Function Point Analysis with Model Driven Architecture Applied on Frameworks of Partial Code Generation

Rodrigo Salvador Monteiro
Instituto de Computação
Universidade Federal Fluminense, UFF
Niterói, Brasil
e-mail: salvador@ic.uff.br

Roque Pinel, Geraldo Zimbrão
and Jano Moreira de Souza
COPPE / UFRJ
Rio de Janeiro, Brasil
e-mail: {repinel,zimbrao,jano}@cos.ufrj.br

Abstract— **Software measurement is a crucial task for the planning and the developing of information systems. The Function Point Analysis (FPA) was developed to measure the complexity of the functionality of systems. Its methods are independent of technology and can be applied directly to the specification of features and the domain. However, the counting should be performed by a metrics analyst, being under subjectivity, wasting time and a large number of resources. This article describes the proposal of automation of function point counting performed using Unified Modeling Language (UML) models and Model Driven Architecture (MDA) methodology. Our approach provides a standard method for counting based on the International Function Point Users Group (IFPUG), eliminating the subjectivity present in traditional procedures. The work counts the number of realized function points, based on the information system already developed. The counting of function points achieved allows for transparency to the client receiving the product besides the construction of an important historical base for the refinement of future estimates.**

Keywords-MDA; Metric; Function Point Analysis; AndroMDA; MDArte.

I. INTRODUCTION

With the increasing complexity of information systems, measuring their features is a crucial task for software projects. Estimation reports become common documents of the customer relationship, essential to development planning and organizing tasks. The Function Point Analysis (FPA) was defined in 1979 as a procedure capable of measuring the functionality and complexity of information systems [1]. It is performed based on the specifications of features and the domain, defined as independent of technology, unlike other metrics, such as Lines of Code (LOC), that depends on the programming language used. The method of function points was developed in order to deliver the customer a measure on the logic functions in the system, based on specifications. Therefore, the metrics analyst should study the documentation and count the number of points. Despite the efforts of the International Function Point Users Group (IFPUG) [2] to establish a standard for counting, this value is still under the subjectivity of the analyst. Moreover, this process is known to require hours of hard work and dedication, being a large consumer of resources.

Aligned with the independence of technology, we have the Unified Modeling Language (UML) [3], which allows projects to have a standard graphical representation. It has been widely used in information systems specifications, being the main source for the establishment of the Function Point Analysis. However, although it is helpful, the analysis of documents continues to be cumbersome and time-consuming.

In 2001, the Object Management Group (OMG) released a guide of definitions about code generation based on models, the Model Driven Architecture (MDA) [4]. This methodology uses, among other standards, UML as a modeling language. Its methods allow the automation of the life cycle of projects based on UML models, reducing development time and allowing the standardization of the system code. The MDA approach provides an environment ripe for introduction of automatic FPA.

In particular, we explore the use of the framework MDArte [5], an extension of the AndroMDA framework [6], to automate the process, extract values from models and generate artifacts useful in the FPA, such as the classification of elements of the information system generated, e.g., Entities, Services and Use Cases. This choice was based on the maturity of the tool and the number of information systems that are in production and use [7]. In addition to automation, its use allows access to systems that can benefit directly from this proposal. Thus, we aim to count the number of function points from what was already realized, i.e., based on functionalities already developed. Providing an automated procedure for counting the realized effort aims at delivering transparency to the client receiving the product. This way, the effort effectively realized can be confronted with the initial estimates. Moreover, although the counting is applied on developed functionalities, the results produced can be used to analyze and understand the estimation errors. This knowledge must be used in order to improve the accuracy of the estimations of complexity for functionalities still under planning.

This paper is organized as follow. Section 2 presents some related works and our approach to the problem. Section 3 describes the concepts of Function Point Analysis. Section 4 explains in detail the proposed automatic counting. Section 5 discusses the prototype developed. Section 6 explores the Case Study. Finally, Section 7 concludes the paper.

 ISBN: 978-1-61208-458-9

II. RELATED WORK

Automation of FPA has been discussed for some time. The works [8] and [9] approach the process based on UML models, using Class Diagrams, Sequence Diagrams and Use Cases as inputs. In general, the models are read and interpreted by the application responsible for the counting, with some exceptions in which the user must interact with the application. This is because, despite the fact that the UML and the FPA are independent of technology, in general, the UML models do not have all the information necessary for the counting.

In order to aggregate automatic counting to models that are actually used in the development, i.e., that are synchronized with the current stage of information system, the work [10] described the process in an MDA framework. The main difference between the work in [10] and that is [8] and [9], is that it explores the fact that the system code and the counting are processed by the same tool, the code generation framework, not requiring any additional effort.

However, even the models used for code generation following the MDA approach may not be sufficient for the automation of the FPA. When working with a specific group of MDA frameworks that apply the partial generation of implementation code [11], the model does not fully represent the business rules of the information system. In this type of development, a portion of the system is implemented manually by the developer. Thus, the code will contain information relevant to the counting, e.g., entities that are changed by the system or not.

In our work, we chose a different approach to the problem. We use the MDA to extract information from models, ensuring its accuracy, and a tool to extract information from the system code. Since the information was only extracted from the code, if the technology is changed, it is necessary to change only the extractor.

By using not only models but part of the code, our work and [10] count the number of function points from what was already realized, i.e., based on information systems already developed. Although the counting is applied on a developed project, the result produced can be used to create a historical base and to adjust and improve accuracy of the FPA rules application. Further, the proposal will be described and exemplified with the Case Study.

III. FUNCTION POINT ANALYSIS

The FPA measures the functionality provided by a single information system [2]. It is a recognized ISO (International Organization for Standardization) standard for measuring software and can be determined from the requirements specification, considered as independent of technology. As proposed in [1], it counts the following system characteristics: files used by the system, external inputs and outputs, user interactions and interfaces. Each feature is considered individually and counted as the weights assigned.

The version proposed by IFPUG FPA, used in this work, provides some modifications to the original rules. It is described in seven steps [2].

1) Determine the type of function point counting.
2) Identify the system boundary.
3) Count the Data Functions.
4) Count the Transaction Functions.
5) Determine the value of unadjusted function points.
6) Determine the adjustment factor.
7) Calculate the adjusted value.

In our work, the type of count used (step 1) will be Development Project. It measures the functions provided to the user with the first installation of the system being delivered. Our work follows steps 2 to 5. Steps 6 and 7 do not fall within the scope of this work, as they use system specific features that must be manually adjusted.

The next two subsections describe the two function types related to the steps 3 and 4, respectively: data function and transaction function.

A. Data Function

The Data Functions are functions that deal with stored data. They are classified as *Internal Logical File* (ILF) or *External Interface File* (EIF). ILFs are related to data that are created or maintained by the system, while EIFs deal with external data.

B. Transaction Function

The Transaction Functions are functions that interact with some user or with external agents. They are classified as *External Input* (EI), *External Output* (EO) or *External Inquiry* (EQ).

a) EI: controls information or processes data. Its main objective is to keep one or more ILFs or to change the system behavior.

b) EO: sends data or controls information outside the system boundary. Its main objective is to provide information to the user, as in reports. They should contain some processing, for example, mathematical formulas, maintain an ILF or alter the system behavior.

c) EQ: sends data or controls information outside the system boundary. Its main objective is to retrieve information from data items. Unlike EO, they should not contain processing, maintaining an ILF or alter the system behavior.

Each Transaction Function also has a number of *Data Element Types* (DETs), the smallest meaningful data items presented (or requested) to (by) the user. Beside DETs, each Transaction Function has a number of *File Type References* (FTRs), the number of Data Functions accessed by the Transaction Function.

IV. PROPOSAL

Although FPA is independent of technology, when performed without the use of models, the automation of counting proposed becomes specific of technology. As an example, we have [12] where only the COBOL code is considered during analysis.

Figure 1 illustrates the scheme proposed [14], where the system code, represented by the points of implementation, and the artifacts with the characteristics of the system are generated from UML models. Particularly, Figure 1 shows an example of the MDA methodology using partial generation of implementation code. These points are the spots that actually contain the business rules of the generated systems.

 ISBN: 978-1-61208-458-9

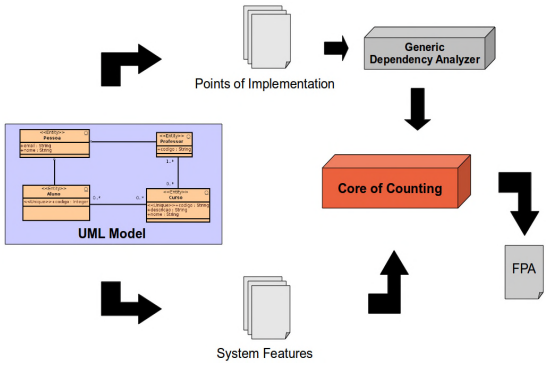

Figure 1. Schema of the proposal.

In our proposal [14], the system code is also used. However, only the dependence between the elements is considered, e.g., a Service that handles Entities. Thus, the points of implementation must pass through a dependency analysis before being used in the count. This allows the decoupling between the language used and the Core of Counting component.

Together with the dependencies, the system features are also used by the Core. Such features represent information that can be extracted from models, to classify elements as, e.g., Entities or Use Cases. Provided with this information, the Core can extract the dependencies that actually have value to count and identify the type of function: data or transaction. Considering the need to recognize and correctly count the types of functions, we developed conditions similar to those described in [8] and adapted for Web systems.

A. Counting Data Functions

As each Data Function has a number of DETs and RETs, they are counted as follows:

a) DET: defined as the number of attributes of the entity plus the number of inherited attributes, recursively, disregarding the identifying attributes.

b) RET: assumed to be 1 (one), since this value is used in most situations and has achieved good results [8].

TABLE I. DATA FUNCTIONS – UNADJUSTED VALUE [2]

RET	Data Element Type (DET)			Complexity	ILF	EIF
	1 ~ 19	*20 ~ 50*	*> 50*	***Low***	7	5
0 ~ 1	Low	Low	Average	***Average***	10	7
2 ~ 5	Low	Average	High	***High***	15	10
> 5	Average	High	High			

Through the combination of the number of DETs and RETs, it is possible to assign a complexity to the Data Function using the left side of Table 1. To each complexity is assigned a value of unadjusted function points, as shown on the right side of Table 1.

B. Counting Transaction Functions

As each Transaction Function has a number of DETs and FTRs. They are counted as follows:

a) DET: for an EI, it represents the number of arguments of the transaction. For an EO, it represents the number of output parameters. Finally, for an EQ, it represents the number of arguments of the transaction plus the number of output parameters.

b) FTR: analogous to the number of RET for Data Functions, it is assumed to be 1 (one) due the achievement of good results [8].

TABLE II. TRANSACTION FUNCTION – COMPLEXITY [2]

FTR	EI Data Element Type (DET)			EO and EQ Data Element Type (DET)		
	1 ~ 4	*5 ~ 15*	*> 15*	*1 ~ 5*	*6 ~ 19*	*> 19*
0 ~ 1	Low	Low	Average	Low	Low	Average
2 ~ 3	Low	Average	High	Low	Average	High
> 3	Average	High	High	Average	High	High

TABLE III. TRANSACTION FUNCTION – UNADJUSTED VALUE [2]

Complexity	EI	EO	EQ
Low	3	4	3
Average	4	5	4
High	6	7	6

Similar to the complexity assigned to Data Functions, the complexity of the Transaction Functions is realized based on the values of DET and FTR. Table 2 is used to determine the complexity of EIs, EOs and EQs. Then, the unadjusted value of function points can be obtained from Table 3, for all three types of Transaction Functions.

V. PROTOTYPE

In this section, we describe how the information necessary to perform the automatic counting of function points is obtained and processed. In order to achieve this goal, we developed a Prototype to demonstrate in practice how the automatic counting of function points is made. Figure 2 shows its operating model.

According to the model, the process begins with framework MDArte [5] to generate the artifacts used as input for the Prototype. The MDArte is a tool that receives UML models and generates the corresponding codes. It is one example of MDA framework with partial generation of implementation code, in which business rules should be described directly in the code at specific locations called points of implementation.

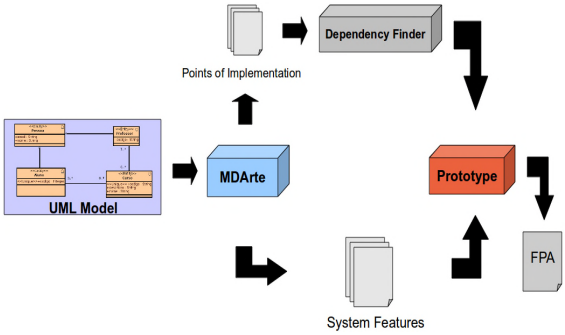

Figure 2. Schema of the prototype.

In our example, we use specific MDArte cartridges to generate information systems written in Java. Thus, we can add the tool Dependency Finder [13] to the proposed model. The Dependency Finder is capable of analyzing compiled Java code and extracting the dependency list of the elements. One of the benefits of its use is the generation of the list of dependencies in XML, as shown in Figure 3, following a pattern easily reproducible, which could be generated by other tools to analyze other programming languages. Thus, the Prototype does not depend directly on a technology, being flexible to deal with other cartridges generation, or even systems coded manually.

```
<?xml version="1.0" encoding="utf-8" ?>

<dependencies>
 <package confirmed="yes">
  <name>br.ufrj.coppe.system</name>
  <class confirmed="yes">
   <name>br.ufrj.coppe.system.Student</name>
   <outbound type="class" confirmed="yes">br.ufrj.coppe.system.Person</outbound>
   <feature confirmed="yes">
    <name>br.ufrj.coppe.system.Student.Student()</name>
    <outbound type="feature" confirmed="yes">br.ufrj.coppe.system.Person.Person()</outbound>
   </feature>
   <feature confirmed="no">
    <name>br.ufrj.coppe.system.Student.getName()</name>
     <outbound type="class" confirmed="no">java.lang.String</outbound>
   </feature>
  </class>
 </package>
</dependencies>
```

Figure 3. Example of XML produced by the Dependency Finder.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<statistics>
 <application>AcademicSystem</application>
 <type>entities</type>
 <entities>
  <entity>
   <name>br.ufrj.coppe.system.Person</name>
   <attributes>
    <size>1</size>
    <attribute identifier="false">
     <name>name</name>
     <type>java.lang.String</type>
    </attribute>
   </attributes>
   <methods>
    <size>1</size>
    <method modifier="false">
     <name>getName</name>
     <return>
      <type>java.lang.String</type>
     </return>
     <parameters>
      <size>0</size>
     </parameters>
    </method>
   </methods>
  </entity>
  <entity>
   <name>br.ufrj.coppe.system.Student</name>
   <extends>br.ufrj.coppe.system.Person</extends>
```

Figure 4. Example of XML generated by MDArte with system features.

Although the list of dependencies has an important role in the process, it is not enough for the FPA to be performed. As seen in Figure 3, the XML produced does not allow for the classification of elements. So, we use the characteristics of the information system as auxiliary entry, illustrated by the XML from Figure 4. This XML excerpt describes some characteristics of Entity elements.

From Figure 4, we can see that "Person" is an entity and has a "name" attribute. You may also notice that the entity "Student" inherits information from "Person", which explains the presence of method “getName” in Figure 3, unconfirmed, since it belongs to "Person" and not to "Student". The information from both types of entries is related and processed by the Prototype and used to identify and count the two types of functions: data and transaction.

The Prototype identifies the Data Functions checking which Entities are inside (ILF) or outside (EIF) of the boundary of the information system. To do so, it verifies which Entities have their set methods accessed. This is done through the attribute "modifier" tag "method", present in the XML with the characteristics of the system as shown in Figure 4. This attribute indicates methods that can be used to change an Entity, which allows its search on the list of dependencies.

After having classified the Data Function as ILF or EIF, the process of counting the number of DETs and the number of RETs begins. Both values are calculated as described in the previous section. The number of DETs is defined by the number of attributes of an entity, considering the inherited attributes and disregarding the identifying attributes. The number of RETs has been assumed to be 1 (one), as stated in the proposal.

The Transaction Functions are identified and counted according to the proposed rules. However, the concepts have been adapted to Use Cases, particularly the Activity Diagrams that describe their flow. Figure 5 illustrates how the Activity Diagrams used by the MDArte are modeled. The activities with the stereotype *<<FrontEndView>>* represent screens and the values associated to the outgoing transitions its parameters.

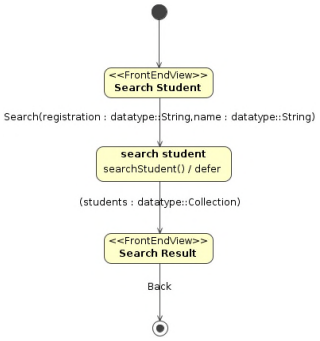

Figure 5. Example of Activity Diagram read by the MDArte.

The identification process is represented by the flowchart in Figure 6. As described in the flow, the functions are classified into EI or EQ. This flowchart represents the first stage of the process, remaining to deal with the EOs.

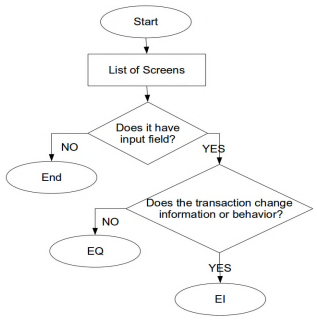

Figure 6. Flowchart of Transaction Functions identification process. First Stage.

 ISBN: 978-1-61208-458-9

After having identified the EIs and the EQs, the process continues with the second stage described by the flowchart in Figure 7. Now, the screens not yet identified are checked looking for a complementary screen of an EQ, e.g., the screen that shows the results of a query. If it is not the case, then the screen is classified as EO.

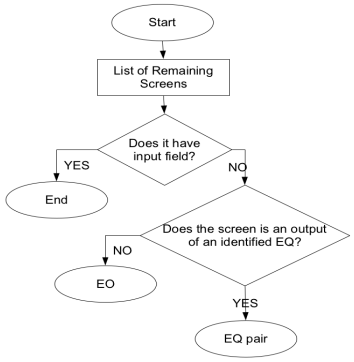

Figure 7. Flowchart of Transaction Functions identification process. Second Stage.

Next, we describe how the number of DETs for each type of Transaction Function is counted based on information provided to the Prototype. Remembering that the number of FTRs has been assumed to be 1 (one), as stated in the proposal.

a) External Input (EI): its number of DETs is calculated by adding the number of input attributes, present on the screen, including buttons.

b) External Output (EO): its number of DETs is calculated by adding the number of attributes in the results screen. Emphasizing that when displaying information as table, only the columns are counted, not lines.

c) External Inquiry (EQ): its number of DETs is calculated by adding the number of the input attributes, as for EIs, and the number of output attributes, as for EOs.

Having counted the number of DETs and RETs for Data Functions, and the number of DETs and FTRs for Transaction Functions, the Prototype performs the assignment of complexity for each function. Afterwards, it also determines the value of unadjusted function points based on Tables 1 to 3.

VI. CASE STUDY

The Case Study was prepared following the proposal described in this paper as well as the rules used by the Prototype. Its goal is to demonstrate how automatic FPA is made for real examples.

Therefore, we chose an information system of an academic environment as example, limited to a few Entities and Use Cases for better understanding. Thus, our example is only responsible for keeping the information of *Students* and allows the *User* to change some of its information, like the *password*.

First, we analyze the Data Functions and then the Transaction Functions.

A. Data Functions

Figure 8 illustrates the class diagram of the Case Study. You may notice the five Entities, separated into two symbolic groups: *academic system* and *access control*.

a) *Person*: person information.

b) *Student*: student information.

c) *User*: system user information.

d) *Group*: user groups information.

e) *Action*: information of actions that can be done through the system related to the group permission.

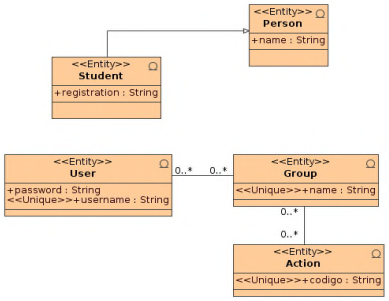

Figure 8. Example of Class Diagram read by the MDArte.

The entities *Group* and *Action* are kept outside the system boundary, as they represent a part of access control based on access groups, like profiles. Thus, as described in the previous section, the identification of the Data Functions is done by searching for methods that can modify each entity among the dependencies of system operations.

a) ILF: *Person*, *Student* and *User*.

b) EIF: *Action* and *Group*.

TABLE IV. COUNTING DATA FUNCTIONS

	DET	RET	Complexity	Value
Action	1	1	Low	5
Group	1	1	Low	5
Person	1	1	Low	7
Student	2	1	Low	7
User	2	1	Low	7
Unadjusted Total				31

After identifying each entity, the counting process of RETs and DETs starts. The number of DETs is defined as the number of attributes of the entity plus the number of inherited attributes, recursively, disregarding the identifying attributes. The number of RETs has been assumed to be 1 (one), as stated in the proposal. Applying the values in Table 1, the unadjusted values will be as defined in Table 4.

B. Transaction Functions

In our example, we will use the CRUD of the Entity *Student* to validate the proposal. The left side of Figure 9 represents a screen that allows the *User* to create a new *Student* in the system. It is an example of EI, with two parameters and one button. Its counting is based on Tables 2 and 3, and shown in Table 5.

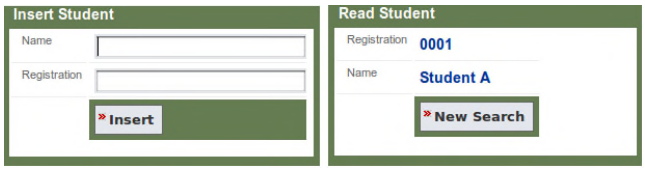

Figure 9. Screens for inserting and reading a student.

As an example of EO, we have the screen displaying the information of a Student, illustrated on the right side of Figure 9, with two result attributes. Applying the values in Tables 2 and 3, we have the counting as shown in Table 5.

The identification and counting of an EQ can be considered the most complicated of the three. We chose to use the same use case described by the Activity Diagram in Figure 5. The activity named *Student Search* is the screen shown on the left side of Figure 10, and the activity *Search Result* represents the screen on the right side. In *Student Search*, the input values are displayed, two parameters and one button. *Search Result* has four attributes, represented by two columns and two buttons, where *View* is counted only once. Based on the same tables as for EO (Table 2 and 3), we have the counting shown in Table 5.

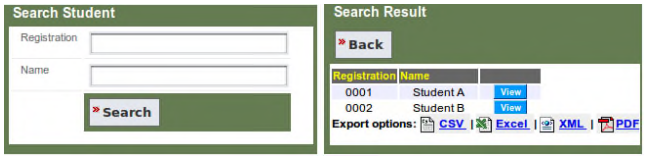

Figure 10. Screens for searching student.

TABLE V. COUNTING TRANSACTION FUNCTIONS

	DET	FTR	Complexity	Value
Insert Student	3	1	Low	3
Read Student	2	1	Low	4
Search Student	7	1	Low	3
Unadjusted Total				10

Finally, from the totals in Tables 4 and 5, *31* and *10*, respectively, we get the unadjusted total of *41* function points. The total obtained represents the complexity of the information system described by the Case Study, according to the rules established by IFPUG [2] and the proposed automation of this work.

VII. CONCLUSION AND FUTURE WORK

In this paper, we presented a proposal for automatic Function Points Analysis (FPA) following the standards presented by IFPUG. Our proposal counts the number of realized function points, based on functionalities already developed. This work is the first step aiming to answer the following question: is the complexity of what was implemented close to the estimated complexity? The answer to this question provides transparency to the client receiving the product and allows for the creation of a historical base that can be used to improve accuracy of functionalities under planning.

The proposal was evaluated through the construction of a prototype and a case study. The counting performed on the case study was accurate. We are aware that some simplification, such as assuming RET and FTR values always as 1, will generate deviations on more complex or realistic scenarios. That is exactly why our future steps are: (1) evaluate the proposal on real applications developed using the MDArte; (2) perform the counting on such applications with the assistance of a FPA specialist; (3) identify the deviations; and (4) evolve the proposal in order to gather more required information from both models and code. The belief is that the more information we can assume about the patterns and architecture of the information system developed the more accurate the automatic counting procedure will be. This led us to another important issue that will be evaluated in future research: which is the minimum set of assumptions about the system implementation in order to achieve a result with reasonable precision?

REFERENCES

[1] A. J. Albrecht, “Measuring application development productivity,” Proceedings of the Application Development Symposium, New York, USA, 1979, pp. 83-92.

[2] IFPUG, “Function point counting practices manual,” release 4.1, International Function Points Users Group, NJ, 2000.

[3] G. Booch, J. Rumbaugh, and J. Jacobson, “The unified modeling language user guide,” Addison-Wesley, MA, 1999.

[4] J. Siegel, and the OMG Staff Strategy Group, “Developing in OMG’s model driven architecture”, OMG white paper, 2001.

[5] MDArte, “Framework MDArte,” https://softwarepublico.gov.br/social/mdarte/, accessed on 23/12/2015.

[6] AndroMDA, “Framework AndroMDA,” http://www.andromda.org, accessed on 23/12/2015.

[7] R. E. A. Pinel, F. B. do Carmo, R. S. Monteiro, and G. Zimbrão, “Improving tests infrastructure through a model-based approach,” ACM SIGSOFT Software Engineering Notes. 36(1), 2011, pp. 1-5, doi: http://dx.doi.org/10.1145/1921532.1921544.

[8] T. Uemura, S. Kusumoto, and K. Inoue, “Function-point analysis using design specifications based on the unified modelling language,” Journal of Software Maintenance: Research and Practice, vol. 13(4), 2001, pp. 223-243.

[9] T. Iorio, “IFPUG Function Point analysis in a UML framework,” Proceedings of Software Measurement European Forum, 2004.

[10] P. Fraternali, M. Tisi, and A. Bongio, “Automating Function Point Analysis with Model Driven Development,” Proceedings of CASCON, 2006, doi: http://dx.doi.org/10.1145/1188966.1188990.

[11] R. Soley, and the OMG Staff Strategy Group 2000, “Model driven architecture,” OMG white paper, 2000.

[12] V. T. Ho, and A. Abran, “A Framework for Automatic Function Point Counting From Source Code,” International Workshop on Software Measurement (IWSM), 1999.

[13] Dependency Finder, http://depfind.sourceforge.net, accessed on 23/12/2015

[14] R. E. A. Pinel, “Análise de pontos de função em sistemas desenvolvidos usando MDA,” COPPE, Master thesis, 2012