A System for Measuring Function Points from Specifications

Technical Report DEIS-LIA-97-006

Fausto Gramantieri[1], Evelina Lamma[1], Paola Mello[2], Fabrizio Riguzzi[1]

[1]DEIS - Università di Bologna
Viale Risorgimento, 2 40136 Bologna
{elamma,friguzzi}@deis.unibo.it

[2]Dipartimento di Ingegneria, Università di Ferrara
Via Saragat, 1 44100 Ferrara
pmello@ing.unife.it

Abstract: We propose a knowledge based approach for the automated measurement of the Function Point metric starting from the specifications of a software system expressed in the form of an Entity Relationship (ER) diagram plus a Data Flow Diagram (DFD). We consider an integration of the two diagrams, which we call ER-DFD, in which the data stores of DFD are substituted by the entities and relationships of the ER. We have specialized the general rules for counting Function Points for the case of a specification in the form of an ER-DFD model. The informal counting rule expressed in natural language have been translated into formal rules that express properties of the ER-DFD graph. A knowledge based system has been implemented in Prolog that automatically counts Function Point by analyzing the graph.

Keywords: Software Engineering, Software Metrics, Knowledge Based Systems

1 Introduction

Software metrics are emerging as a powerful tool for the management of the software development process. The field of software metrics is a relatively young one, its origins are in the work by Halstead published in 1972. From then on, the interest in software metrics has steadily increased because they have been recognized as an helpful instrument for managing effectively the software process. Software metrics allow to use a real engineering approach to software development, providing the quantitative and objective base that software engineering was lacking. In fact, their use in industry is becoming more and more widespread.

Among software metrics, Function Points [Albrecht 79] [Albrecht and Gaffney 83] (FP for short in the following) give a measure of the size of the system by measuring the functionalities that the system offers to the user. This metrics is applicable both at the beginning of the development process, in the requirements or specification phases, and at the end of the process, after implementation. When FP are measured from requirements or specifications, the metric is used in order to make prediction about effort and therefore costs: [Albrecht and Gaffney 83] showed that FP are highly correlated with work-hours and argued that FP can be used effectively for estimating effort. When a developed application is measured, the metric can be used for evaluating the effectiveness of the process, by comparing the achieved productivity with productivity on previous projects of the same organization or with the average productivity of the industry.

FP measurement rules are defined in the International Function Point User Group (IFPUG) Counting Manual [IFPUG 94a]. The method is based on identifying and counting the functions that the system has to perform, i.e. logical internal files and external interface file (data functions), external inputs, external outputs and external inquiries (transaction functions). Each function identified in the system is then classified to three levels of complexity (i.e. simple, average and complex), and according to the complexity and the function type, a number of FP is assigned to each function. The sum of the contributions from all the functions gives the unadjusted FP count. The final FP count is then obtained by multiplying the unadjusted count by an adjustment factor that expresses the influence of 14 general system characteristics.

A great number of tools [Mendes et al. 96] exists on the market that help the software engineer in the process of FP counting. However, only a very limited number of them is able to completely automate the identification of functions and the evaluation of their complexity.

Identification is a well-known problem which can be solved by applying knowledge-based techniques. In this work, we propose an approach for the automated measurement of the FP metric starting from the specifications of a software system expressed in the form of an Entity Relationship (ER) diagram plus a Data Flow Diagram (DFD). In particular, we consider an integration of the two diagrams inspired to [Fuggetta et al. 88], which we call ER-DFD, in which the data stores of DFD are substituted by the entities and relationships of the ER. In order to automate the measurement process, we have specialized the rules for counting FP for the case of a specification in the form of ER-DFD model. The informal counting rule expressed in natural language in [IFPUG 94a] have been translated into formal rules that express properties of the ER-DFD graph. The overall system has been implemented in Prolog and automatically counts Function Points by analyzing the ER-DFD graph according to the formal rules expressed as clauses.

The advantages we obtain by our approach are twofold: first we provide a tool which automate the FP method in the identification task, at least for a certain kind of specification (i.e., ER and DFD). Furthermore, the formalization of the rules eliminates the ambiguous points in the IFPUG rules when counting from ER-DFD diagrams and can be beneficial to improve our understanding of the strengths and weaknesses of FP.

Finally, by applying a knowledge-base approach to the identification of FP functions, we achieve for the implemented system the usual advantages of this approach, such as readability, easiness of maintenance, modularity, separation of domain knowledge and control knowledge (see, for instence, [Stefik 95]).

The paper is organized as follows. In Section 2 we describe the FP measurement process. In Section 3 we recall ER and DFD specification and present their integration into ER-DFD model. Section 4 formalizes the FP rules for ER-DFD model, and section 5 shows and example. The implementation of the system is described in section 6. Related works are mentioned in section 7. Conclusions and future work follows. In the Appendix the IFPUG counting rules are reported in details and it is shown how they have been translated to formal rules.

2 Function Point Measurement Process

The calculation of FP is performed in 6 steps:

1) Identifying the type of FP count: development project, enhancement project or application.
2) Identifying the boundary of the application subject to the measure.
3) Identifying the data functions, classified as Internal Logical Files (ILF) and External Interface Files (EIF), and evaluating their complexity by counting the number of Data Element Types (DET) and Record Element Types (RET)

4) Identifying the transaction functions, divided in External Inputs (EI), External Outputs (EO) and External Inquiries (EQ) and their complexity by counting the number of Data Element Types (DET) and File Types Referenced (FTR)
5) Determining the number of unadjusted FP by summing the contributions of all functions
6) Calculating the final number of FP by multiplying the unadjusted FP count for an adjustment factor. The factor is obtained by evaluating the 14 general characteristics of the system on which the application will run.

Our aim is to automate steps 3), 4) and 5). Steps 3) and 4) are the most complex, time consuming and prone to error, therefore they are the most interesting to automate. We assume that the type of FP and the boundary of the application are known. The boundary of the application is indicated in the ER-DFD diagram. We did not automate step 6 because it requires many other notions on the application and on the environment not present in the ER-DFD specification and because, for some characteristics, it requires the subjective judgment of the counter.

3 Entity Relationship - Data Flow Diagrams

We will perform the measurement on a specification of the application composed of an Entity Relationship (ER) diagram [Chen 76] integrated with a Data Flow Diagram (DFD) [DeMarco 78]. We consider an integration of the diagrams which is similar to the one in Formal Data Flows Diagrams [Fuggetta et al. 88]: the data stores of DFD are replaced by entities and relationships of the ER diagrams, therefore we have data flows directly entering and coming out from entities and relationships. We call such an integration ER-DFD. In order to distinguish between elements of the DFD and ER diagrams which have a similar graphical symbol, we adopt the following conventions: external agents of DFD are represented with a shadowed box to distinguish it from entities represented as normal boxes and data flows are represented by arrows while the connections between entities and a relationships are represented by lines. In figure 1 an example diagram is shown.

A number of fields are associated to each data flow: a field corresponds to an attribute of an entity or a relationship when it has the same name. When the field does not correspond to any attribute, it can represent control information used to influence the function performed by the process, or it can represent data derived from attributes through an elaboration.

We suppose that the diagram contains also the indication of the boundaries of the different applications in the form of dashed lines.

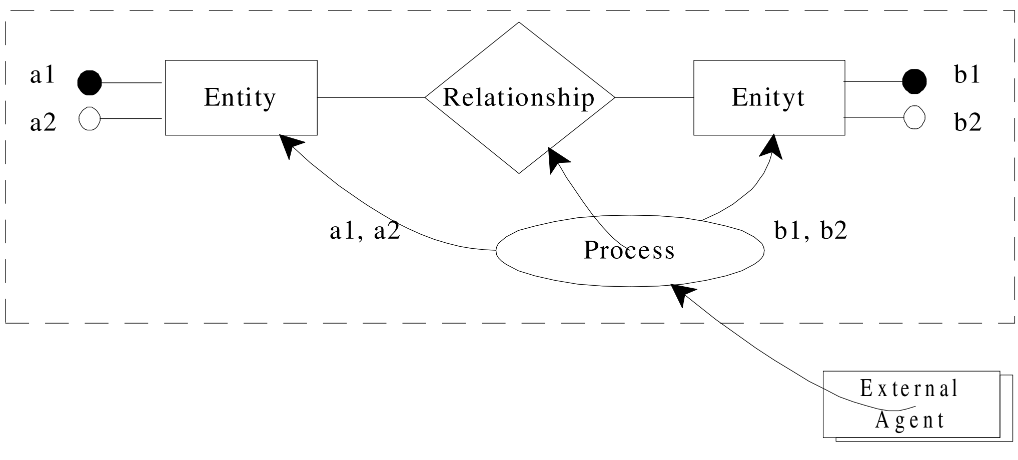


Figure 1: example of an ER-DFD diagram.

4 Formal Rules for Counting Function Points on ER-DFD

In this section we present the formal rules for counting Function Points from the specification of an application expressed in the form of an ER-DFD diagram. The detailed analysis of IFPUG counting rules and the description of how each single condition has been translated to a formal condition is reported in the Appendix. For ILFs we present both the IFPUG identification rule and the formal rule, so that the reader can compare the two. For the other data and transaction functions, we directly present the formal rules.

In order to make possible the translation to formal rules, we had to make a number of simplifying assumptions:

1) Every process in the DFD is an elementary process, i.e. it is the smallest unit of meaningful activity from the user perspective, it is self contained and leaves the application in a consistent state.
2) The processing logic of every process in the DFD is unique.
3) Every attribute of an entity or a relationship is unique, user recognizable and non recursive, i.e., it is a DET. Attributes are not recursive because a recursive field is an external key and external keys are not explicitly represented in ER.
4) Every field associated to a data flow that is not an attribute of a logical file, represent control information specified by the user in order to ensure conformity with the requirements or represents information derived from attributes of entities.
5) All the fields of the data flows from or to an ILF are DETs of the ILF.
6) All the fields of the data flows from an EO or an EQ to the user are either DETs of the referenced files or data derived from these DETs. We assume to have no details about the format of the output in the data flow fields. These details are considered only when counting from implementation.

These assumptions are not very restrictive since they represent the typical situation and ensure the correctness of the specification. For example, it is normal that every process in the DFD is elementary and unique, otherwise the DFD would be a bad specification document because of redundant information. It is normal also that each attribute of an ER is unique, user recognizable and does not represent a foreign key, if the ER has been correctly laid down. Also, the fields of the data flows from or to an ILF must be DETs, because no other data can come from or go to an ILF.

In section 4.1 we describe the formal rules for the identification of the Data Function Types, while in section 4.2 we give the complexity rules for them. In section 4.3 we describe the identification rules for the Transaction Function Types and we give the complexity rules in section 4.4.

4.1 Identification Rules for Data Function Types

Data Function Types are Internal Logical Files and External Interface Files. The IFPUG definition of an ILF is:

> "an ILF is a group of logically related data or control information, user identifiable, maintained inside the application boundary" [IFPUG 94a].

The identification rule for ILF is: a group of data or control information is an ILF if it satisfies all the following conditions:

1) "The group of data of control information is a logical, or user identifiable, group of data that fulfills specific user requirements.
2) The group of data is maintained within the application boundary.
3) The group of data is modified, or maintained, through an elementary process of the application.
4) The group of data identified has not been counted as an EIF for the application." [IFPUG 94a].

We propose the following identification rule: a set of connected entities and relationships is an ILF if:

1) All the elements of the set are inside the application boundary.
2) There is at least one process that has data flows entering in each element of the set and no flows entering in entities or relationships outside the set.

In the following we show some examples of the possible cases that can occur in practice. In figure 2 it is shown the simplest case: an ILF composed by a single entity. In this case the entity is clearly modified through an elementary process of the application and therefore forms an ILF by itself.

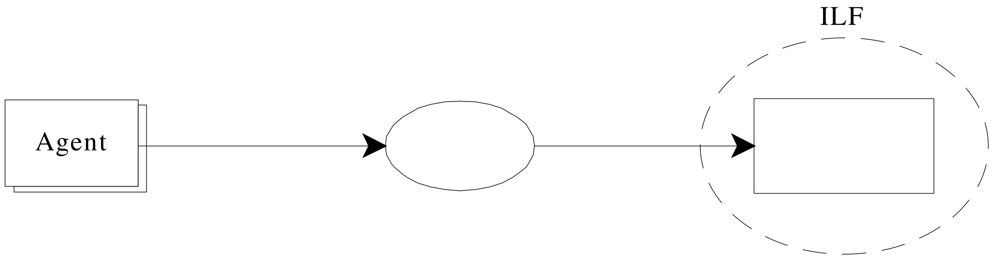


Figure 2: ILF composed of one entity

Figure 3 shows the case of an ILF composed by two entities and the relationship connecting them. Each entity separately would not be an ILF because they are not maintained by an elementary process. In fact, modifying just one entity does not leave the application in a consistent state. Both entities and the relationship between them must be updated to reach a new consistent state, therefore they form a single ILF.

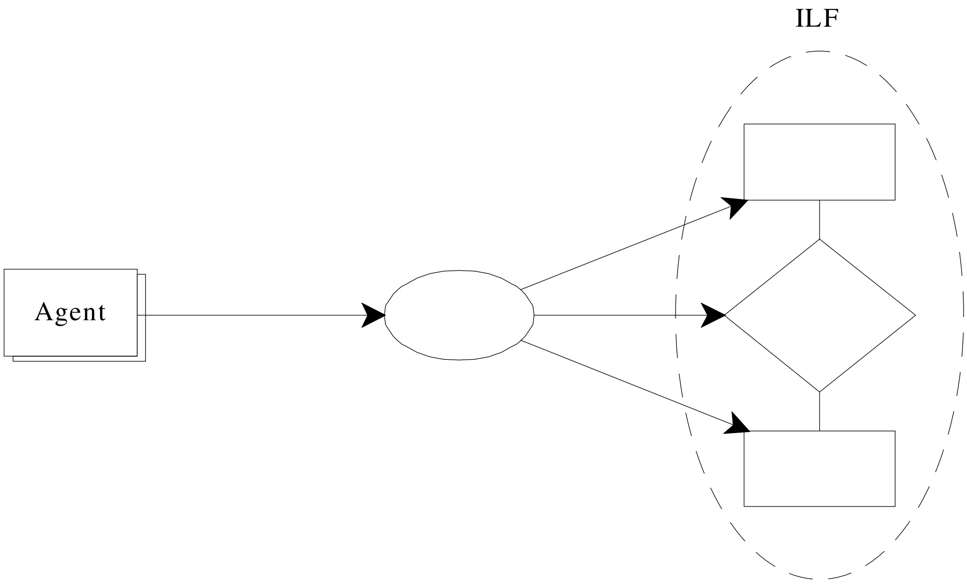


Figure 3: ILF composed of two entities and one relationship.

In figure 4 we present a more complex case: two ILFs that have an entity in common. It may seem unnatural to have an entity in two ILFs. However, if we consider the entity in the intersection as an ILF in itself, we contradict the condition that requires that an ILF is maintained through an elementary process of the application, because no process maintains only that entity.

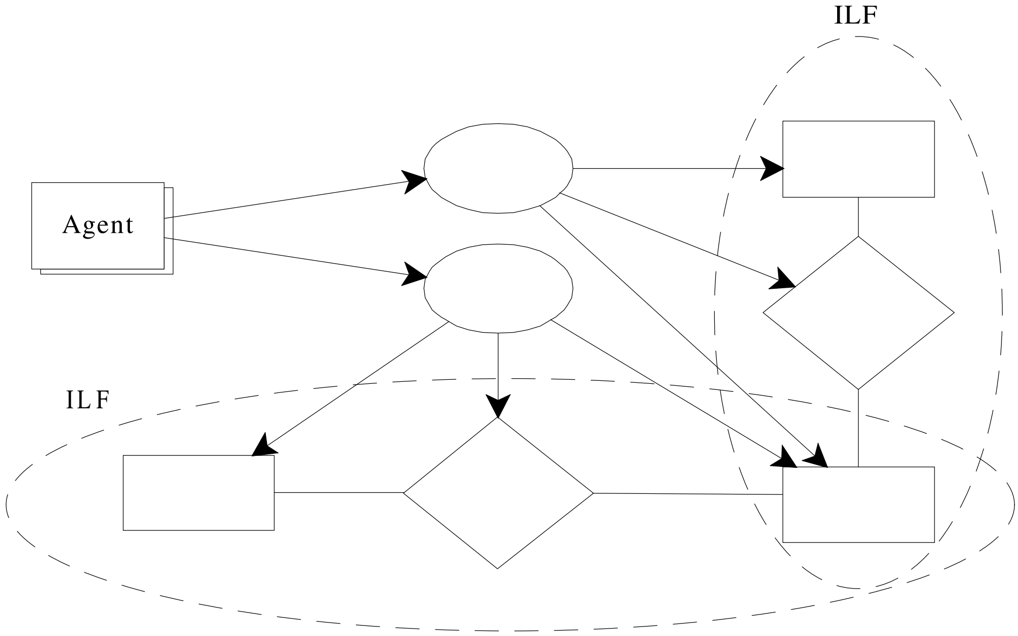


Figure 4: two ILFs composed of two entities and a relationship.

The IFPUG definition of an EIF is:

> "an EIF is a group of logically related data or control information, user identifiable, referenced by the application but maintained inside the boundary of another. This means that an EIF counted for an application must be an ILF for another." [IFPUG 94a].

The formal rule we propose is: a set of connected entities and relationships in an external application is an EIF if:

1) The set satisfies the condition for ILFs with respect to the external application.
2) There is at least one data flow crossing the boundary that go from an element of the set to processes inside the application.
3) There are no data flows from the counted application to any entities or relationships of the set.

In figure 5 we show an example in which we have an EIF composed of two entities and a relationship. The set is outside the counted application boundary and satisfies the rule for ILFs. Moreover, it has a data flow going from one entity to a process in the counted application: it is important to note that it is not necessary to have data flows from all the elements of the set. Finally, no data flow enters in the set from the counted application.

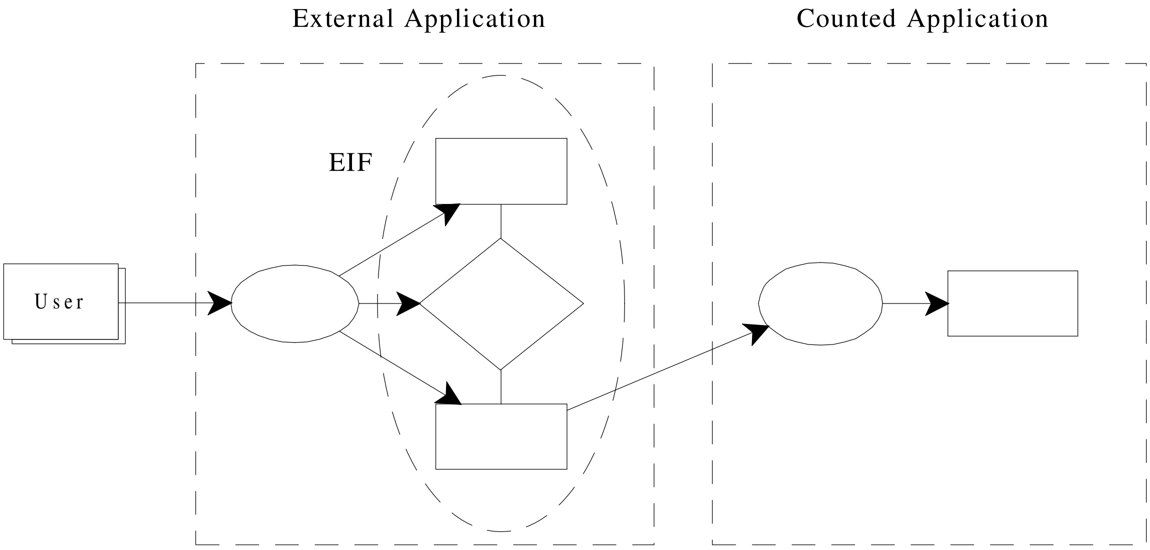


Figure 5: example of an EIF.

4.2 Complexity Rules for Data Function Types

In order to assign the right number of FP to each identified data function, we have to count the Data Element Types and Record Element Types associated with the function. "A Data Element Type (DET) is an unique field, user recognizable and non recursive on an ILF or EIF" [IFPUG 94a]. We count DETs in the following way.

1) One DET for each attribute of the entities and relationships in the logical file.
2) One DET for each attribute composing the key of the entities in other files connected by relationships to entities of the file (external keys). We do not count any DET for an external key in the case in which it contains the internal key of an entity of the ILF, otherwise we would doubly count the internal key.

   "A Record Element Type (RET) is an user identifiable subgroup of data elements in an ILF or EIF" [IFPUG 94a].

We count one RET for each entity and for each relationship with attributes of the logical file. If the file contains hierarchies, then we count RETs in the following way. Let us consider the case of an entity with two sub-entities. Depending on the type of hierarchy, we have four cases.

1) Exclusive and total hierarchy: we have 2 mandatory RETs because we have two possible subgroups, one made by the attributes of the father entity plus the attributes of one sub-entity and the other made by the father plus the other sub-entity.
2) Exclusive hierarchy: we count 3 RETs, 1 mandatory for the father entity and 2 optional for the sons, because an instance of the father may not be in any of the sons.
3) Total hierarchy: we count 3 mandatory RETs, one made by the father entity plus one son, one made by the father entity plus the other son and one made by the father entity plus two sons.
4) Neither total nor exclusive hierarchy: we count 1 mandatory RET for the father and 2 optional RETs for the sons.

4.3 Identification Rules for Transaction Function Types

Transactional Function Types are External Inputs, External Outputs and External Inquiry.

   "An External Input (EI) elaborates data or control information coming from outside the application boundary. The External Input is itself an elementary process. Elaborated data maintain one or more ILFs. Control information may or may not maintain an ILF" [IFPUG 94a].

We have to distinguish between EI of data and EI of control information. In order to identify an EI of data, we have to consider a process of the ER-DFD diagram. The process is an EI if the following conditions are satisfied:

1) It has one data flow from outside the application boundary.
2) It has a data flow entering in an ILF.

The simplest case of an EI of data is shown in figure 2.

A process in the ER-DFD is an EI of control information if

1) It has one data flow from outside the application.
2) The fields of the data flows are not attributes of any entity in an ILF.

The simplest case of an EI of control information is shown in figure 6.

Figure 6: simplest case of EI of control information.

The only difference between EI identification rules for data and control information is that control information may or may not maintain an ILF. Therefore, we do not have to check for a data flow going from the process to an ILF.

"An External Output (EO) is an elementary process that generates data or control information that are sent outside the application boundary" [IFPUG 94a].

A process is an EO if

1) it has at least one data flow from an ILF;
2) it is has one data flow F_{OUT} going outside the application boundary;
3) F_{OUT} contains at least one field that is not contained in none of the data flows from ILFs.

In figure 7 it is shown the simplest case of an EO.

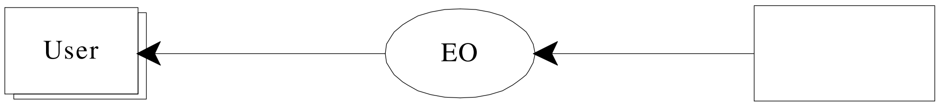


Figure 7: simplest case of an EO.

"An External Inquiry (EQ) is an elementary process composed by a combination of input and output that results in data retrieval. The output side does not contain derived data. No ILF is maintained during the process" [IFPUG 94a].

A process is an EQ if

1) it has at least one data flow from outside the application (input part);
2) it has at least one data flow F_{OUT} that goes outside the application (output part);
3) it has at least one data flow $F_{IL}F$ from an ILF;
4) F_{OUT} and F_{ILF} have the same fields.

In figure 8 the simplest example of an EQ is shown. What distinguishes an EO from an EQ is the fact that an EQ does not elaborate the data retrieved, while an EO outputs derived data.

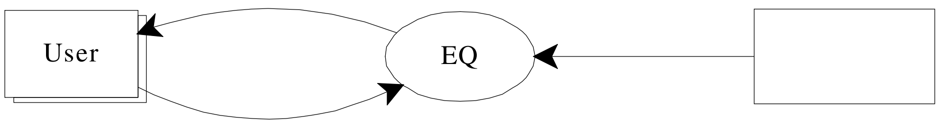


Figure 8: simplest case of an EQ.

4.4 Complexity Rules for Transaction Function Types

In order to assign the right number of FP to each identified transaction function, we have to count the File Types Referenced and DETs associated with the function.

"A File Type Referenced (FTR) is

- an internal logical file read or maintained by the function
- an external interface files read by the function" [IFPUG 94a].

For EI we count one FTR for each file that is connected to the process and one DET for each field of the data flows from ILFs to the process.

For EO, we count one FTR for each file that is connected to the process and one DET for each field of the data flows from the process to the user.

For EQ, we have to distinguish between the input side and the output side. We consider the input side as the data flow from the user to the EQ, while the output side is composed of the data flows from the files to the EQ and from the EQ to the user. As regards the calculation of FTRs, for the input side we consider the fields of the data flow from outside and we count one FTR for each logical file that has a DET among those fields. For the output side, we count one FTR for each logical file is connected to the process through a data flow. As regards the calculation of DETs, for the input side we count one DET for each field of the data flow from the user to the process. For the output side, we count one DET for each field of the data flows from the files to the process.

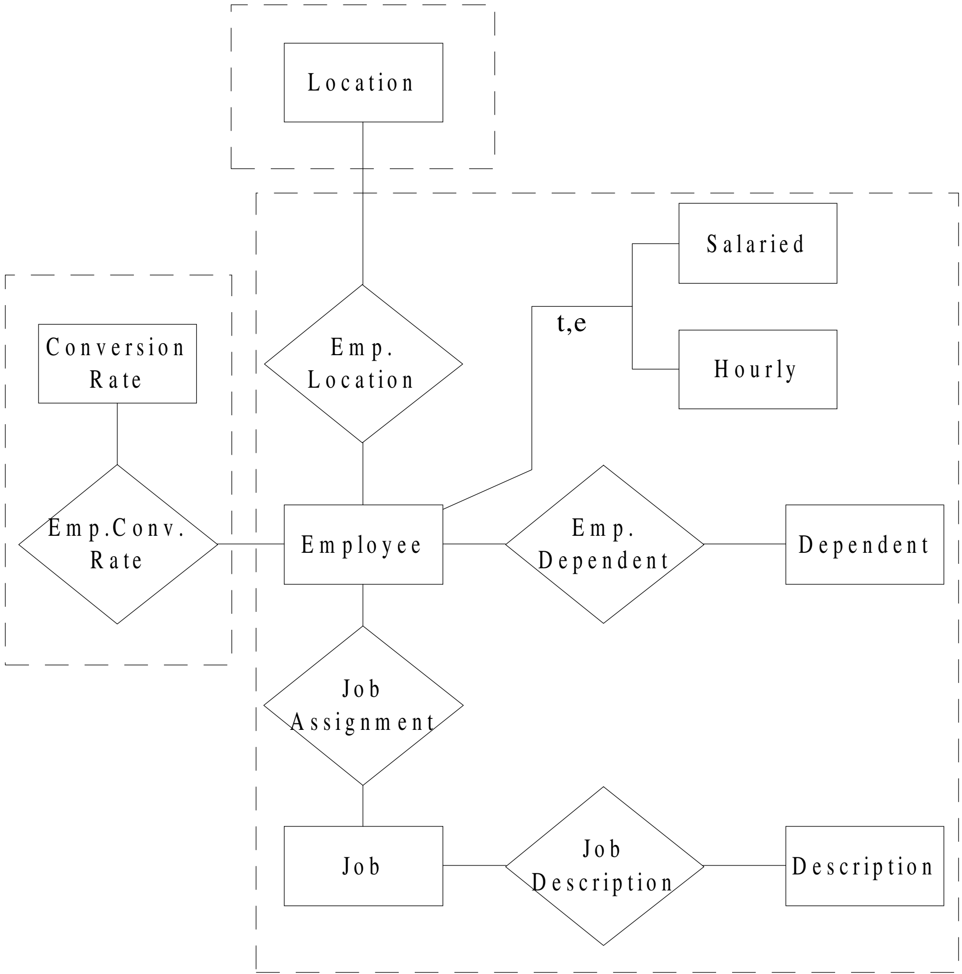
5 Example

In this section we describe the application of the formal rules to the measurement of an application for the management of Human Resources. This application is the subject of a series of case studies of Function Point measurement published by IFPUG, we will consider the case study [IFPUG 94b] in which the measurement is performed starting from the specification of the application expressed as an ER diagram and a DFD.

The aim of the application is to manage information about employees of a firm. In particular, the user requires to store information about each employee, comprehending data on the dependents of the employee, data on the salary or the hourly rate and data on the work location. The location must be a valid location in the Fixed Asset System. If the employee works abroad, the hourly rate must be converted to US dollars by accessing the Currency Application System and retrieving the conversion rate. Moreover, the application has to store information about different jobs, together with a description composed of a series of text lines. Finally, the user requires to store information about the assignment of jobs to employees. In figure 9 is reported the ER diagram alone.

Some of the operations required by the user are:

- add an employee, together with data on his dependents and on the salary or hourly wage (figure 10);
- report on all employee, printing the list of employee together with the total number of employee (figure 11);
- inquire on the data of an employee, given his social security number (figure 12);
- add job information, together with its description (figure 13);
- add a job assignment (figure 14).



Entities or relationships	Attributes
Employee	Social_Security_Number (key), Name, Nbr_Dependents, Type_Code.
Salaried	Supervisory_level
Hourly	Standard_Hourly_Rate, US_Hourly_Rate, Collective_Bargaining_Unit_Number
Dependent	Dep_SSN (key), Dep_name, Dep_birth_date
Job	Name, Job_Number (key), Pay_grade
Description	Line_Number, Description_Line
Job Assignment	Effective_Date, Salary, Performance_Rating
Location	Location_Name (key), Address, City, State, Zip, Country.
Conversion Rate	Conversion_Rate_To_Base_Currency.

Figure 9: complete ER diagram for the Human Resource application.

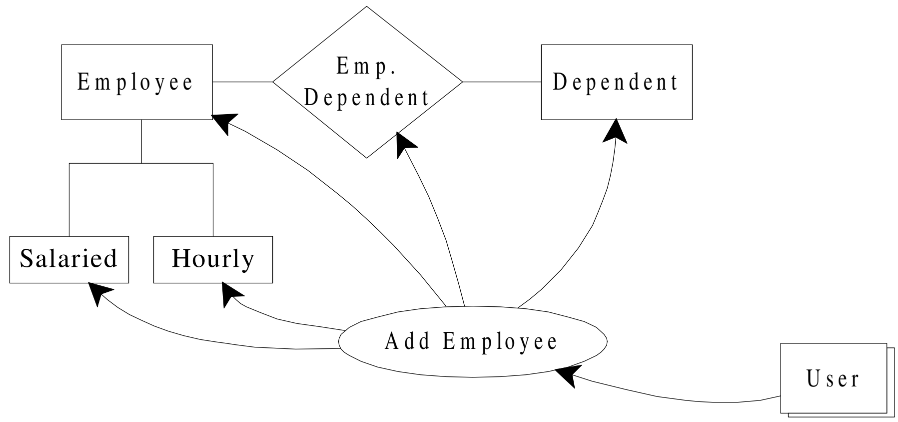


Figure 10: Add Employee process.

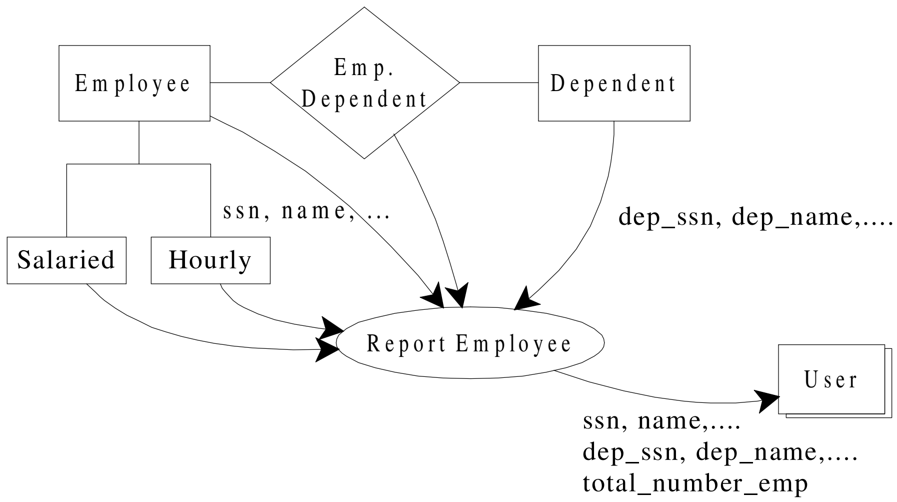


Figure 11: Report Employee process.

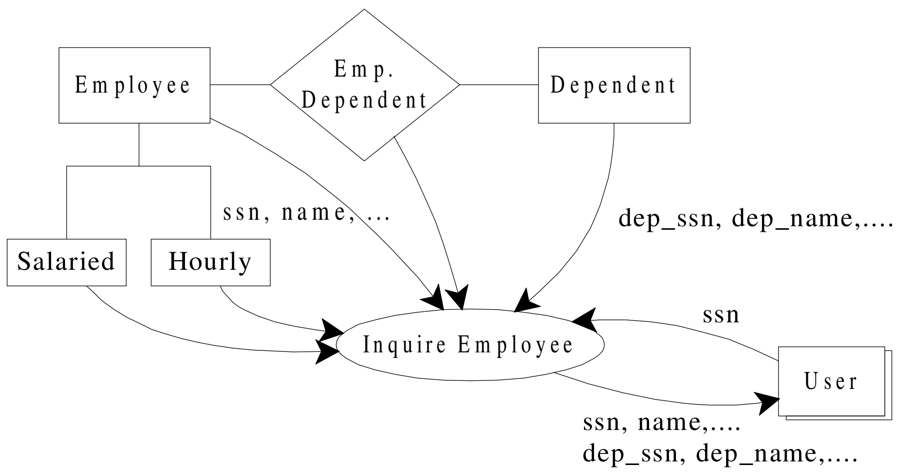


Figure 12: Inquire Employee process.

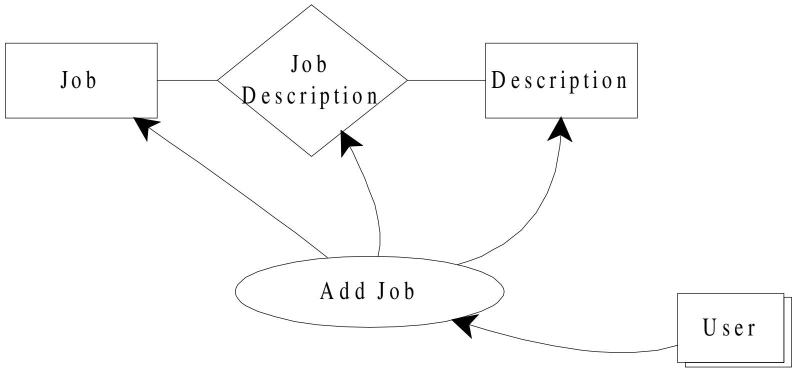


Figure 13: Add Job process.

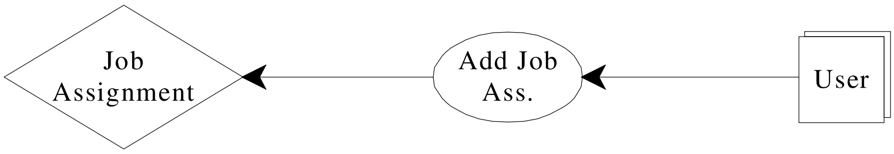


Figure 14: Add Job Assignment process.

We consider first the identification of ILFs. The set formed by the entities Employee and Dependents, their relationship and the sub-entities of Employee Salary and Hourly, is an ILF because it is inside the application boundary and the process Add Employee has data flows to all of them. This means that the entities of the set are maintained through the elementary process Add Employee: it is not possible to add an Employee without adding also his Dependents and therefore they must be put together in a single logical file. As regards the complexity count, the ILF has one DET for each attribute of its entities plus 2 DET for the external keys: one of the key of the entity Location and one for key of Conversion Rate. We then count 2 RET for Employee and its two subentities, since the hierarchy is total and exclusive, 1 RET for Dependent and no RET for the relationship Employee Dependent because it does not have any attribute.

The set made by the entities Job and Description and by the relation between them is an ILF because it is inside the application boundary and the process Add Job has data flows to all of them. Job Assignment is an ILF because it is inside the application boundary and the process Add Job Assignment has a data flow to it.

The ILF Job-Description has 2 RETs and 5 DETs: 1 RET for Job, 1 for Description, 5 DETs for the five attributes of Job and Description. We do not count 1 DET for the key of Job Assignment because it is constituted by the keys of Employee and Job, otherwise we would doubly count the attribute Job_number. The ILF Job Assignment has 1 RET and 5 DETs: 3 DETs for the three attributes of the relationship and 2 DET for the external keys Social_Security_Number (link to Employee) and Job_Number (link to Job).

As regards EIF, we are not able to fully apply the rules because we do not know the processes for the external applications. However, we can observe that the entity Location has a data flow from the counted application and, supposing it is maintained by a process of the external application, we can consider it as an EIF. The same reasoning applies to the entity Conversion Rate and the relationship Employee-Conversion Rate that constitute another EIF.

Let us now identify transactional functions. The processes Add Employee, Add Job Assignment and Add Job are EIs because they have a data flow from outside the application (the user in this case) and they have data flows entering in all elements of an ILF.

The process Report Employee is an EO because it has data flows from an ILF, a data flow F_{OUT} going outside the application, and F_{OUT} contains one field, total_number_emp, that is not in the flows from the ILF.

The process Inquire Employee is an EQ because it has a data flow from outside, a data flow F_{OUT} going outside and a data flows from an ILF. Moreover, the fields in the data flow F_{OUT} are the same as those of the flows from the ILF.

6 Implementation

The system was developed using Sicstus Prolog 3#5 [SICS 95]. The ER-DFD diagram is given as input to the system in the form of a Prolog program, using the following predicates:

- `application(name,[entity1,..entityN,relationship1,...,relationshipM],[process1,...,processP])`
  indicates which processes, entities and relationships are contained in the application `name`.
- `entity(name,[keyattribute1,...,keyattributeN],[attribute1,...,attributeN])`
  `relationship(name,attribute1,...,attributeN])`
  `connection(relation,entity,cardmin,cardmax)`
  `specialization(entity,[entitychild1,...entitychildN],Total,Exclusive)`
  represent entities and relationships, together with their attributes, the connections between them, the cardinality of connections, and the hierarchies of entities.
- `dataflow(sorg,dest,[field1..,fieldN])`
  represent a data flow between processes and entities or relationships, together with the list of associated fields.

From this input and through a set of clauses that implement FP rules, the system identifies the functions, evaluates their complexity and returns the unadjusted FP count. In order to illustrate the behavior of the system, let us show how the ILF are recognized. The predicate `ilf(Appl,EntList)`, given the application name `Appl`, returns in `EntList` the list of entities and relationships of an ILF of the application. By backtracking on the solutions, we obtain all the ILFs of the application, possibly with duplicates, which are successively removed by another predicate. The clause for ILF identification is:

```
ilf(Appl,EntList):-
              application(Appl,EntList1,ProcList),
              member(Proc,ProcList),
              findall(Ent,dataflow(Proc,Ent,_),EntList),
              sublist(EntList,EntList1),
              EntList \== [].
```

The predicate `application` returns in `Entlist1` the list of all entities of the application `Appl`, then we select a process `Proc` with `member` and we find the list `EntList` of all the entities that are reached by data flows from `Proc`. These entities form an ILF if they are all contained in the application (predicate `sublist`) and if the set is not empty. For the lack of space the rest of the code is not reported but is available from the authors.

7 Related Works

[Mendes et al.] is a survey of FP tools available on the market. The authors review 52 tools, out of which only 8 perform automatic FP counting:

Tool	Vendor
Autopoint	Integrated Software Specialists
Before You Leap v. 1.52	Strategic Systems Technology Ltd.
Composer FP Report	Texas Instruments
FP Analyst	Cayenne Software Inc.
LDA - LINC Development Assistant	Unisys
Revolve v. 3.1	Micro Focus Ltd.
PCA Calc Add-on (prototype)	System House SHL Québec
VIA RECAP: ESW Portfolio Analysis	VIASOFT Inc.

Among these 8 tools, 3 perform the counting starting from ER, DFD or similar diagrams: *FP Analyst* counts FP starting from Cayenne models that contain DFD, *Composer FP Report* starts from ER diagram plus Dialog Flow / Process Action diagrams, *Before You Leap* starts from DFD an ER-D. All these tools perform identification of the functions and evaluation of their complexity, except for *Before You Leap* that is not able to count DETs and RETs for data function types. Moreover *FP Analyst* and *Composer FP Report* are also able to automatically identify the application boundary and *FP Analyst* is able to decide which files and processes are unique, while we consider all files and processes as unique by assumptions. However, being commercial products, it was not possible to get to know the detailed behavior of these system and therefore a comparison with our system is difficult. In any case, our analysis, besides leading to a counting tool, has the benefit of increasing our understanding of the FP counting rules when counting from ER-DFD, removing the ambiguities that are present in the counting rules and translating them into simple unambiguous formal rules by making only a limited number of assumptions.

Moreover, by adopting a knowledge based approach for the architecture of our tool, we obtain a system that is very easy to maintain, allowing us to modify the rules as our understanding of them improves or because of introduction of new rules.

6 Conclusions and future works

In this paper we have presented a knowledge based system for the automatic counting of the FP metric starting from an ER-DFD diagram, a formalism that integrates ER and DFD by replacing the data stores of DFD with the entities and relationships of ER. The FP counting rules have been specialized for the case of ER-DFD and made formal by making a number of simplifying assumptions. The formal rules express simple properties of the ER-DFD graph that can be easily checked both by a human or a computer program. In this way we have obtained a twofold result: we have obtained an automated counting tool and we have removed the ambiguities in FP rules when counting from ER-DFD.

In the future, on one side we intend to continue the testing of the system on practical cases and, on the other side, we intend to investigate the counting of FP from other, more formal, specification languages, such as Z [Spivey 89].

Bibliography

[Albrecht 79] A. Albrecht. *Measuring application development productivity.* in *Proc. Joint SHARE/GUIDE/IBM Applications Development Symposium*, Monterey, CA, 1979

[Albrecht, Gaffney 83] A. Albrecht, J. Gaffney: *Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation*; in IEEE Trans. Software Eng., 9(6), 1983, pp. 639-648

[Chen 76] P.P. Chen. *The Entity-Relationship model. Toward a unified view of data*. ACM Transactions On Database System, Vol. 1, No. 1, Marzo, 1976

[DeMarco 78] T. DeMarco. *Structured Analysis and System Specification.* Yourdon Press, New York, 1978.

[Fenton 94] N. Fenton, *Software Measurement: a Necessary Scientific Basis*, IEEE Trans. Software Eng., 20, 1994, pp. 199-206.

[Fuggetta et al. 88] A. Fuggetta, C. Ghezzi, D. Mandrioli, A. Morzenti, "VLP: a Visual Language for Prototyping", IEEE Workshop on Languages for Automation, College Park, MD, August 1988.

[IFPUG 94a] International Function Points User Group, *Function Point Counting Practices Manual*, *Version 4.0*, 1994.

[IFPUG 94b] International Function Points User Group, *Function Point Counting Practices: Case Studies, Case Study 2, Release 1.0*, 1994.

[Mendes et al. 96] O. Mendes, A Abran, P. Bourque, *Function Point Tool Market Survey 1996*, Research Report, Software Engineering Management Laboratory, Université du Quebec à Montreal, 14 December 1996.

[SICS 95] *SICStus Prolog User Manual, Release 3#0*, Swedish Institute of Computer Science, 1995.

[Spivey 89] J. M. Spivey, *The Z Notation: A Reference Manual*, Prentice Hall, 1989.

[Stefik 95] M. Stefik, *Introduction to Knowledge System*s, Morgan Kaufmann Pub., S. Francisco CA, (USA), 1995

Appendix

We consider each condition of the IFPUG counting rules singularly and we explain how it has been translated to a formal condition on the ER-DFD diagram.

Identification Rules for Data Function Types

ILF: identification rules	
IFPUG rules	Formal rules
1) The group of data of control information is a logical, or user identifiable, group of data that fulfills specific user requirements.	1) An ILF is formed by a set S of connected entities and relationships.
2) The group of data is maintained within the application boundary.	2) S is inside the application boundary.
3) The group of data is modified, or maintained, through an elementary process of the application.	3) There is a process P such that all the data flows from it go to one of the elements of S and all the elements of S are reached by a data flow from P. Moreover P is elementary by assumption.
4) The group of data identified has not been counted as an EIF for the application.	4) The fact that S is internal excludes this possibility.

EIF: identification rules	
IFPUG rules	Formal rules
1) The group of data of control information is a logical, or user identifiable, group of data that fulfills specific user requirements.	1) An EIF is formed by a set S of connected entities and relationships.
2) The group of data or control information is referenced by, and external to, the application being counted.	2) The elements of S must be outside the application boundary and must have at least a data flow going from S to processes inside the application.
3) The group of data is **not** maintained by the application being counted.	3) S has no data flow coming from inside the application boundary. If it has one, we have probably made a mistake in the identification of the application boundary.
4) The group of data is counted as an ILF for at least one other application.	4) This has to be verified by applying the rule for ILFs to the external application.
5) The group of data has not been counted as an ILF for the application.	5) The fact that S is external excludes this possibility.

Complexity Rules for Data Function Types

ILF, EIF: rules for DETs	
IFPUG rules	Formal rules
1) Count one DET for each unique field, user recognizable, non recursive on an ILF or EIF	1) We count one DET for each attribute of entities and relationships of the file considered. We assume that each attribute is unique, user recognizable. In fact, recursive field are external keys that represent relationships between entities, while in ER diagrams external keys are not represented.
2) Count one DET for each field in an ILF or EIF that exist because the user has request the relation with another ILF.	2) We count one DET for each attribute in the key of the entities of other ILFs or EIFs in relation with the entities of the file considered (external keys).
3) Count the following technical solutions as one DET for the group of fields	3)
i) Fields that appear more than one time in an ILF or EIF because of technology or implementation techniques.	i) This is the case of a reflexive relationship. In this case, no DET is counted in 2) for the external key because the related entity is not outside the file.
ii) Repeated fields that have the same format and exist in order to give multiple occurrences of a datum.	ii) ER can have attributes with multiple values, therefore no extra DET are counted in 2) for repeated fields.

ILF, EIF: rules for RETs	
IFPUG rules	Formal rules
1) Count one RET for each optional or mandatory subgroup.	1) Count one RET for each entity and for each relationship with attributes in the file. If there are sub-entities, count them depending on the type of the hierarchy (see below).
2) If there are no subgroups, count the ILF or EIF as one RET.	2) The file is constituted by at least one entity or relationship, therefore this case is already accounted for in 1).

Let us consider the case in which an entity has an hierarchy of sub-entities.

Type of hierarchy	Number of RETs
total, exclusive	1 RET for each son
exclusive	1 RET for the father plus 1 RET for each son
total	1 RET for the father plus 1 RET for each possible combination of the sons
none	1 RET for the father plus 1 RET for each son

Identification Rules for Transaction Function Types

EI of data: identification rules	
IFPUG rules	Formal rules
1) The data are received from outside the application boundary.	1) An EI is a process P with at least one data flow from outside the application boundary.
2) Data in an ILF is maintained through an elementary process of the application.	2) The process P has a data flow entering in an ILF.
3) The process is the smallest unit of meaningful activity from the user perspective.	3) This condition is true by hypothesis, we assume that each process in the DFD is an elementary process.
4) The process is self contained and leaves the business in a consistent state.	4) True because P is elementary by hypothesis.
5) For the identified process, one of the following two rules applies:	5)
i) Processing logic is unique from other external inputs.	i) True by hypothesis, we assume that all the processes in the DFD are unique.
ii) Data elements are different from other external inputs.	ii) This condition is not verified because i) is always true.

EI of control information: identification rules	
IFPUG rules	Formal rules
1) The data are received from outside the application boundary.	1) An EI is a process P with at least one data flow from outside the application boundary.
2) Control information are specified by the user in order to ensure the conformity with functional requirements.	2) We assume that all the filed associated to the data flows going into the process that are not DET, are control information specified by the user in order to ensure conformity with the requirements.
3) For the identified process, one of the following two rules applies:	3)
i) Processing logic is unique from other external inputs.	i) True by hypothesis, we assume that all the processes in the DFD are unique.
ii) Data elements are different from other external inputs.	ii) This condition is not verified because i) is always true.

EO: identification rules	
IFPUG rules	Formal rules
1) The process sends data or control information outside the application boundary.	1) An EO is a process P with at least one data flow that goes outside the application boundary.
2) The process is the smallest unit of meaningful activity from the user perspective.	2) True because P is elementary by hypothesis.
3) The process is self contained and leaves the business in a consistent state.	3) True because P is elementary by hypothesis.
4) For the identified process, one of the following two rules applies:	4) This condition is true also for the hypothesis that the processes in the DFD are all elementary.
i) Processing logic is unique from other external outputs.	i) True by hypothesis, we assume that all the processes in the DFD are unique.
ii) Data elements are different from other external outputs.	ii) This condition is not verified because i) is always true.

Moreover, we require two other conditions for EOs:

1)It must have at least one flow from an ILF. This is natural because the case in which the process outputs data without retrieving it from an ILF is not very significant.

2)The flows going outside must have at least on field that is not present in the flows from the files. This is required in order to ensure that the output contains derived data, that is the condition for distinguishing EO from EQ, as explained below.

EQ: identification rules	
IFPUG rules	Formal rules
1) An input request enters the application boundary.	1) An EI is a process P with at least one data flow from outside the application boundary.
2) Output results exit the application boundary.	2) P has at least one data flow that goes outside the application boundary.
3) Data is retrieved.	3) P has at least one data flow from an ILF.
4) The data retrieved does not contain derived data.	4) The field in the data flow going outside the application are the same as the ones in the data flow from the ILF. Otherwise it is an EO.
5) The process is self contained and leaves the business in a consistent state.	5) True because P is elementary by hypothesis.
6) The process is self contained and leaves the business in a consistent state.	6) True because P is elementary by hypothesis.
7) For the identified process, one of the following two rules applies:	7)
i) Processing logic is unique from other external inquiries.	i) True by hypothesis, we assume that all the processes in the DFD are unique.
ii) Data elements are different from other external inquiries.	ii) This condition is not verified because i) is always true.

Complexity Rules for Transaction Function Types

EI: rules for FTRs	
IFPUG rules	Formal rules
1) Count one FTR for each ILF only maintained	1) Count one FTR for each ILF that has data flows coming from the process
2) Count one FTR for each ILF or EIF only referenced	2) Count one FTR for each ILF or EIF that has data flows going to the process
3) Count one FTR for each ILF referenced and maintained	3) Count one FTR for each ILF that has data flows going to and coming from the process

EI: rules for DETs	
1) Count one DET for each user recognizable, non recursive field maintained in ILF by an EI.	1) We count one DET for each field associated with the data flow from the EI to the ILF. We assume that all this fields are DETs of the ILF.
2) Count one DET for each field that is not inserted by the user but is maintained, through an external input, in an ILF.	2) This is already accounted for in 1) because we do not count the fields on the data flow from the user to the process but from the process to the ILF.
3) Count the following technical solutions as one DET for the group of fields	3)
i) Fields that are stored on more physical fields but is requested by the user as a single information.	i) Since fields in the data flow correspond to DETs of the ILF, we already do not consider the way they are stored. This regards the count from the implementation.
ii) Fields that appear more than one time in an ILF because of technology or implementation techniques.	ii) As i), true since we have consider DETs for the ILF.
iii) Fields that indicate an error during an elaboration or the completion of an elaboration.	iii) This is considered only when counting from implementation.
iv) Count one DET for the possibility of specifying the action that must be executed by the EI.	iv) As iii).

EO: rules for FTRs	
1) Count one FTR for each ILF or EIF referenced	1) Count one FTR for each ILF or EIF that has data flows going to the process

EO: rules for DETs	
1) Count one DET for each user recognizable, non recursive field that appears in the EO.	1) We count one DET for each field associated with the data flow from the EO to the user.
2) Do not count constant as DETs	2) We assume that the fields of the data flows are either DETs of the referenced files or data derived from these DETs. We assume not to have details about the format of the output in the data flow fields. These details are considered only when counting from implementation.
3) Do not count variables containing the page number or notation generated by the system.	3) Already verified by assumption that the fields do not contain details about the output format.
4) Count the following technical solutions as one DET for the group of fields	4) Already verified by the assumption that fields are DETs for the file referenced.
i) Fields that are stored on more physical fields but is requested by the user as a single information.	
ii) Every variable type and every type of numerical value associated reported in a graphical output.	
iii) Textual information that can be a single word, phrase or period.	

For EQ, we consider the input side as the data flow from the user to the EQ, while the output side is composed of the data flows from the files to the EQ and from the EQ to the user.

EQ: rules for FTRs	
Input side	
1) Count one FTR for each ILF or EIF referenced.	1) Consider the DETs associated with the flow from the process to the user. Count one FTR for each different ILF or EIF to which these DETs belong.
Output side	
1) Count one FTR for each ILF or EIF referenced.	1) Count one FTR for each ILF or EIF that has data flows going to the process

EQ: rules for DETs	
Input side	
1) Count one DET for each user recognizable, non recursive field that appears in the input side of an EQ.	1) We count one DET for each field associated with the data flow from the user to the EQ.
2) Count one DET for each field that specifies the data selection criteria.	2) We do not see any difference with 1): the field that appear on the input side are clearly used as the data selection criteria.
3) Count the following technical solutions as one DET for the group of fields	3)
i) Fields that indicate an error during an elaboration or the completion of an elaboration.	i) This is considered only when counting from implementation.
ii) Count one DET for the possibility of specifying which EQ must be executed.	ii) As i).
Output side	
1) Count one DET for each user recognizable, non recursive field that appears in the output side of an EQ.	1) We count one DET for each field associated with the data flow from the files to the EQ. They must be the same as the ones on the data flow from the EQ to the user, since the output can not contain derived data.
2) Do not count constant as DETs	2) We assume that the fields of the data flows are DETs of the referenced files. We assume not to have details about the format of the output in the data flow fields. These details are considered only when counting from implementation.
3) Do not count variables containing the page number or notation generated by the system.	3) Already verified by assumption that the fields do not contain details about the output format.
4) Count the following technical solutions as one DET for the group of fields	4) Already verified by the assumption that fields are DETs for the file referenced.
i) Fields that are stored on more physical fields but is requested by the user as a single information.	
ii) Fields that appear more than one time in an ILF or EIF because of technology or implementation techniques.	