Manage the automotive embedded software development cost by using a Functional Size Measurement Method (COSMIC)

A. Sophie Stern[1], B. Olivier Guetta[2]

1, 2: RENAULT S.A.S., 1 avenue du Golf, 78 288 Guyancourt cedex, France

Abstract:

Year after year, more and more cars' functionalities are performed by software: electrical vehicle, multimedia, connectivity with the outside world and so on. As its software development costs are increasing, Renault decided to develop metrics and an estimation process in order to be able to predict its software costs early in the vehicle or power-train project. At the same time, Renault is working with its major Electronic Control Units suppliers to contract with them on the basis of software metrics. After different studies, Renault chose the COSMIC method as its embedded software metric. COSMIC is for COmmon Software Measurement International Consortium, and is also the name of a functional size measurement method, ISO standard since 2003.

Keywords: Software metrics, Software functional size, Software development workload estimation process, Productivity models, COSMIC.

1. Introduction

Year after year, more and more cars' functionalities are performed by software: electrical vehicle, multimedia, connectivity with the outside world and so on. Software development costs are more and more important in the whole project development costs. Depending on the Electronic Control Units (ECUs), Renault subcontracts their development partly or entirely. But if Renault is very used to estimate accurately the development cost of physical parts such as harness or electronic components, it was quite lacking in for software development cost estimation. So the Renault embedded software group was asked to develop a software workload estimation process that had to be usable for all suppliers.

The main goal of this presentation is to give a feedback on the Renault practical experimentations with the COSMIC method on its real-time embedded software. We will first explain productivity models and our choice of the COSMIC method to measure the software size. Then we will present the COSMIC method and how we tested it on industrial cases. Along the way, we will give advices stemming from our own experience. Each of them will be flagged by the symbol ☺.

2. The software development productivity models

2.1 Introduction

The final target of having a software development workload estimation process usable for all suppliers requires the knowledge of the relationship between a software characterization and the corresponding development workload, either for the whole software or either for a piece of software. When the relationship is clearly established, the software development workload prediction becomes possible.

2.2 What are productivity models?

Productivity models are really the key of our estimation process, they are obtained by statistics methods.

A software productivity model is a linear relation between a functional software size and the corresponding development workload.

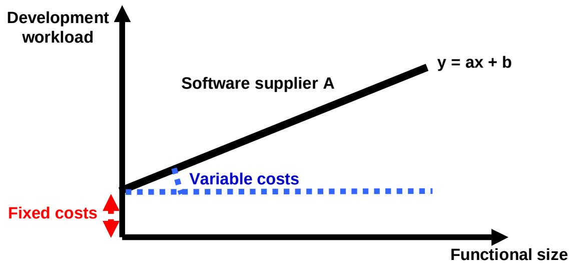
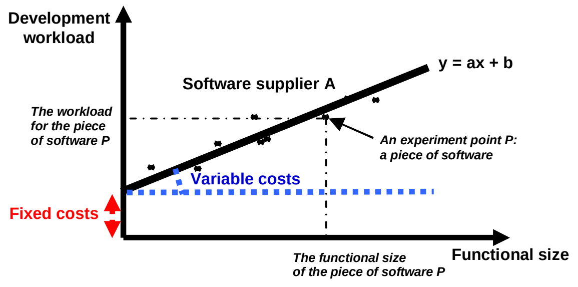


Figure 1: A software development productivity model

The software functional size is independent from the kind of implementation. The number of lines code, still used in a lot of organizations, is not pertinent, especially for embedded software. Everyone knows that for saving place, an engineer may spend a lot of time to compact its code. Furthermore, with Automatic Code Generators, the number of lines is more important but the spent time is less than with manual coding.

The linear relation between the functional size and the software development workload is obtained with data set on past projects by using linear regression methods. Each datum is representing the whole software or only a part of it, like a software module for example.



Development workload
Software supplier A
y = ax + b
The workload for the piece of software P
An experiment point P: a piece of software
Variable costs
Fixed costs
The functional size of the piece of software P
Functional size

Figure 2: The construction of a productivity model with a data set on past projects

A productivity model has to be constructed on a coherent perimeter. We explained in more details this notion in *§ 5.4 to 5.6.*

2.3 What are the uses of software productivity models?

As we said before, productivity models are the key of our estimation process. We exploit them in different uses, the major ones are:

- Estimation of development workloads (and deduction of costs, delays),
- Benchmarking of suppliers productivity,
- Managing of suppliers' annual productivity.

To be pertinent, benchmark must be performed on a coherent perimeter and with the same conditions.

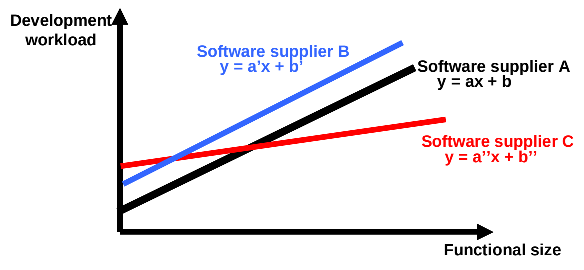


Figure 3: Comparison of software productivity models for different suppliers

Productivity models may be used as a basis of discussion with each supplier about its software development tooling for example. Notice that development productivity must be related to the quality level of the associated software.

3. The choice of the Functional Size Measurement Method

3.1 What are functional size measurement methods?

To construct a productivity model, you need to measure first the software functional size. Functional Size Measurement (FSM) methods have been used since the early eighties.

FSM methods enable to measure software independently from its implementation, they measure the software functional wealth.

Each FSM method has its own unit, Function Points (FP) for the IFPUG method, COSMIC Function Points (CFP) for the COSMIC method. The software functional size is an intrinsic metric of software, as the length in meters and centimetres for the size of a car.

Once we decided to define productivity models, the first step was to choose the FSM method.

3.2 The different FSM methods experimented

Renault experimented the COCOMO method a few years ago in the ECU diagnostic and in the Engine Control Module departments, but with unsuccessful results.
The IFPUG method has been used for several years in the Renault Information System department.
The possible application of IFPUG on embedded software had to be checked with an evaluation.
Furthermore, we decided to experiment the COSMIC method as it was announced to be well adapted to real-time embedded software.

Our first experiments started on the Engine Control Module (ECM) in 2008 with the IFPUG and the COSMIC methods. The ECM is modular and each of its modules is a set of model-based specifications. The effort supplier invoice is available for each module.
Functional size measurements were realized on the same modules with the two studied methods, and then results were compared.
In our experimentation, the COSMIC method suited well for embedded software whereas the IFPUG method appeared not pertinent especially when the software functional size was increasing.

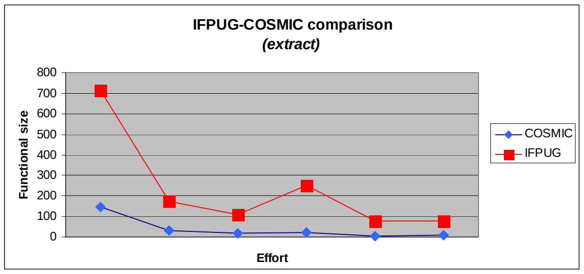


Figure 4: Part of the Renault comparison between the IFPUG and the COSMIC FSM methods

Furthermore, the measurements with the COSMIC method seemed more repeatable and faster than the ones with the IFPUG method. Last but not least, as we are more and more involved in the model based design process for our ECUs, it appeared to us that COSMIC could be automated in a further step on models realized with a simulation tool.

So we decided to pursue the experimentation for embedded software with the COSMIC method on several types of ECU and for different suppliers. Our results have been very encouraging since the beginning. Let's see now more deeply how we applied COSMIC and the productivity models on embedded software.

4. The COSMIC application on embedded software

The COSMIC method is an ISO standard (19761, 2003). The whole COSMIC documentation is free and available on Internet, cf [3].

The software functional size measurement according to the COSMIC method is based on data movements measurement: either through the boundaries between functional users and software, either data movements forwards or from memory device. So data movements are: entries, exits, reads and writes.

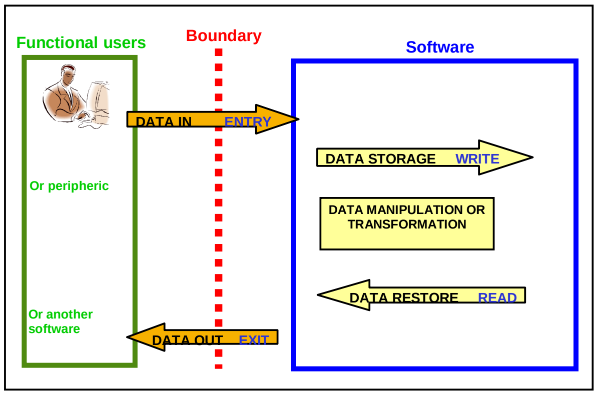


Figure 5: The mapping of COSMIC concepts on a view of software functional requirements

All COSMIC concepts are detailed in [2].

In this chapter, we will explain with some highlights how we applied the COSMIC method on embedded software with an industrial example: our study on the Engine Control Module (ECM).

The COSMIC method is composed of three phases:

- The measurement strategy phase.
- The mapping phase.
- The measurement phase.

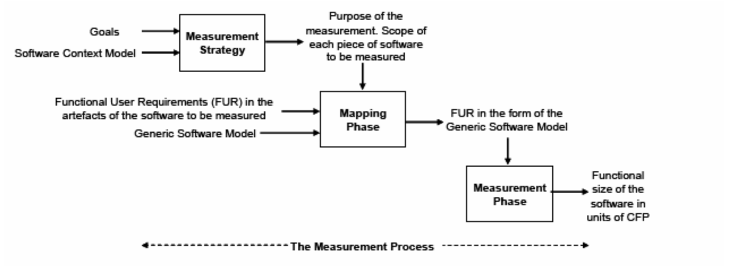

Goals
Software Context Model
Measurement Strategy
Purpose of the measurement. Scope of each piece of software to be measured
Functional User Requirements (FUR) in the artefacts of the software to be measured
Generic Software Model
Mapping Phase
FUR in the form of the Generic Software Model
Measurement Phase
Functional size of the software in units of CFP
The Measurement Process

Figure 6: The measurement process in the COSMIC measurement manual ([2])

4.1 The COSMIC measurement strategy

☺*This phase has to be realized with the customers of the productivity models.*

4.1.1 Determine the purpose of the measurement

The purpose of the measurement was defined with the purchasing and the ECM development departments.

The purpose was multiple:

1. To be able to estimate the function software development cost as soon as its specification is written to decide to implement or not the function.
2. To predict the cost of the software functions development in order to negotiate if necessary with suppliers.
3. To benchmark and manage the productivity of our suppliers.

☺*The purpose of measurement may be different from one ECU to the other one.*

4.1.2 Determine the scope of the measurement

The ECM is developed schematically depending on the following architecture: hardware, basic software, specific software depending on the platform, portable software independent from the platform.

The measurement scope in 2009 included all suppliers developing portable software, since 2010 it also includes specific software developed by major suppliers.

☺*It is important for you to know that it is possible to start working only on some pieces of software and not on the whole software.*

☺ *In a general way, we decided to restreint our COSMIC studies to ECUs which are model based.*

4.1.3 Determine the functional users

For the ECM, functional users are interfacing peer software, in other words, when we consider one module, functional users are the others modules in contact with it.

4.1.4 Determine the level of granularity

The choice of the granularity level is very important, it may be very different from one ECU to the other one. For the ECM, we chose the granularity level of the software modules. There are two reasons for this choice, the first one is that it is already the level of costs negotiation with suppliers, the second one is that this choice enables to have quite a well-dimensioned data set on past projects for constructing productivity models.

4.2 The COSMIC mapping phase

This second phase includes the identification of functional processes, data groups and data attributes.

4.2.1 Identify the functional processes

A model-based specification is composed of blocks and data flows that the COSMIC method interprets as functional processes and data movements, or not.

It is important to understand that the COSMIC method doesn't take into account the specification's architecture. The important point is to identify parts of specification which are not only structure but which are active parts.

4.2.2 Identify data groups

We measure variables and calibrations as data groups with the COSMIC method.

4.2.3 Identify data attributes

☺*At the present time, we do not take into account this step of the COSMIC method in our mapping.*

4.3 The COSMIC measurement phase

4.3.1 Identify data movements

This is the identification of data which are going through the boundaries between functional users and functional processes (Entries or Exits), and the identification of data read from or written in the memory (Readings and Writings).

Once you identified functional processes and functional users, you are able to identify data movements.

4.3.2 Apply measurement function

When data movements have been identified, you just have to count one COSMIC Function Point (or 1 CFP) for each data movement, whatever its nature.

4.3.3 Aggregate measurement results

The last step is to aggregate the COSMIC Function Points on the measurement.

☺*For the ECM, a module is described by several specifications files. We measure each specification* *and aggregate the results in each specification and then for the whole module.*

4.4 Software enhancements COSMIC measurements

4.4.1 Introduction

With the arrival of ECU's architecture standards as AUTOSAR, there is now the possibility of developing the same code for different hardware targets. In this context, more and more software is reused from one ECU to another one. So beside new software developments there are more and more software enhancements.

The functional size measurement process is different for a new development or for an enhancement.

4.4.2 Highlight points for enhancements measurements

A software enhancement functional size measurement is obtained by comparing the two specifications releases, what we called release N and release N-1. The COSMIC method measures differences between the enhanced specification and the previous one.

When modifications affect functional processes, data movements or elementary blocks in the release N, they are caught by COSMIC measures.

There are three types of modifications:

- Addings,
- Modifications,
- Removals.

When an element is added, we measure its size.

When an element is deleted, we measure its size before its deletion.

When an element is modified, we measure the size of the modification.

Elements modifications caught by the COSMIC method are for example a variable renaming, a modification of a variable size.

The three types of enhancements are measured, and functional sizes are added at the end.

5. Industrial examples and key points

5.1 Obtained results

We already studied and obtained results on several projects, on several kinds of ECUs, for new developments and for enhancements.

For example, we have COSMIC results on:

- The Body Control Module, or BCM.
- The Engine Control Module, or ECM.

That means we realized several productivity models for each ECU and for different suppliers.

We are studying other types of software.

5.2 An example of cost prediction

We applied our BCM COSMIC new developments productivity models during a Request For Quotation with suppliers for a new BCM in order to have a target software development cost on the applicative software. As soon as the specifications were written, we predicted the development effort for each BCM function within a prediction interval, we also predicted the development effort and the associated prediction interval for the whole applicative software. At the same time, the suppliers realized their own estimation each of them with their *home made* method. Then we compared their estimations and ours before negotiating.

The graph below shows the comparison between our COSMIC predictions and the estimations of one supplier after the first technical and economical round and after the second round of the RFQ.

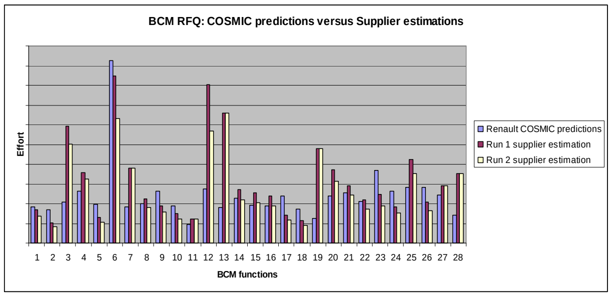


Figure 7: COSMIC use for costs predictions and costs negotiations

We discussed with suppliers on software modules for which their estimations were very far from our predictions based on COSMIC and on our productivity model.

☺ *Having factual data on software development workloads was a very good basis of discussion and a good lever of negotiation.*

5.3 COSMIC measurement's Return On Investment and Model Based Design

Whatever the reason why you are realizing COSMIC measurements, one day or another you probably have to check the Return On Investment (ROI) of your software measurement.

So you will have to manage and to limit the cost of your COSMIC measurements. The best way to reduce this cost is having the more automated measurement process as possible. Even if the measurement is performed by humans, the more standard the specifications are, the less interpretable they are, the easiest and the quickest the measurement will be. If specifications are written in natural language, with different formalisms, with residual errors, the measurers will have to be experimented people as functional measurement experts.

The COSMIC method may be applied on non model based specifications, for example in natural language, but there are several brakes to realize very performing functional size measurements in these conditions.

The COSMIC application is really favoured by Model Based Design (MBD) because specifications are already consistent for the measurer, the formalism is always the same and it is easy to train new measurers who are not necessary FSM experts. Furthermore, automation of COSMIC measurements on MDB specifications seems possible.

5.5 Productivity models and outliers

Remember, productivity models are realized on data set on past projects with statistical methods as linear regression. A productivity model is defined for a perimeter, at least for the couple ECU and supplier.

By trying to realize a correlation on data set for a coherent perimeter, there may be some points outside the regression curve, these points are called outliers. There are two major reasons of existing outliers: either the outlier corresponds to software intrinsically different from the other ones of the data set, either there are other influent factors than the functional size. Influent factors may be: restricted delay to develop the software, high experienced people, and so on. As you can see, influent factors may reduce or increase the development workload.

Either outliers will go in another productivity model, either they will be stored in a outliers checklist: let's see on a real industrial case.

You can see below the different steps to detect outliers and to take them into account.

On this first figure, you can see the whole data set on past projects for BCM for one supplier named X.

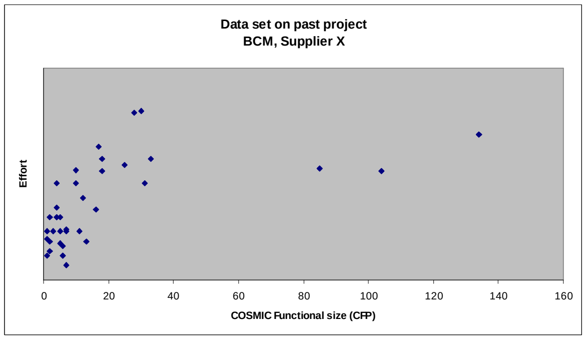


Figure 8: The data set on past projects for BCM

Three points are really outside the curve, they are materialized on the figure below with a big square.

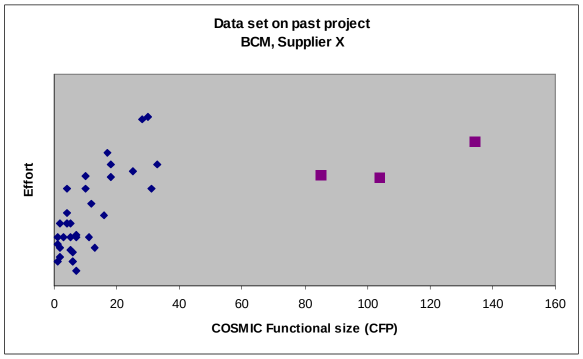


Figure 9: Materialization of the outliers

☺ *Before removing the outliers, you must absolutely find the explanation for each of them otherwise your estimation process will never be strong. The ECU development team has probably this explanation.*

In our case, one point was corresponding to an automatically coded software, and that is the reason why we began to split productivity models in function of the coding type, manual or automatic.

The two other points had been coded manually by an expert of this software, so we put these points in the outliers checklist.

After removing these three outliers, the data set is the following.

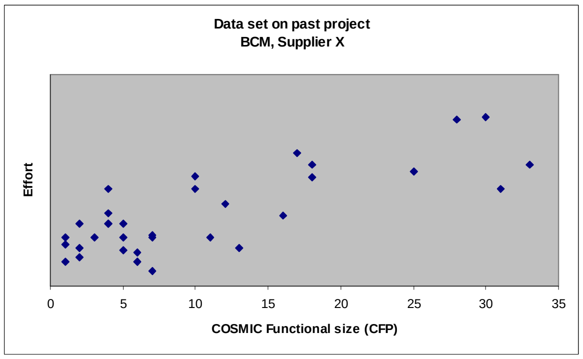


Figure 10: The BCM data set on past projects, outliers removed

Then we realized the linear regression on the corrected data set and the correlation factor or R^2, 0.61, was quite good.

☺ *R2 seems to be better when software functional sizes are larger.*

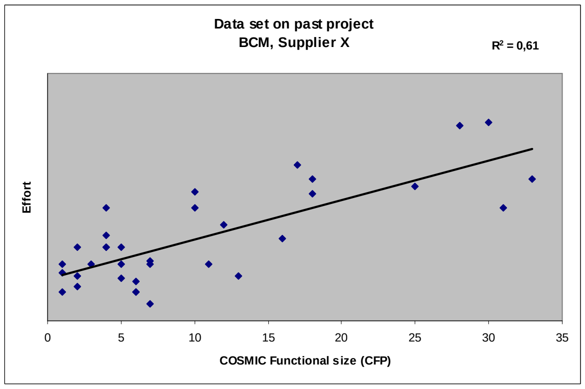


Figure 11: Linear regression on BCM

5.6 Productivity models split, influent factors

There is not one software development productivity model for all ECUs, all suppliers and all situations.

The influent factors we have already seen are: the ECU's type, the supplier, manual or automatic coding, new developments or enhancements. You can treat influent factors by separating models in several models or consider multi-factors approaches. For the time being, we have preferred to separate models because it is easier to understand and to explain simple linear regression models.

☺*Do not confuse software functional size which is a software intrinsic metric and productivity models. For example, the software functional size is the same either the software is manually coded or automatically generated, but there will be two productivity models on one perimeter with different slopes and ordinates at the origin according to the coding type.*

☺ *People often have presupposed ideas on influent factors and on the best productivity models split. For example, before realizing the ECM productivity model, the development team thought there will be several productivity models: one for interfaces modules and one for algorithmic modules. Our principle was, let's try to put all modules on one model and then we will see.*

You can see below the data set on past projects for one supplier of the ECM.

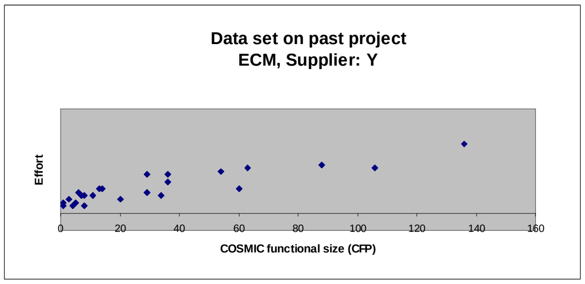


Figure 12: Data set on past projects for ECM and for one given supplier

The result is that the correlation with all the types of modules is good: correlation factor or R^2 about 0.81. You can see below the productivity model after the linear regression.

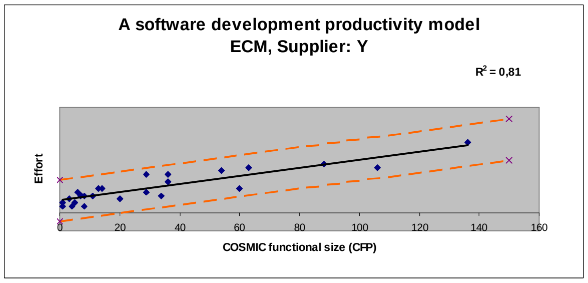


Figure 13: A ECM productivity model for one supplier

You can see that all points are in the confidence interval. The confidence interval is a factual boundary to detect which points are outliers.

5.7 Contracts with suppliers based on productivity models

Large organizations are already used to establish contracts with their software companies based on the fixed price of a Function Point with the IFPUG method.

We found productivity models more interesting than the Function Point fixed price because they take into account the difference between large software and small ones. There is a scale factor. Nevertheless, the habit of contracting on metrics with a supplier is a very mature way of working and we can *copy paste* this way from Information Systems to Embedded Software, two different worlds but with interesting similarities.

So we decided to share with several Renault major suppliers the COSMIC method mapping and their productivity model on some ECUs.

5.8 Where using COSMIC, inside our outside?

FSM methods as the COSMIC method is, productivity models, can be applied inside or outside the company, depending if the software development is totally or partially outsourced.

5.9 Possible organizations for software workload estimation process

To construct a complete and strong software workload estimation process, it is a necessity to have in the organization a core measurement team with different skills, especially: informatics and statistics.

Once the process is constructed and strong, there are several types of possible organizations for functional size measurers and productivity models specialists. Either there is only a core team, either the measurement is completely disseminated, either there is a merge of these two ways.

Automation may have a role in disseminating the COSMIC measurement task in the whole organization.

6. Cooperation with the embedded real-time software community

Our Embedded Software group has written rules to map all the COSMIC concepts on simulation tools used by the ECUs' development Renault teams.

Our first objective is to have the same rules for all our suppliers.

Our final goal is to have in the real-time embedded software community shared rules for the mapping of COSMIC.

☺ *Renault is co-managing with the Ecole de Technologie Supérieure (ETS) of Québec the production of the official COSMIC guideline for sizing real-time system software. Its Web publication is planned for the end of 2010, cf [3].*

☺ *Renault is also involved in the French organization CG2E (Club des Grandes Entreprises de l'Embarqué), in the working group "Productivity of the development chain", and sponsors the COSMIC method in order to have the feedback of the community.*

7. Conclusion

When we began working on COSMIC in 2008, our first objective was to put in place a software development workload estimation process. With our results, we are confident and we are pursuing on the same track.
What we have found along the way is: if COSMIC is a very good metric to help us to predict workloads of software projects, COSMIC might be also our software reference metric and the basis for a lot of new uses.

8. Acknowledgement

We want to acknowledge here the contribution of Professor Alain Abran, teacher at the ETS, Montréal, Québec, who is one of the COSMIC father and who is always there to answer to our questions.

9. References

[1] Sophie Stern: "*Practical experimentations with the COSMIC method in Automotive embedded software field*", IWSM, Amsterdam, Netherlands, 2009.

[2] COSMIC: Common Software Measurement International Consortium, *The COSMIC Functional Size Measurement Method Version 3.0.1, Measurement manual (The COSMIC implementation Guide for ISO/IEC 19761),* 2003.

[3] COSMICON: http://www.cosmicon.com/

10. Glossary

BCM:	Body Control Module
CFP:	COSMIC Function Point
COSMIC:	Common Software Measurement International Consortium
ECM:	Engine Control Module
FSM method:	Functional Size Measurement method
IFPUG:	International Function Points Users Group
MDB:	Model-Based Design
ROI:	Return On Investment