A Generalized Representation for selected Functional Size Measurement Methods

Thomas Fetcke
Otto-von-Guericke-Universität Magdeburg
fetcke@acm.org

Abstract

The management of software cost, development effort and project planning are key aspects of software development. Functional size measurement (FSM) has been proposed as a tool for these management requirements.

Function Point Analysis (FPA) can be considered as the first FSM method published. Based on FPA, other methods have been proposed as improvements and alternatives that differ in their respective views on functional size.

FPA is an intuitive approach without theoretical foundation, and without a measurement model. It is therefore unclear, what FPA actually measures and what the differences between the FSM methods are.

We use an axiomatic approach based on measurement theory to develop a model for existing FSM methods. In this paper, we propose a model as a generalized representation for a set of methods: IFPUG FPA, Mark II FPA, and FFP. This view can be used as a basis for the analysis of FSM methods and for a discussion of their differences.

1. Introduction

The management of software cost, development effort and project planning are key aspects of software development. Software size is a critical element in these measurement requirements. Various approaches for the measurement of software size have been formulated, among others the number of lines of source code. Functional size measurement (FSM) methods have been proposed to overcome some of the deficiencies of approaches based on source code. It is the goal of these methods to measure the functionality of the software, independent of its implementation.

Function Point Analysis (FPA), published by Albrecht in 1979, can be considered as the first FSM method. Based on FPA, several revisions and alternative FSM methods have been formulated. These methods differ in their views and definitions of functional size.

FPA is an intuitive approach without theoretical foundation. With the lack of a measurement model for FPA, it remains unclear, what the method actually measures. It is therefore also difficult to analyze the differences between the different FSM methods.

We propose here a model for existing FSM methods that gives a new view on these methods and will help in understanding the methods. We use an axiomatic approach based on measurement theory to formulate our characterization. Based on the model, assumptions of reality can be formulated as axioms, and the FSM methods can be tested against the assumptions.

While FSM methods differ in their views on functional size, a number of methods share a core view and certain core concepts. We propose here a representation for a set of FSM methods. The representation is generalized such that it applies to each of these methods. The representation thus allows a detailed analysis and discussion of differences and common concepts of different FSM methods. Given such a representation, the actual measurement step can be formalized, as we have demonstrated in [6]. In this paper, we give a detailed description of the generalized representation, and we describe how it can be used for the discussion of FSM methods.

The following paragraphs give a short overview of the evolution of the FSM methods studied and on related work. Section 2 describes the approach taken and our view on the measurement process of FSM. The generalized representation is presented and discussed in Section 3. Section 4 describes the application of the generalization, and Section 5 gives some conclusions.

1.1. Evolution of FSM methods

Function Point Analysis (FPA) was developed by Albrecht in the 1970s, with its first presentation to the public in 1979 [4]. The purpose of FPA was to measure the amount of software produced. Albrecht wanted to measure the functionality of software from the user viewpoint, independently of the implementation. He therefore introduced

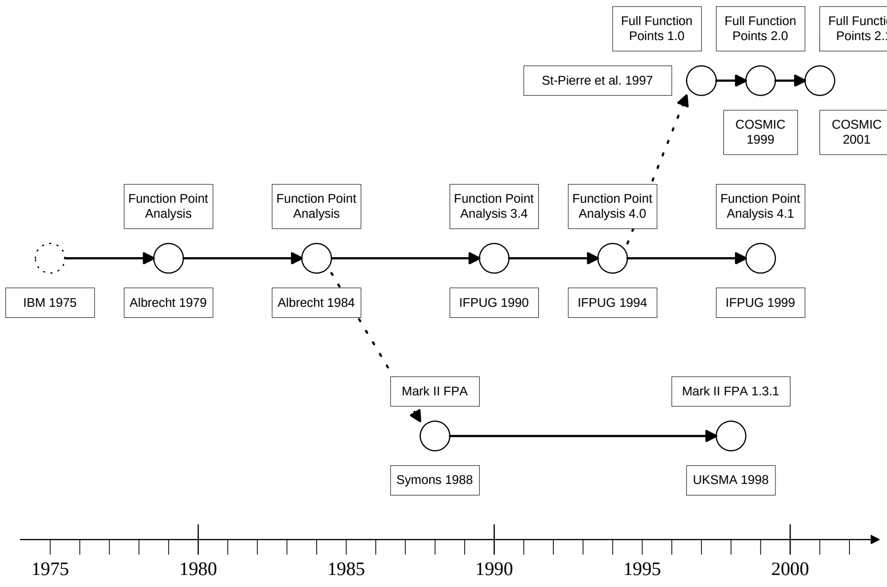


Figure 1. The evolution of selected functional size measurement (FSM) methods.

Function Points as a measure of "functional size".

In 1984, the International Function Point Users Group (IFPUG) was formed to maintain Albrecht's FPA. IFPUG has since then published Counting Practices Manuals that give standard rules for the application of FPA [7, 8]. IFPUG has thus both clarified the rules and modified Albrecht's original method.

Several authors published extensions and alternatives to the FPA versions of Albrecht and IFPUG. Symons [11] formulated several "concerns and difficulties" with Albrecht's FPA. His critique led him to the proposal of a new variant called Mark II Function Point Analysis. Today, the United Kingdom Metrics Association (UKSMA) maintains Mark II FPA [12].

In 1997, St-Pierre et al. [10] proposed the Full Function Points (FFP) approach as an extension to IFPUG FPA 4.0. The purpose of the extension was to capture the functional size of real-time applications. The Common Software Metrics International Consortium (COSMIC) has been formed in 1998 to develop an advanced FSM method. COSMIC has taken the FFP approach as a basis and has published revisions of this method as COSMIC-FFP 2.0 [1] and 2.1 [2].

For this study, we focus on the following FSM methods:

- IFPUG FPA release 4.0 and 4.1 [7, 8],
- Mark II FPA 1.3.1 [12], and
- FFP 1.0, COSMIC-FFP 2.0 and 2.1 [10, 1, 2].

We will see that these methods share a core view of the items that determine functional size. Figure 1 presents a view of the evolution of these FSM methods.

1.2. Related studies

Abran et al. [3] analyze the measurement process of IFPUG FPA. In their view, FPA constructs the Function Point measure in a hierarchical process of measurements. The counting of data elements, e.g., is considered as a measure on the lowest level of the hierarchy. Based on the lowest level measurements, higher levels are constructed, e.g., the assignment of weights to transaction types. Abran et al. then identify scale types for the measurements at each level.

We view FSM differently, as a single measurement that assigns numbers to software applications. Based on measurement theory, we can thus discuss conditions or axioms that formulate assumptions of reality (cf. [13, Chapter 4]). Hence, we attempt to get a better understanding of existing FSM methods.

2. FSM measurement process

Measurement can be understood as an abstraction that captures certain attributes of the measurement objects. Measurement theory views this abstraction as a mapping that assigns numerical objects to empirical objects. In the context of FSM, the empirical objects are software applications. In terms of ISO 14143-1 [9], the software application is characterized by functional user requirements. The FSM methods each define measures that assign numbers to software applications.

A goal in the definition of Function Point Analysis was to define the method independent of the technology used for implementation. FSM is therefore formulated without

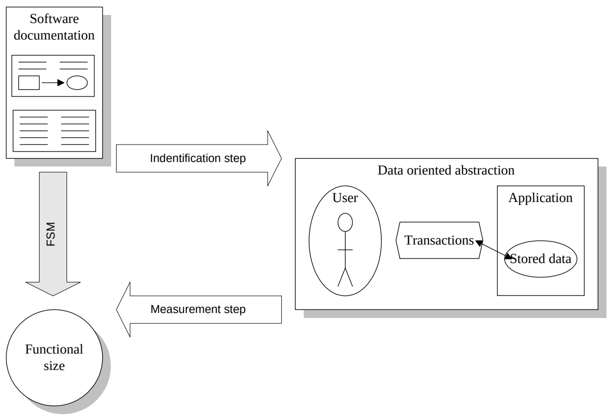


Figure 2. Functional size measurement (FSM) requires two steps of abstraction.

reference to any particular development method. In consequence, FSM requires two steps of abstraction.

2.1. Abstraction steps in FSM

Instead of using the concepts and models of a particular development method, FSM methods define their own concepts for the representation of a software application. The FSM methods thus define an abstraction of software that represents the items deemed relevant for functional size. The abstraction used in the methods covered in this paper can be characterized as *data oriented*. The FSM methods studied thus define the following two steps of abstraction:

1. The software documentation is represented in a data oriented abstraction.

2. The items in the data oriented representation are mapped into numbers.

The first step of abstraction is applied to the software documentation, regardless of its form. In terms of ISO 14143-1 [9], the source of the first step is the documentation of functional user requirements. The standard does, however, not prescribe a format for this documentation. Any FSM method must define how the abstraction has to be obtained, independently of the development method used.

Thus, FSM achieves independence of the technology used for implementation. The result is a representation in the data oriented abstraction of the particular method, that contains the items deemed relevant for functional size. Because of the independence of any formal documentation, this step requires interpretation of rules by humans.

The second step is the actual measurement, the mapping into numbers. Because the source in this step must be in the form of the data oriented abstraction, this step can in general be automated.

Our view on the two steps of abstraction is illustrated in Figure 2. Both steps are defined by the rules in the respective documents that define the FSM methods. In these documents, the two steps of abstraction are typically not separated clearly. The abstractions underlying the methods are not presented explicitly in most method definitions. However, COSMIC FFP defines two phases—mapping and measurement—that correspond closely to the two steps of abstraction. In the following section, we give a description of the data oriented abstraction. The detailed generalized representation is presented in Section 3.

2.2. The data oriented abstraction

Although not described explicitly, Albrecht's original approach introduces the basic concepts of the data oriented abstraction. The FSM methods discussed here have been proposed as improvements over the original FPA. These methods differ in both steps of abstraction, i.e., they differ both in their representation of software functionality and in the measure functions. However, IFPUG FPA, Mark II FPA and the FFP approach rely on the same core concepts (cf. Fig. 3):

- **User concept**. The users interact with an application. Users are not necessarily restricted to human users, but may include software and hardware "users".

- **Application concept**. The application is the object of the measurement. Applications provide functions to the users. These functions are the attribute of interest.

- **Transaction concept**. Transactions are processes of interaction of the user with the application from a "logical" or "functional" perspective.

- **Data concept**. Data is stored by the application. **Data elements** represent the smallest data items meaningful to the user. Data elements are structured in logically related groups similar to tables in a database.

- **Type concept**. Multiple instances of elements identified by the above concepts are considered as a single type.

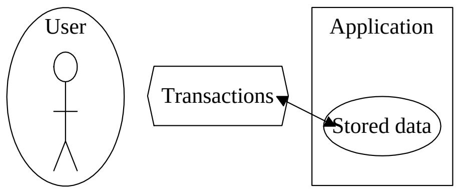


Figure 3. The data oriented abstraction.

All FSM methods studied represent an application by a set of transaction types and a set of groups of stored data. The methods differ, however, in the detailed views of these concepts. For example, IFPUG FPA classifies transactions into three classes, while the other methods do not define classes of transactions. Furthermore, the attributes used to characterize transactions differ in detail, e.g., an error message that is displayed to the user may be considered as an input in IFPUG FPA 4.0, while it is considered as an output in other methods. The methods also use different names for the core concepts.

As a consequence, it is difficult to analyze and discuss the differences and similarities between the methods. We therefore propose a generalized view of the data oriented abstraction that allows us to represent each of the methods in a uniform way. The generalized representation is described in the following section.

3. Generalized representation

The functionality of an application is represented by a set of transaction types and a set of data group types in the FSM methods studied. We therefore present generalizations for these two core concepts in the following two paragraphs. In the remainder of this section, we demonstrate how the generalization relates to the FSM methods.

3.1. Generalized representation of data groups

The data concept recognizes data elements as elementary items. A data group type is a set of data elements stored by the application. Sub-groups may be defined on the data elements of a data group type. This characterization applies directly to IFPUG FPA and FFP 1.0. In Mark II FPA, the data elements in a data group may be ignored. In COSMIC-FFP, sub-groups are not considered. Hence, the generalization introduces additional information only for Mark II FPA and COSMIC-FFP. The generalized view of a data group type is illustrated in Figure 4.

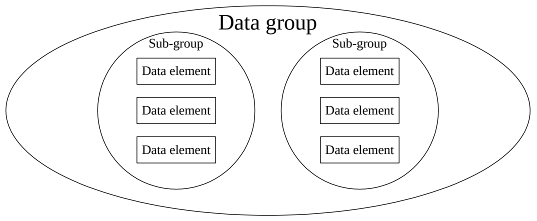


Figure 4. A data group type is a set of data elements, with optional sub-groups.

3.2. Generalized representation of transactions

Transaction types are represented very differently in the methods. IFPUG FPA defines three classes of transaction types with up to four attributes, Mark II FPA uses a single representation with three attributes, and the FFP approach defines a transaction type as a collection of sub-processes. The spectrum of logical activities associated with transaction types, however, is nearly the same in the three variants. Similarly to the sub-process concept in the FFP approach, we represent transaction types here with seven classes of logical activities that manipulate data elements:

1. **Entry activity**. The user enters data elements into the application.
2. **Exit activity**. Data elements are output to the user.
3. **Control activity**. Control information data elements are entered by the user.
4. **Confirm activity**. Confirmation data elements are output to the user.
5. **Read activity**. Data elements are read from a stored data group type.
6. **Write activity**. Data elements are written to a stored data group type.
7. **Calculate activity**. New data elements are calculated from some data elements.

The definitions of the logical activity classes are rather brief, because we do not intend to propose a new method for the identification of items here. Nevertheless, these activities can be found in the identification rules of the methods, and we use logical activities to represent the elements identified with those rules.

Given the concept of logical activities, a transaction type can be characterized as a collection of logical activities.

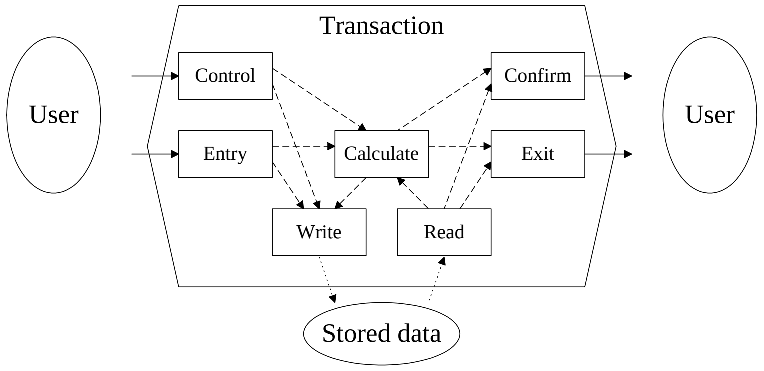


Figure 5. Transaction types are represented with logical activities.

The logical activities thus represent the transaction types in more detail than the methods originally do. As a result, the identification rules of each method have to be augmented with rules to map the transactions identified with the original rules onto the generalized transaction types. We describe these mappings in Section 3.3 to 3.7 below.

Figure 5 illustrates the generalized view on transaction types. Input from the user is indicated by straight arrows that enter the transaction, and lead to Entry and Control activities. Output, on the other hand, is sent to the user by Exit and Confirm activities. Dotted arrows depict access to stored data with Read and Write activities. The data flow internal to the transaction is depicted with dashed arrows.

As mentioned before, the concept of logical activities is very similar to the notion of sub-processes in the FFP approach, and Entry, Exit, Read, and Write sub-process classes are used in that FSM method. The generalized representation introduces three additional classes mainly required for the representation of IFPUG FPA.

IFPUG FPA and FFP 1.0 distinguish between data and control information in regard to input and output. In our generalized view, an input of data is represented by Entry activities, while control information input is represented by Control activities. On the output side, control information, e.g., error and confirmation messages, are distinguished from data output as well. Therefore, Exit activities represent data output and Confirm activities represent control information output. Neither Mark II FPA nor COSMIC-FFP 2.x make this distinction between data and control information. In general, these latter methods recognize both types of information as input or output, respectively.

Calculate activities are relevant as a criterion for classification in IFPUG FPA. Neither Mark II FPA nor the FFP approach classify transactions, and Calculate activities are not (yet) recognized in their respective measure functions. The internal data flow depicted by dashed arrows in Figure 5 and in the examples below illustrates the interpretation that can be associated with a transaction. It is, however, not taken into account by any of the methods discussed here.

3.3. Mapping for IFPUG FPA 4.0

3.3.1 Data groups in IFPUG FPA 4.0

As mentioned above in Section 3.1, mapping the IFPUG FPA 4.0 data group type to the generalized representation of a data group is straight forward: data element types correspond to data elements and record element types correspond to sub-groups of data elements.

The class of a data group is not represented as an attribute of the data group, i.e., we do not consider the classification of data groups into Internal Logical Files (ILF) and External Interface Files (EIF) as a part of the first step of abstraction. In fact, this classification can only be made in the context of an application, because IFPUG FPA determines the class of a data group type with the following identification rules [7, p. 5-6]:

> ILF Identification Rules
> (. . .)
> The group of data **is maintained within** the application boundary.
> (. . .)
>
> EIF Identification Rules
> (. . .)
> The group of data **is not maintained** by the application being counted.
> (. . .)

Hence, the class of a data group type in the context of a particular application is ILF, if and only if at least one transaction of that application does write to this particular data group. Otherwise, it must be classified as an EIF.

Therefore, the classification is a part of the second step of abstraction, i.e., the classification can be calculated with the measure function from the application context.

3.3.2 Transactions in IFPUG FPA 4.0

In IFPUG FPA, transactions are classified as either External Inputs (EI), External Outputs (EO), or External Inquiries (EQ). The identification rules are used to classify the transactions. The rules are also used to identify the attributes that characterize each transaction, i.e., the data element types (DET) and file types referenced (FTR). Each class of transactions and the relevant items are identified with a different set of rules. The following paragraphs map these rules to the generalized representation with logical activities. The items relevant for each class are represented by the logical activities.

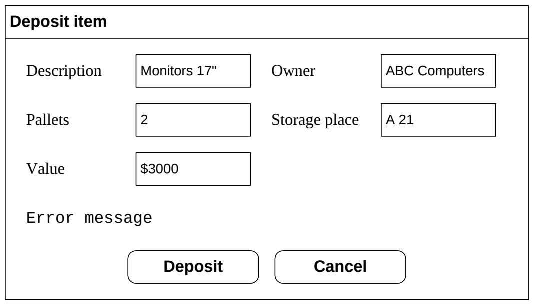
Note that the class of a transaction is not a part of the generalized representation, but the classification in IFPUG FPA 4.0 can be derived from the logical activities that play a role in a particular transaction. The classification with logical activities is explained in the last paragraph of this section.

External Inputs in IFPUG FPA 4.0 The identification rules for External Inputs (EI), including the rules for the identification of FTR and DET, relate to logical activities as follows:

- Data is received from the outside. This is represented with an Entry of data elements from the user.
- Data in an ILF is maintained. This presents a Write activity to a data group. The data elements written are identified. More than one data group may be updated, represented by a Write activity for each data group.
- Data groups may be read, represented by Read activities.
- Error and confirmation messages may be output, these data elements are identified. We represent this with Confirm activities.
- The user may specify information that controls the behavior of the application. This information is represented separately from data Entry by Control activities. The data elements are identified.
- Data elements additional to the data entered may be written to an ILF. The determination of these data elements may be represented with Calculate activities.

Output of data elements other than error and confirmation messages are not considered as a part of transactions in IFPUG FPA 4.0. Therefore, Exit activities do not appear in the representation of EI in IFPUG FPA, although a transaction might include these activities in its requirements.

As an example for a transaction that would be classified as an EI, consider the Deposit item transaction depicted in


Deposit item

Description	Monitors 17"	Owner	ABC Computers
Pallets	2	Storage place	A 21
Value	$3000		

Error message

Figure 6. User interface of the Deposit item transaction.

Figure 6. A customer deposits an item in a warehouse and the transaction registers attributes of the item, its owner, and storage place (see [5] for a full description of the examples). The following logical activities represent the transaction:

- An Entry of the data elements that describe the item: Description, Pallets, Value, Owner and Storage place.
- A Read of the Name from the Customer data group to verify that the Owner is registered.
- A Read of the Location and Space from the Place data group to verify that the required amount of space is available.
- A Read of the Description from the Item data group to prevent storage of an item under an existing name.
- A Calculate activity that determines the current date, which is stored as the Storage date.
- A Write of Description, Pallets, Value, Owner, Storage date and Storage place data elements to the Item data group.
- A Confirm to output any error messages.

Figure 7 presents an illustration of the representation of this transaction with the logical activities described above. IFPUG FPA does not require all the details of these logical activities. Nevertheless, these activities do represent information important for the view in IFPUG FPA:

- The data Entry is required by the rules of an EI.
- The Read activities must be considered to correctly determine the file types referenced (FTR).
- The Write activity and the data elements (DET) are written to the data group are essential for the EI.
- The Confirm activity must be considered to correctly determine the data element types (DET).

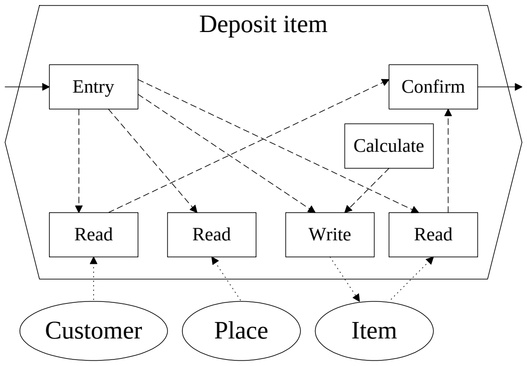


Figure 7. Logical activities in the Deposit item transaction.

External Outputs in IFPUG FPA 4.0 As with EI, we relate the identification rules for External Outputs (EO), including the rules for FTR and DET, to logical activities:

- Data is sent to the outside, represented as an Exit of data elements. These data elements are identified.
- Data elements may be read from data groups, represented as Read activities.

User input of data elements is not considered in the rules, although it might be relevant, e.g., as selection criteria for a report EO. Hence, Entry or Control activities do not appear in this view. Neither do the rules consider confirmation and error messages as output, excluding Confirm activities from the representation. Write activities are not permissible for EO. Data elements output may be the result of Calculate activities. Derived data is, in fact, the criterion used to distinguish between EO and EQ.

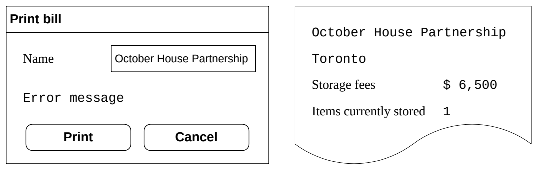


Figure 8. User interface of and report produced by the Print bill transaction.

The Print bill transaction depicted in Figure 8 is an example for an EO. The bill for a customer identified by name is printed. The following logical activities are required for this transaction, illustrated in Fig. 9:

- The name of the customer is specified with an Entry activity by the user.
- A Read of Name, Address, and Amount due from the Customer data group retrieves data elements for the bill.
- A Read of Owner names from the Item data group is required to obtain the number of items owned by the customer.
- A Calculate activity determines the total number of items from the Owner data.
- Error messages are output by Confirm activities, if necessary.
- An Exit activity outputs the data elements Name, Address, Amount due, and Total items.

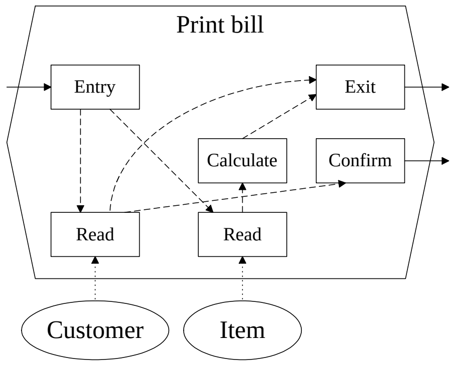


Figure 9. Logical activities in the Print bill transaction.

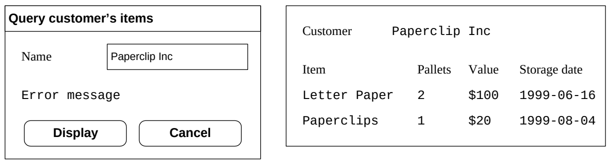
Here, the Total items data element is not retrieved by a Read activity, but it is calculated from other data elements. Total items is thus "derived data", and therefore, Print bill cannot be classified as an EQ and must be classified as an EO. Hence, the Calculate activity in the generalized representation is relevant for classification in IFPUG FPA 4.0.

While IFPUG FPA 4.0 ignores the Entry of selection criteria, the Read activities are necessary to correctly determine the file types referenced by this transaction. Although the error message may be output to the user, it is not considered by the rules for EO. Hence, the distinction of Exit and Confirm activities is also relevant for the accurate representation of an EO.

External Inquiries in IFPUG FPA 4.0 The External Inquiry (EQ) transaction class is more complex than the two previously discussed classes as it comprises both an input and an output side. However, the rules allow a distinction to which side of an EQ a logical activity contributes. Hence, in the generalized data oriented abstraction, the representation of EQ is not different from the representation of EI or

EO. The input and output sides appear in the formulation of the measure function in the formalized representation. The identification rules are represented as follows:

- Input data elements enter from the outside. This is represented by an Entry activity. The data elements entered are identified.
- Output results exit the application, which is represented with an Exit activity. The data elements are identified.
- Data is retrieved from data groups. This presents Read activities.
- The retrieved data does not contain derived data. Hence, the data elements output are obtained by Read activities, but not with Calculate activities.
- Error and confirmation messages may be output. This is represented by Confirm activities. Data elements are identified.
- The user may specify information that controls the behavior of the application. This information is represented separately from data Entry by Control activities. Data elements are identified.



Name: Paperclip Inc

Error message

Customer: Paperclip Inc

Item	Pallets	Value	Storage date
Letter Paper	2	$100	1999-06-16
Paperclips	1	$20	1999-08-04

Figure 10. User interface of the Query customer's items transaction.

Consider the following example shown in Figure 10: the Query customer's items transaction. With this transaction, the Items owned by a Customer are displayed. The logical activities required are the following, see Fig. 11:

- An Entry of the customer name by the user.
- A Read from the Customer data group to verify that the customer exists.
- An error message is output with a Confirm activity if the customer does not exist.
- A Read activity retrieves the Description, Pallets, Value, Storage date, and Owner from the Item data group for the items owned by the customer.
- An Exit activity displays the customer Name and a list of the items with their
- A Read activity retrieves the Description, Pallets, Value, and Storage date.

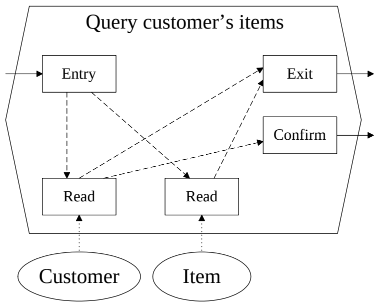


Figure 11. Logical activities in the Query customer's items transaction.

For the input side, the Entry and Confirm activities determine the data element types. On the input side, the Customer data group is referenced (Read) to verify the name entered. On the output side, the Exit activity represents the data element types. To generate the output, the Item data group is an additional file type referenced, represented by the second Read activity.

Classification with logical activities As we have seen in the preceding paragraphs, the three classes of transactions can be represented with the seven logical activities of the generalized data-oriented abstraction. However, in a particular transaction class, certain activity classes may be required or may be prohibited. We observe:

- Write activities may only appear in EI.
- An EI may also receive Control input that does not result in data Exit.
- Both an EO and an EQ output data via Exit activities.
- The data sent by Exit activities of an EQ must not contain data elements that are derived by Calculate activities. Otherwise, the transaction is classified as an EO.

Hence, we can determine the class of a particular transaction from the classes of activities that are used to represent the transaction. This does actually not come as a surprise, because the identification rules used in the classification of transactions are in part represented by logical activities, as we described in the preceding paragraphs. Nevertheless, transactions identified with IFPUG FPA 4.0 can therefore be represented in generalized representation that does not include transaction classes.

As with data groups, classification of transactions is therefore a part of the second step of abstraction, i.e., it is part of the measure function.

3.4. Mapping for IFPUG FPA 4.1

The changes in release 4.1 of IFPUG FPA are not fundamental in relation to release 4.0. However, both the identification rules and the measure function have been changed. In respect to transactions, these modifications have a significant impact on the representation in the generalized data oriented abstraction. We will discuss these changes here.

3.4.1 Data groups in IFPUG FPA 4.1

The concept of a data group type has not been changed in IFPUG FPA 4.1. The formulation of the identification rules has been simplified without change to the notions of an ILF or an EIF. In IFPUG FPA 4.1, the classification of data group types depends on the application context, as it does with IFPUG FPA 4.0. The generalized abstraction therefore applies directly to IFPUG FPA 4.1, as it did apply to IFPUG FPA 4.0 (cf. Sec. 3.3.1).

3.4.2 Transactions in IFPUG FPA 4.1

The overall concept of transaction types has not been changed in release 4.1. IFPUG FPA 4.1 still classifies transactions into the three classes EI, EO, and EQ. The detailed identification rules, however, have been modified. A new concept of *primary intent* has been introduced. The items that contribute to the weights have also been changed in several details.

The representation of transactions with logical activities is principally the same with IFPUG FPA 4.1 as with release 4.0. The changes in the rules have to be reflected mainly in the measure function.

The classification of transactions, however, cannot entirely be derived from the logical activities that represent a transaction, because the logical activities permissible in transaction classes have been extended and classification requires the new concept of primary intent in IFPUG FPA 4.1. The primary intent of a transaction can only be determined as an additional element by the analyst in the first step of abstraction, as either one of:

- alter the behavior of the system,
- write to stored data groups, or
- output information to the user.

Hence, the data oriented abstraction of IFPUG FPA 4.1 is extended by this element for each transaction.

Nevertheless, the generalized representation can still be used to represent the relevant information needed in IFPUG FPA 4.1, provided we add the primary intent for each transaction. We have, however, not included this element in the representation presented in Section 3.2, because it is not relevant for any of the other methods, and because it is a concept external to the view of transactions shared by all methods studied here.

The examples presented in Section 3.3.2 are represented with the same logical activities in IFPUG FPA 4.1. The primary intent of the Deposit item transaction is the Write to the Item data group type. Deposit item would therefore be classified as an EI.

The primary intent of the Print bill and Query customer's items transactions is output to the user. For these two transactions, classification into EO and EQ has to be derived from the logical activities. The result of the classification is here the same as with IFPUG FPA 4.0.

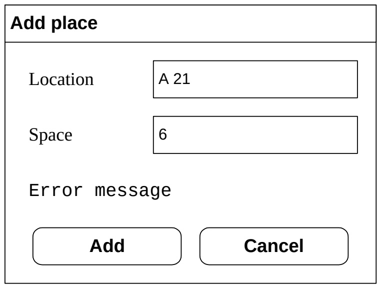
3.5. Mapping for Mark II FPA

3.5.1 Data groups in Mark II FPA

In Mark II FPA, data groups appear as entity types that do not contribute to functional size. Therefore, it is not necessary to identify data elements and sub-groups of data elements in Mark II FPA. On the other hand, data elements are used to characterize transactions and thus data elements are an item of the data oriented abstraction of Mark II FPA. Hence, data elements are not foreign to Mark II FPA and the representation of entity types can be extended with data elements and sub-groups of data elements. The measure function of Mark II FPA simply assigns a zero value to data groups.

3.5.2 Transactions in Mark II FPA

Transactions are called Logical Transactions in Mark II FPA. A Logical Transaction is regarded as a unit of input, processing and output. The input part corresponds to Entry and Control activities, and output corresponds to Exit and Confirm activities. The processing part actually represents access to entity types and therefore corresponds to Read and Write activities. Mark II FPA does not recognize aspects


Add place

Location: A 21

Space: 6

Error message

Figure 12. User interface of the Add place transaction.

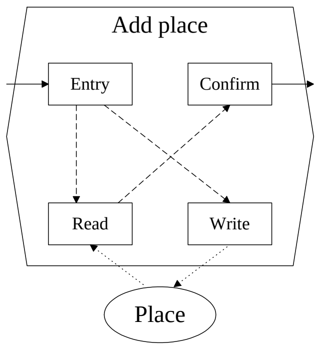


Figure 13. Logical activities in the Add place transaction.

in Logical Transactions that would correspond to Calculate activities.

As an example, let us consider the Add place transaction shown in Figure 12. The input part of the transaction consists of an Entry from the user, comprising the data elements Location and Space. The processing part includes a Read activity that tests whether the Place already exists, and a Write activity to store the new record. The output part, finally, consists of a Confirm activity that outputs an error message to the user if the Place already existed. Figure 13 illustrates the representation of the Add place transaction with logical activities.

3.6. Mapping for FFP 1.0

The FFP approach version 1.0 has been formulated as an extension to IFPUG FPA 4.0, where part of an application—designated in FFP 1.0 as “management function types”—is covered by IFPUG FPA, and the remaining part of “control function types” is covered by the FFP approach. Nevertheless, the FFP approach can be regarded as an FSM method in its own right. In this section, we only discuss the FFP concepts. Given an indication which method is to apply to a particular element, it is also possible to represent a mixture of IFPUG FPA and FFP in the generalized representation following the original proposal of FFP 1.0.

3.6.1 Data groups in FFP 1.0

Data group types in FFP 1.0 are defined analogously to IFPUG FPA 4.0. The identification rules are practically identical with IFPUG data group types, in respect to the identification of data elements, and sub-groups, and in respect to classification. Therefore, the mapping of data groups described in Section 3.3.1 applies to FFP 1.0 as well.

However, FFP 1.0 introduces a new class of data groups: single occurrence groups. These data groups comprise all data elements which only have a single instance in the application. Still, the single occurrence group can be represented by a generalized data group, as it consists of a set of data elements where no sub-groups are defined.

The classification of single occurrence data groups into read-only and read/write is given by the application context, as with the conventional “multiple occurrence” data groups. Hence, there is only one single occurrence data group in the generalized representation of an application. Of course, the measure function of FFP 1.0 assigns different values to single occurrence groups than to multiple occurrence groups.

3.6.2 Transactions in FFP 1.0

FFP represents transactions as collections of sub-processes. Four classes of sub-processes are defined for entry and exit of data elements from and to the user, and for read and write from and to stored data groups. The representation of sub-processes with logical activities is quite obvious. However, FFP 1.0 does not have different sub-process classes for data and control information. Even when applied independently from IFPUG FPA, any input would be handled by entry sub-processes, and any output by exits.

The generalized representation with its Control and Confirm activities thus provides for more detail that is simply ignored in the FFP measure function, where Control is equivalent to entries and Confirm equivalent to exits. In this sense, the logical activities of the examples presented in the previous sections represent FFP sub-processes.

3.7. Mapping for COSMIC-FFP 2.x

COSMIC-FFP 2.0, as opposed to FFP 1.0, has been published as a FSM method that is applied to all parts of an application, i.e., without reference to IFPUG FPA. In respect to the generalized representation of functional size, there is no difference between versions 2.0 and 2.1 of COSMIC-FFP. We therefore present the mapping for both versions here.

3.7.1 Data groups in COSMIC-FFP 2.x

In COSMIC-FFP, Data groups do not contribute to functional size, in contrast to FFP 1.0. Nevertheless, data groups are essential in the identification of sub-processes, because a sub-process may handle only one data group. COSMIC-FFP even extends the notion of data groups from stored data that is the subject of Read and Write activities to “transient” data groups that are used to identify Entry and Exit sub-processes. However, these transient data groups are not represented in the generalized representation as data group types, because these transient data groups are an attribute in the corresponding input and output activities.

As with Mark II FPA, the measure function of COSMIC-FFP assigns a zero value to data groups in the generalized representation.

3.7.2 Transactions in COSMIC-FFP 2.x

The view on transactions in COSMIC-FFP is essentially the same as in FFP 1.0: Transactions are represented by sub-processes that are classified as either Entry, Exit, Read or Write. Apart from the naming conventions, the differences lie in the details of the identification rules of the FFP versions. Therefore, sub-processes of COSMIC-FFP 2.x can be represented in the generalized representation in the same manner as with FFP 1.0 (cf. Sec. 3.6.2).

Note that COSMIC-FFP provides a mechanism for custom extensions that allows among others the definition of an equivalent to Calculate activities, described as “manipulation sub-processes”. Calculate activities are, however, not (yet) a part of the official COSMIC-FFP method.

3.7.3 Layers in COSMIC-FFP 2.x

A concept not present in the other FSM methods discussed has been introduced in COSMIC-FFP: software layers. With this concept, the application functionality can be partitioned into layers at different levels of abstraction, e.g., device drivers, graphical user interfaces, and application data management.

According to COSMIC-FFP, one layer acts as a “client” of another layer. In terms of the user and application concepts of the data oriented abstraction (cf. Fig. 3), a “client layer” is a user from the point of view of another layer. As a consequence, interfaces that would otherwise be internal to the application can be included in the abstraction of the software. Sub-processes that move data between the layers are included in the abstraction. Hence, the data oriented abstraction is applied with a higher level of granularity, yet the concepts used to represent the layer functionality are the same used to represent application functionality. Therefore, the generalized abstraction applies to COSMIC-FFP 2.x, including the layer concept.

4. Applications of the generalization

In Section 3, we have proposed a generalized representation for the first step of abstraction of a number of FSM methods. In our approach, this representation is the basis of a model for the FSM measurement process. Hence, the immediate result of this proposed representation is that we have formulated the basis of a model that allows to analyze and discuss FSM methods. As the generalization represents the first step of abstraction, we can discuss in detail whether this abstraction is adequate for FSM. With a formalization of the generalized representation, we can also discuss the measure functions. The generalized representation is then necessary to give an interpretation to observations obtained with the formalization.

Furthermore, the representation proposed is generalized, i.e., it applies to the abstraction of each of the FSM methods studied. Note that we do not assume that in all cases each method will represent a given application with the same transactions and data groups (although we believe that this is not unusual, cf. [5] for examples). With the layer concept introduced in COSMIC-FFP, e.g., transactions can be identified that would not be relevant in other methods. However, we believe that the generalized representation can be used to identify cases where the methods do or do not arrive at the same abstraction, and that this knowledge is relevant for understanding FSM methods. Nevertheless, the generalized representation allows direct comparisons of the methods and analysis of their differences. Obviously, the concepts underlying the different methods are not so different and one might assume that the effort required for the identification step does not vary dramatically between the methods.

As mentioned above, based on the generalized representation, a formalization can be given such that the measurement step has the form of mathematical functions. This approach has two benefits: on the one hand, we can study the measures, on the other hand, we can use the representation for automation and the representation of experience data. We discuss these aspects in the following two paragraphs.

4.1. Generalized Function Point Structure

We call the formalization of the generalized representation a generalized Function Point Structure. In the generalized Function Point Structure, an application $\mathbf{a}$ is a vector of transaction types $\mathbf{t_i}$ and stored data group types $\mathbf{f_j}$:

$$\mathbf{a} = (\mathbf{t}_1, \ldots, \mathbf{t}_\tau, \mathbf{f}_1, \ldots, \mathbf{f}_\sigma).$$

Here, the $\mathbf{t_i}$ are each a vector of logical activities and the $\mathbf{f_j}$ are sets of data elements with sub-groups. In the case of IFPUG FPA 4.1, the $\mathbf{t_i}$ also comprise the primary intent of the transactions. Each FSM method defines a measure function FPM such that $\mathrm{FPM}(\mathbf{a})$ is the functional size of application $\mathbf{a}$.

With the generalized Function Point Structure, we can formulate assumptions about functional size and test those assumptions with the FSM measures. For example, let us assume that we have two applications $\mathbf{a}$ and $\mathbf{a}'$, and we have a view of functional size that implies that $\mathbf{a}$ is larger than $\mathbf{a}'$. A measure then assumes our view of functional size if $\mathrm{FPM}(\mathbf{a}) > \mathrm{FPM}(\mathbf{a}')$. With the generalized Function Point Structure, we can formulate such properties of FSM methods axiomatically, i.e., we describe properties of FSM measures in general, instead of by example. In [6], we have demonstrated two such properties that are significant for the use of FSM methods for prediction of other variables.

4.2. Experience data and automation

A practical difficulty with the different FSM methods is that results obtained with one method cannot be compared directly with the results of another method. The practical use of functional size measurement data generally requires experience data from other, previous projects, and such data is coupled with the FSM method used. It is not possible to truly convert measurement values obtained with one method into values that would have been measured with another method. This observation can easily be derived from the measure definitions in the generalized Function Point Structure. Hence, the experience data cannot be used with another method in a meaningful way.

However, the generalized representation defines a set of elements that are sufficient to obtain the measurement with any of the FSM methods covered. Experience data stored in a form equivalent to the generalized representation can therefore be used to obtain measurement results for any of the methods studied here.

Furthermore, the formalized measure functions can be calculated automatically. Under the assumption that for a given application each FSM method arrives at the same abstraction regarding transactions and data groups, all FSM measures can be calculated automatically from a single source.

5. Conclusions

In this paper, we have proposed a model for a number of existing FSM methods that gives a new view on these methods. We use an axiomatic approach based on measurement theory, that allows us to formulate assumptions of reality based on the model.

The basis of our model is a representation of the data oriented abstraction of the FSM methods. This representation is generalized such that it applies to each of the methods studied. The generalized representation allows direct comparisons of the methods and analysis of their differences.

A formalization of the generalized representation in a generalized Function Point Structure allows the formulation of FSM measures as functions. Axioms can be formulated in the generalized Function Point Structure as assumptions of reality. The generalized representation relates these axioms to the elements defined by the FSM methods.

The formalization may also be used as a basis for the representation of experience data from previous projects. Furthermore, the mathematical formulation of the measure functions allows automation of the measurement step.

The detailed definitions of the generalized Function Point Structure and of the formalized measure functions have to be presented separately.

References

[1] A. Abran, J.-M. Desharnais, S. Oligny, D. St-Pierre, and C. Symons. *COSMIC-FFP Measurement Manual*. Common Software Measurement International Consortium, Oct. 1999. Version 2.0.

[2] A. Abran, J.-M. Desharnais, S. Oligny, D. St-Pierre, and C. Symons. *COSMIC-FFP Measurement Manual*. Common Software Measurement International Consortium, May 3, 2001. Version 2.1.

[3] A. Abran and P. N. Robillard. Function points: A study of their measurement processes and scale transformations. *Journal of Systems and Software*, 25(2):171–184, May 1994.

[4] A. J. Albrecht. Measuring application developement productivity. In *IBM Applications Development Symposium*, pages 83–92, Oct. 14–17, 1979.

[5] T. Fetcke. The warehouse software portfolio: A case study in functional size measurement. Report 1999-20, Technische Universität Berlin, Fachbereich Informatik, 1999.

[6] T. Fetcke. Two properties of Function Point Analysis. In R. Dumke and F. Lehner, editors, *Software-Metriken: Entwicklungen, Werkzeuge und Anwendungsverfahren*, pages 17–34. Deutscher Universitäts Verlag, 2000.

[7] *Function Point Counting Practices Manual*. International Function Point Users Group, Westerville, Ohio, 1994. Release 4.0.

[8] *Function Point Counting Practices Manual*. International Function Point Users Group, Westerville, Ohio, 1999. Release 4.1.

[9] ISO/IEC 14143-1:1998(e) – information technology – software measurement – functional size measurement – definition of concepts. International standard, 1998.

[10] D. St-Pierre, M. Maya, A. Abran, J.-M. Desharnais, and P. Bourque. Full function points: Counting practices manual. Technical Report 1997-04, Software Engineering Management Research Laboratory and Software Engineering Laboratory in Applied Metrics, Sept. 1997.

[11] C. R. Symons. Function point analysis: Difficulties and improvements. *IEEE Transactions on Software Engineering*, 14(1):2–11, 1988.

[12] *Mk II Function Point Analysis Counting Practices Manual*. United Kingdom Software Metrics Association, Sept. 1998. Version 1.3.1.

[13] H. Zuse. *A Framework of Software Measurement*. de Gruyter, 1998.