Afferent Coupling	1	4	2	2	0.849	0.994	0.992	0.992
Depth of Inheritance Tree	1	2	2	2	0.958	0.584	0.572	0.571
Efferent Coupling	1	2	2	2	0.958	0.584	0.572	0.571
Instability	1	0.5	0.667	0.667	-0.995	-0.846	-0.837	-0.837
Lack of Cohesion of Methods*	0.71	0.639	0.51	0.519	-0.588	0.025	0.040	0.041
McCabe Cyclomatic Complexity*	1.238	1.312	1.261	1.25	0.773	0.995	0.996	0.996
Method Lines of Code	58	94	62	49	0.525	0.916	0.922	0.922
Nested Block Depth*	1.143	1.25	1.174	1.15	0.720	0.983	0.985	0.985
Normalized Distance*	0.5	0.567	0.583	0.583	0.885	0.419	0.405	0.404
N. of Attributes	0	22	20	17	0.988	0.710	0.700	0.699
N. of Children	0	0	0	0	0.000	0.000	0.000	0.000
N. of Classes	5	7	7	7	0.958	0.584	0.572	0.571
N. of Methods	2	29	20	17	0.982	0.883	0.876	0.875
N. of Packages	3	4	4	4	0.958	0.584	0.572	0.571
N. of Parameters	3	3	3	2	-0.163	0.224	0.234	0.234
N. of Static Attributes	20	28	17	19	0.408	0.864	0.870	0.871
N. of Static Methods	8	3	3	3	-0.958	-0.584	-0.572	-0.571
Total Lines of Code	146	283	190	170	0.782	0.996	0.996	0.996
Weighted methods per Class	26	42	29	25	0.629	0.962	0.966	0.966

without affecting physical properties or even improving them. In this section, we analyze our experimental results and show some tradeoffs between software engineering guidelines and code optimizations to improve as much as we can physical properties of the final system, looking for a good balance between both sides.

The best OO practices indicate that a reduced coupling is desired, so the coupling metrics Ca, Ce, and Instability should have small values. We observed that there is a high correlation (around 0.9) among the metric Ca and data and program memory, which suggests that this metric impacts on the memory footprint. This is confirmed, by the case studies, where Sol-1 of the wheelchair controller and Sol-2 of the MP3 player present the smallest Ca value and achieve the smallest memory size in comparison with the other solutions. Instability (I) indicates if a package is stable or not. A value of zero is required. A correlation around -0.9 was found between Instability and energy as well as between Instability and performance, showing that this quality metric has a negative impact on these physical metrics. It means that solutions with higher *I* values are the best ones in terms of performance/energy, as confirmed by our results.

The OO paradigm leads designers to build cohesive modules that require little or no interaction with other modules. It suggests that, in order to have components architecturally and logically well defined, smaller values for Lack of Cohesion (LCOM) are desired. We have observed that the best solution for the MP3 player in terms of energy/performance is Sol-3, which has the smallest LCOM value. The opposite situation is found for the wheelchair controller, where Sol-1 is the best so-

lution for all physical properties and presents the highest LCOM. The reason for that is the fact that Sol-1 of the wheelchair controller has the smallest number of attributes (NOA), which is a metric strongly related to performance and energy.

High reuse is desired in all traditional software projects. High values for the metrics Abstractness (A) and Depth of Inheritance Tree (DIT) are thus required, because they indicate that the components are extensible and can be reused. Abstractness measures the number of abstract classes and has an impact on the memory footprint, as can be observed in the results for the MP3 player. For the wheelchair case study, no abstract class or interface is used. DIT measures the depth of inheritance tree, and high values for this metric lead to higher reuse. As expected, inheritance causes an overhead in memory, performance, and energy. The best solution for all of these physical aspects has the smallest DIT numbers.

Normalized Distance (Dn) is another reuse metrics, but numbers close to zero indicate a good packaging design. The best solutions for physical metrics in our experiments also show the smallest Dn values. However, the variation of Dn is too small in our experiments to consider it as an interesting correlation.

The quality of software is also evaluated using population metrics. However, there are no safe value ranges for these metrics because they depend on the size of the project. Since these population metrics also impact on the physical properties, we have also analyzed the correlation between them.

As expected, when the number of static attributes (NOSA) increases, the data memory also increases, which is confirmed by our results. Sol-3 of the MP3 player and Sol-2 of the wheelchair controller have the highest values for NOSA in comparison with the other solutions, and, consequently, these solutions are the less efficient regarding data memory.

A considerable high correlation among the number of attributes (NOA) and the performance and energy is found for both case studies. The solution with small NOA is the best one regarding performance and energy. As expected, the number of attributes impacts on the required data memory size. Sol-1 of wheelchair controller presents NOA equal to 0 and has the smallest data memory size. It is interesting to notice that Sol-3 of the MP3 player has the smallest number of attributes (NOA) but has the highest number of static attributes (NOSA). This shows that the designer of this solution decided to pay an overhead in memory footprint by the use of static attributes in order to improve the performance and energy metrics. The high correlation between the NOA and memory cannot be found in the MP3 player because of the strong correlation between NOA and NOSA. The reduction on the number of dynamic attributes (NOA) and the increase of static attributes (NOSA) lead to a better result in terms of performance and energy. This can be observed in Sol-4 of the wheelchair controller and in Sol-3 of the MP3 player.

It is known that the number of packages (NOPk) impacts the program memory size, and this has been observed in our experiments by the high correlation among these metrics and by the fact that the solutions with less NOPk present small program memories.

As expected, the number of methods (NOM) has a direct impact on the number of cycles and on the energy, as confirmed by the high correlation found among them.

The best solutions regarding performance and energy are those that have a small number of methods. However, in the OO paradigm using a small numbers of large methods is not a good practice.

Embedded software designers usually replace dynamic methods by static ones in order to reduce the overhead for method invocation. In the wheelchair controller case study, the results confirmed this statement, since the best solution in performance and energy is Sol-1, which has the highest number of static methods (NOSM). However, in the MP3 player case study, this is not found. For this case study, Sol-1 has the highest NOSM (127), but this solution is not the best one regarding performance/energy. The reason for this is that the number of methods (NOM) is strongly related to the NOSM and this solution has the highest (NOM) value (463), which causes a huge overhead in both performance and energy that was not compensated by the variation in NOSM values. Sol-3 is more efficient regarding performance and energy and has an intermediate (NOSM) value (71).

5 Conclusions and future work

We have presented an analysis of the relationship between software quality metrics and physical metrics for embedded systems. The experiments have shown that decisions on the software design phase can greatly impact on the physical properties of the final system. We have shown that it is also possible to use software quality metrics to help in design decisions in order to improve the physical properties of embedded systems. However, our experiments show that there are strong correlations between some quality metrics and, in this case, they cannot be separately analyzed.

Moreover, we have proposed the use of software quality metrics to indicate modifications that can be applied to a given modeling solution in order to obtain a better solution in terms of performance, energy, or memory footprint, with a small decrease, for instance, in code reuse. We are currently developing a tool to modify a modeling solution with respect to the quality metrics in order to find a sweet spot in the design space.

A large subset of the metrics used in this work can be measured directly on UML models. Using these metrics on UML models can help designers to early explore the solution space, looking for sweet spots without the use of a previously measured library of components directly in UML.

References

1. Aggarl, K. K. et al. Empirical Study of Object Oriented Metrics. Journal of Object Technology, v. 5, n. 8, 2006.
2. Beck, A.C. et al. CACO-PS: A General Purpose Cycle-Accurate Configurable Power Simulator. In: Proc. of Symposium on Integrated Circuits and Systems Design (SBCCI), 2003.

3. Graaf, B.; Lormans, M.; Toetenel, H. Embedded Software Engineering: the State of the Practice. IEEE Software, v. 20, n. 6, p. 61- 69, Nov. – Dec. 2003.
4. Henderson-Sellers, B. Object-Oriented Metrics, Measures of Complexity. Prentice Hall, 1996.
5. Henzinger, T.A.; Sifakis, J. The Discipline of Embedded Systems Design. IEEE Computer, v. 40, n. 10, p. 32-40, Oct. 2007.
6. Ito, S.; Carro, L.; Jacobi, R. Making Java Work for Microcontroller Applications. IEEE Design & Test of Computers, v. 18, n. 5, 2001.
7. Jerraya, A.A. et al. Embedded Software for SoC. Kluwer Academic Publishers, 2003.
8. Martin, R. Agile Software Development, Principles, Patterns and Practices. Prentice Hall, 2002.
9. Metrics. Eclipse Plug-in Available at: http://metrics.sourceforge.net/
10. Sommerville, I. Software Engineering, 7th ed. Pearson, 2004.
11. Xenos, M. et al. Object-oriented Metrics - A Survey. In: Proc. of the Federation of European Software Measurement Associations (FESMA), 2000.
12. Zhang, W.; Jarzabek, S. Reuse without Compromising Performance: Industrial Experience from RPG Software Product Line for Mobile Devices. In: LNCS, n. 3714, 2005.