

# A Generalized Structure for Function Point Analysis

Thomas Fetcke

## Abstract

Since its first publication by Albrecht, Function Point Analysis has been revised and modified several times. Today, a number of variants are in use, which differ in their respective views on functional size.

Function Point Analysis relies implicitly on a model of software. We propose the Function Point Structure as a formalization of the software model of Function Point Analysis. The Function Point count is then defined as a function on the Function Point Structure. Function Point variants differ in their abstract models of software as well as in their measure functions. Therefore, different formalizations of the Function Point Structure are required for each variant. We present here a generalized Function Point Structure for several data oriented variants of Function Point Analysis. With the generalized Function Point Structure, we can analyze the empirical assumptions made by the FPA variants and the implications on the prediction of other variables. We can also study the differences between the views and assumptions of the variants.

## 1 Introduction

Since its first publication by Albrecht, Function Point Analysis (FPA) has been revised and modified several times. Today, a number of variants are in use, which differ in their respective views on functional size. In this paper, we give a characterization of the measurement process of three variants of Function Point Analysis. In our view, Function Point Analysis defines both a model of software and a measure of “functional size” on this model. We therefore formulate the data oriented model of soft-

ware used implicitly by the FPA variants. The initial abstraction in Function Point Analysis is thus made explicit.

Given the data oriented abstraction of Function Point Analysis, the Function Point measures of the variants are functions into numbers. In the generalized Function Point Structure, we give a formalization of the data oriented software model. The Function Point measures can thus be formulated mathematically as functions. Given this formalization, empirical properties and assumptions made by the different variants of Function Point Analysis can be analyzed. For this analysis, we present two additional conditions for the weak orders assumed by Function Point measures. These conditions can give us additional insight into the behavior of Function Point Analysis. We present some experiments of thought in this paper to illustrate observations obtained with these criteria based on the generalized Function Point Structure.

In the following paragraph, we give a short overview of the evolution of the FPA variants studied.

### 1.1 Evolution of Function Point Analysis

Function Point Analysis was developed by Albrecht in the 1970s, with its first presentation to the public in 1979 [2]. The purpose of Function Point Analysis 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 Function Points as a measure of “functional size”.

In 1984, the International Function Point Users Group (IFPUG) was formed to maintain Albrecht’s

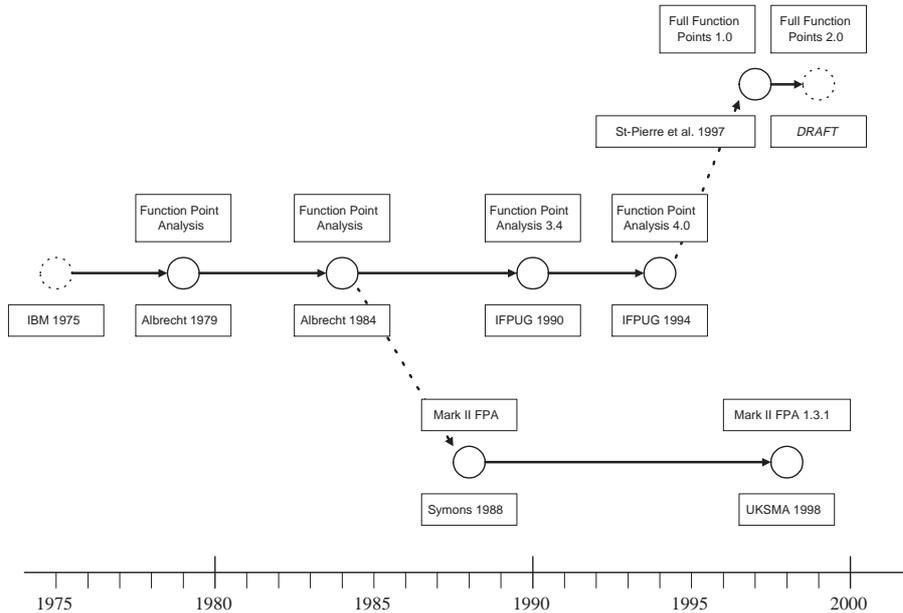


Figure 1: The evolution of three variants of Function Point Analysis.

FPA. IFPUG has since then published Counting Practices Manuals that give standard rules for the application of Function Point Analysis. 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 [6] 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 [7]. In 1997, St-Pierre et al. [5] proposed the Full Function Points approach as an extension to the IFPUG standard. The purpose of the extension was to capture the functional size of real-time applications.

In this study, we focus on three variants of FPA:

- the IFPUG standard as defined in the Counting Practices Manual 4.0 [4],
- Mark II FPA 1.3.1 defined in [7], and
- the Full Function Points approach 1.0, formulated in [5].

We will see that these variants share a core view of the items that determine functional size. Figure 1 presents a view of the evolution of the three variants of FPA studied here.

## 1.2 Related studies

Abran et al. [1] analyze the measurement process of IFPUG Function Point Analysis. In their view, FPA constructs the Function Point count<sup>1</sup> 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.

In this paper, we view Function Point Analysis as a measure that assumes an ordinal scale with the empirical objects in Function Point Analysis. The

<sup>1</sup>With the term Function Point *count*, we use the standard term of IFPUG for the FPA measure here to help readers familiar with this nomenclature. Note, however, that in our view FPA defines a measure function that maps applications into real numbers, rather than a “count”.

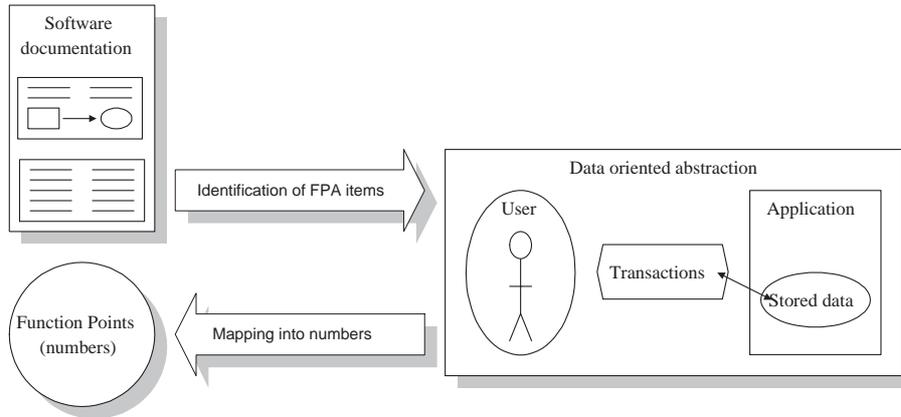


Figure 2: Function Point Analysis defines two steps of abstraction.

assumed weak order can then be examined with conditions or axioms that formulate assumptions of reality [8, Chapter 4]. We present two additional conditions to the weak order here.

## 2 Two steps of abstraction

Any measurement can be interpreted as an abstraction that focuses on some attributes of an object under study. In measurement theory, we view this abstraction as a mathematical function that maps empirical objects onto numerical objects. From this viewpoint, Function Point Analysis defines a function that assigns numbers to software applications.

Function Point Analysis has been declared as a measurement method that is independent of the technique used for implementation. It is formulated without reference to any particular development method. Therefore, instead of using the concepts and models of a development method, FPA introduces its own concepts for the description of a software application. FPA thus defines its own abstraction of software that represents the items deemed relevant for functional size. This abstraction is data oriented. Function Point Analysis thus actually defines two steps of abstraction:

1. The software documentation is represented in the 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. Thus, Function Point Analysis achieves independence of the technology used for implementation. The result is a representation in the data oriented abstraction that contains the items deemed relevant for functional size. This step requires the evaluation 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 be automated.

Our view of the two steps of abstraction is illustrated in Figure 2. Both steps are defined by the rules in the respective Counting Practices Manuals of the FPA variants. Although the rules make a distinction between the identification of FPA items on one hand, and the assignment of Function Points on the other hand, the two steps of abstraction are not separated clearly. The data oriented abstraction is not presented explicitly. In the following section, we give a generalized description of the data oriented abstraction.

## 3 Data oriented abstraction

IFPUG Function Point Analysis uses a data oriented abstraction of the application measured

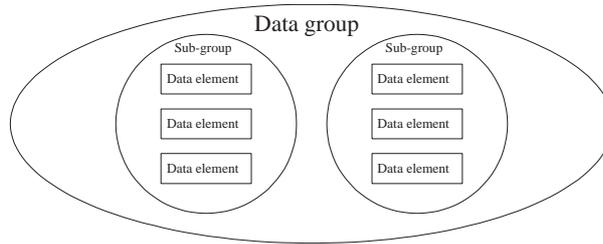


Figure 3: A data group type is a set of data elements.

(cf. Fig. 2). The variants of Function Point Analysis studied here have been proposed as improvements over the original approach. They therefore differ both in the items identified and in the measure function. However, both the Mark II and Full Function Points approaches rely on the same core concepts as the IFPUG standard. These core concepts are:

- **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. Transactions
  - are the smallest unit of activity meaningful to the user,
  - are “self-contained”, i. e., logically complete,
  - leave the application in a consistent state.
- **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.

While these core concepts are common to the three variants studied here, the detailed definitions

and names for these concepts and the identification procedures differ significantly. However, we do not present a discussion of these differences here. Instead, we will give a characterization of the core concepts that can serve as a basis for all three variants.

At the highest level of the data oriented abstraction, all three variants identify transaction types and data group types. The term *type* refers here to the principle that multiple instances of the same logical function are identified only once. The term *function type* refers both to transaction types and data group types.

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 the Full Function Points approach. In Mark II FPA, the data elements in a data group may be ignored.

Transaction types are represented very differently in the variants. The IFPUG standard defines three classes of transaction types with four attributes, Mark II FPA uses a single representation with three attributes, and the Full Function Points 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 Full Function Points approach, we represent transaction types here with six classes of logical activities:

1. **Entry activity.** The user enters data elements into the application.
2. **Exit activity.** Data elements are output to the user.

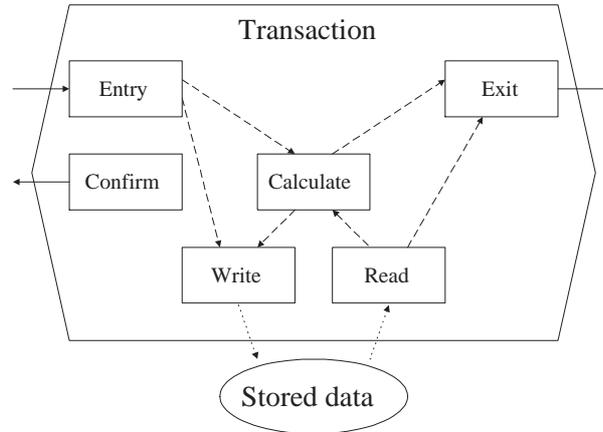


Figure 4: Transaction types are represented with activities.

3. **Confirm activity.** Confirmation data elements are output to the user.
4. **Read activity.** Data elements are read from a data group type.
5. **Write activity.** Data elements are written to a data group type.
6. **Calculate activity.** New data elements are calculated from some data elements.

Given the concept of logical activities, a transaction type can be characterized as a collection of logical activities. IFPUG Function Point Analysis, e. g., requires that an “external input”:

- receives data from the user (Entry),
- maintains data in a data group type (Write),
- may read data from other data group types (Read),
- may output confirmation or error messages (Confirm).

Mark II FPA views transaction types as a unit of input (Entry), processing (Read and Write) and output (Exit and Confirm). In the Full Function Points approach, the first five classes of logical activities correspond to the four classes of subprocesses.

With the concepts described before, we can represent the items identified by the different FPA variants in a uniform way. Thus, we have a generalized representation of the data oriented abstraction that is applicable to any of the three variants. Note that we do not require that all variants arrive at the same items for a given set of user requirements.

In terms of the two step mapping approach, the first step of abstraction is now a mapping into the generalized abstraction. This does, however, not present a change to any of the variants, as the logical activities of the generalized representation appear in all three variants. The same identification rules are used as before, merely the representation is more explicit. The resulting Function Point counts will not be affected.

## 4 Generalized Function Point Structure

As mentioned before, in measurement theory we view a measure as a function that assigns numbers to empirical objects. In FPA, the empirical objects are applications, characterized by functional user requirements. These requirements are represented in the data oriented abstraction discussed in the previous section. We will give a formalization of this representation in the generalized Function Point Structure in this section. The Function Point

count of each variant can then be defined as function on the generalized Function Point Structure.

#### 4.1 Portfolio concept

The universe of applications defined by some user requirements is virtually unlimited. However, at any given point in time, we will study only a finite set of applications. Such a set of applications may contain related applications, e. g., several applications that share one database. As an experiment of thought, a set of applications may also contain different versions of an application. The functions of the applications may overlap, i. e., we may find the same function types in more than one application.

We call such a set of applications a software portfolio. In the data oriented approach of Function Point Analysis, a software portfolio is formed by a finite set of transaction types and a finite set of data group types. We assume that these function types have been identified and that they are fixed for the study. In any given situation, we thus have a finite set of empirical objects measured. This notion simplifies the formalization without limiting the area of application, as no limit on the number of function types is imposed.

#### 4.2 Function Point structure

In the Function Point structure, the function types in a software portfolio are represented in the application closure. The application closure consists in principle of a collection of transaction types and a collection of data group types.

Function types are associated with some interpretation, e. g., addition of a customer record to a database. Any two function types differ in respect to their logical function. This distinction is made in the first step of abstraction, i. e., it is given by the detailed rules defining the respective Function Point Analysis variant. In the representation of the items and their attributes, we represent this logical distinction with an index in a vector. Hence, the application closure is a vector of function types, the position in the vector identifies the interpretation given to that function type. The order of indices in the application closure has no other meaning.

We define the application closure as a vector of  $\tau$  transaction types  $\mathbf{T}_i$  and  $\sigma$  data group types  $\mathbf{F}_j$ .

#### Definition 4.1 (Application closure)

An application closure  $\mathbf{H}$  is defined as the vector

$$\mathbf{H} = (\mathbf{T}_1, \dots, \mathbf{T}_\tau, \mathbf{F}_1, \dots, \mathbf{F}_\sigma).$$

Transaction types consist of a number of activities. The interpretation associated with an activity is represented as an index as well.

#### Definition 4.2 (Transaction type)

The transaction type  $\mathbf{T}_i$  is a vector of activities

$$\mathbf{T}_i = (\mathbf{P}_{i1}, \dots, \mathbf{P}_{in_i}).$$

An activity is characterized by four attributes:

- its class  $\theta_{ik} \in \{\text{Entry, Exit, Confirm, Read, Write, Calculate}\}$ ,
- for Read and Write activities, the data group type referenced  $r_{ik}$ ,
- the set of data elements  $D_{ik}$  handled,
- for Calculate activities, the set of data elements calculated  $C_{ik}$ .

#### Definition 4.3 (Activity)

An activity  $\mathbf{P}_{ik}$  is a quadruplet

$$\mathbf{P}_{ik} = (\theta_{ik}, r_{ik}, D_{ik}, C_{ik}).$$

Data group types are a collection of data elements. Each data element belongs to exactly one sub-group. A data group type is represented as a set of pairs, where each pair represents a data element and its sub-group.

#### Definition 4.4 (Data group type)

The data group type  $\mathbf{F}_j$  is a set

$$\mathbf{F}_j = \{(d_{j1}, g_{j1}), \dots, (d_{jr_j}, g_{jr_j})\}.$$

where the  $d_{jk}$  are data elements and the  $g_{jk}$  designate sub-groups.

#### 4.3 Applications in a software portfolio

Given a software portfolio, we can perform experiments of thought. An application is formed by a

subset of the function types in the portfolio. Furthermore, the function types in an application may be subsets of the function types in the portfolio. In the generalized Function Point Structure, we define an instance relation, such that applications are instances of the application closure. We write  $\mathbf{a} \sqsubseteq_A \mathbf{H}$ , if application  $\mathbf{a}$  is an instance of the closure  $\mathbf{H}$ .

**Definition 4.5 (Application instances)**

An application  $\mathbf{a} = (\mathbf{t}_1, \dots, \mathbf{t}_\tau, \mathbf{f}_1, \dots, \mathbf{f}_\sigma)$  is an instance of the application closure  $\mathbf{a} \sqsubseteq_A \mathbf{H}$ , with  $\mathbf{H} = (\mathbf{T}_1, \dots, \mathbf{T}_\tau, \mathbf{F}_1, \dots, \mathbf{F}_\sigma)$ , if the  $\mathbf{t}_i$  are instances of  $\mathbf{T}_i$  and the  $\mathbf{f}_j$  are instances of  $\mathbf{F}_j$ .

The instance relation can be extended to applications in general. An application  $\mathbf{a}$  is then an instance of  $\mathbf{a}'$  if all function types of  $\mathbf{a}$  are instances of the corresponding function types in  $\mathbf{a}'$ , written as  $\mathbf{a} \sqsubseteq_A \mathbf{a}'$ .

**4.4 Unification of applications**

Two applications  $\mathbf{a}$  and  $\mathbf{a}'$  can be combined to form a single new application that comprises the function types of both  $\mathbf{a}$  and  $\mathbf{a}'$ . We assume that the two applications may contain both distinct and overlapping function types. In the combined application, all function types of both applications will be present. However, those function types that overlap will be present only once in the form that comprises all their functionality, as required by the function *type* concept. Therefore, similar to the operations introduced in the analysis of measures for object-oriented software in [3, 8, Chapter 6], the combination of two applications into a single application can be expressed with a unification operation. In the generalized Function Point Structure, we define the unification of application instances as follows:

**Definition 4.6 (Instance unification)**

Let  $\mathbf{a}, \mathbf{a}' \sqsubseteq_A \mathbf{H}$  be two applications. The unification of applications  $\mathbf{a}, \mathbf{a}'$  is defined as

$$\mathbf{a} \cup_A \mathbf{a}' := (\mathbf{t}_1 \cup_T \mathbf{t}'_1, \dots, \mathbf{t}_\tau \cup_T \mathbf{t}'_\tau, \mathbf{f}_1 \cup_F \mathbf{f}'_1, \dots, \mathbf{f}_\sigma \cup_F \mathbf{f}'_\sigma).$$

The unified application is thus a vector of the unified function types of the two applications. For

brevity, we do not present the detailed definitions of the unification of transaction types  $\cup_T$  nor that of the unification of data group types  $\cup_F$  here.

**5 Empirical observations**

In the preceding section, we have characterized the empirical model of Function Point Analysis formally. With this characterization, we can study the empirical assumptions made by the different variants, and the differences between them.

Independent of the algorithm used to obtain the measurement values, any FPA variant defines an order on applications. The interpretation of this order in Function Point Analysis is that whenever the Function Point count FPC of an application  $\mathbf{a}$  is greater than the Function Point count of another application  $\mathbf{a}'$ , then  $\mathbf{a}$  is larger in functional size than  $\mathbf{a}'$ . We assume that the Function Point count defines an ordinal scale on applications, i. e., that the axioms of the weak order (transitivity and completeness) hold (cf. [8, Chapter 4]). We will now discuss conditions additional to the weak order.

**5.1 Dominance**

The axiom of dominance (cf. [8, pp. 318]) is a condition related to the instance relation. As an experiment of thought, let us assume that we make an extension to the function types of a given application. As the function types represent the notion of functional size, there is an empirical meaning to this extension, i. e., we have an expectation of the effect of such an extension on functional size.

A reasonable assumption would be that an extension of the functionality of an application should increase its functional size. If we relate this empirical assumption to other variables, e. g., the effort required to implement the functionality, this is a reasonable assumption.

The axiom of dominance requires that the weak order assumed by a Function Point measure agrees with the application instance relation, i. e., if application  $\mathbf{a}$  is an instance of application  $\mathbf{a}'$ , written as  $\mathbf{a} \sqsubseteq_A \mathbf{a}'$ , then  $\mathbf{a}'$  must be at least as large in functional size as  $\mathbf{a}$ . With the Function Point

**Query orders per customer**

Customer	<input type="text" value="ABC Computers"/>	Status	<input type="text" value="pending"/>
Begin	<input type="text" value="1999-03-05"/>	Category	<input type="text" value="Monitors"/>
End	<input type="text" value="1999-04-10"/>		

Customer			ABC Computers
Part no.	Part	Quantity	
56074	CTX 17"	20	
56087	CTX 19"	5	
59327	Eizo 17"	10	
59103	Eizo 19"	2	

Figure 5: Input and output of the order query ( $\mathbf{t}_Q$ ).

count FPC, we formulate the axiom of dominance as follows:

$$\mathbf{a} \sqsubseteq_A \mathbf{a}' \implies \text{FPC}(\mathbf{a}) \leq \text{FPC}(\mathbf{a}'). \quad (1)$$

Consider the following example. An order processing application maintains three data group types for customers, stored parts and orders. A query allows to view orders for a customer issued between a start and an end date. The user can also choose the status of the order and the category of the parts ordered. The output list is retrieved from the three data group types, the fields are customer name, part id, part description and quantity. Figure 5 illustrates input and output screens of this transaction type.

Suppose the query has been identified as a transaction type  $\mathbf{t}_Q$ . According to IFPUG FPA rules, this transaction type is an “external inquiry”, rated with six Function Points

$$\text{FPC}_T^{\text{IFPUG}}(\mathbf{t}_Q) = 6.$$

Now, an additional data element is added to the output, the total value of the order, the rest of the transaction type remains unchanged. We identify this as the transaction type  $\mathbf{t}'_Q$ . We therefore have  $\mathbf{t}_Q \sqsubseteq_T \mathbf{t}'_Q$ . The extension is illustrated in Figure 6.

According to IFPUG FPA, the total value data element must be calculated and hence the class of  $\mathbf{t}'_Q$  is “external output”. Now, we have

$$\text{FPC}_T^{\text{IFPUG}}(\mathbf{t}'_Q) = 4 < 6 = \text{FPC}_T^{\text{IFPUG}}(\mathbf{t}_Q)$$

although  $\mathbf{t}_Q \sqsubseteq_T \mathbf{t}'_Q$ , i. e., IFPUG Function Point Analysis violates the axiom of dominance (1).

In other cases, however, an additional data element may increase the IFPUG Function Point count. Therefore, we cannot predict in general the influence an extension of the function types has on the Function Point count according to IFPUG FPA rules. The consequence is that if we use the Function Point count for the prediction of development effort, for example, an extension to the functionality requested by the user may increase *or* decrease the predicted effort, which may be unexpected.

The Mark II and Full Function Points variants, on the other hand, assume the axiom of dominance (1), i. e., an extension cannot decrease the functional size measured. The extension does, however, not necessarily increase the Function Point count. This is due to the fact that all FPA variants abstract from some items identified in the generalized Function Point structure. In Mark II FPA, e. g., an extra data element in a data group referenced by an application would not change the result of the measurement. Nevertheless, an extension to the functionality requested by the user cannot decrease an effort estimate that is based on Mark II FPA or the Full Function Points approach.

With the dominance axiom (1), we have thus found a significant difference between the three FPA variants.

## 5.2 Monotonicity

We now consider another condition called monotonicity (cf. [8, p. 321]). The axiom of monotonicity requires that for any three applications  $\mathbf{a}, \mathbf{a}', \mathbf{a}''$ ,

Query orders per customer			
Customer	<input type="text" value="ABC Computers"/>	Status	<input type="text" value="pending"/>
Begin	<input type="text" value="1999-03-05"/>	Category	<input type="text" value="Monitors"/>
End	<input type="text" value="1999-04-10"/>		
<input type="button" value="OK"/>			

Customer    ABC Computers		
Part no.	Part	Quantity
56074	CTX 17"	20
56087	CTX 19"	5
59327	Eizo 17"	10
59103	Eizo 19"	2
Total value		
		\$31,000

Figure 6: Input and output of the extended order query ( $t'_Q$ ).

it holds

$$\begin{aligned} \text{FPC}(\mathbf{a}) \leq \text{FPC}(\mathbf{a}') &\iff \\ \text{FPC}(\mathbf{a} \cup_A \mathbf{a}'') &\leq \text{FPC}(\mathbf{a}' \cup_A \mathbf{a}''). \quad (2) \end{aligned}$$

Hence, if application  $\mathbf{a}$  is smaller in functional size than  $\mathbf{a}'$  according to the Function Point count, then an extension of both applications with the same function types  $\mathbf{a}''$  cannot have the result that the extended application  $\mathbf{a} \cup_A \mathbf{a}''$  is larger than the extended  $\mathbf{a}' \cup_A \mathbf{a}''$ .

Suppose, for example, that two applications  $\mathbf{a}, \mathbf{a}'$  are given which are almost equal in their functional size. Both operate on similar yet different instances of a data group type  $\mathbf{F}_E$  holding employee data. Let us assume that application  $\mathbf{a}$  merely references its data group type instance  $\mathbf{f}_E \sqsubseteq_F \mathbf{F}_E$ , while application  $\mathbf{a}'$  also updates its data group type instance  $\mathbf{f}'_E \sqsubseteq_F \mathbf{F}_E$ . The functional size of both applications has been calculated according to IFPUG FPA and the results are equal except for the respective instances of  $\mathbf{F}_E$ . For  $\mathbf{f}_E$ , the data elements of Table 1 have been identified.  $\mathbf{f}_E$  is classified as an “external interface file” in application  $\mathbf{a}$ .

In application  $\mathbf{a}'$ , the data group type  $\mathbf{f}'_E$  has the data elements listed in Table 2. It has been classified as an “internal logical file”.

We assume that the other function types of both  $\mathbf{a}$  and  $\mathbf{a}'$  contribute  $c$  Function Points to the mea-

surement value. Thus,

$$\begin{aligned} \text{FPC}(\mathbf{a}) &= c + \text{FPC}_F(\mathbf{f}_E) = \\ &= c + 5 < c + 7 \\ &= c + \text{FPC}_F(\mathbf{f}'_E) = \text{FPC}(\mathbf{a}'). \end{aligned}$$

Now, let us assume that we extend both applications with the same application functionality  $\mathbf{a}''$  that comprises an additional group of data elements  $\mathbf{f}''_E$  in the instances of  $\mathbf{F}_E$  listed in Table 3 and an additional transaction type  $\mathbf{t}_E$  for administrator edit of  $\mathbf{F}_E$  instances.

Now, both extended applications maintain the extended instances of  $\mathbf{F}_E$ , which are therefore both classified as “internal logical files”. With the new function types, we have

$$\begin{aligned} \text{FPC}(\mathbf{a} \cup_A \mathbf{a}'') &= \\ c + \text{FPC}_T(\mathbf{t}_E) + \text{FPC}_F(\mathbf{f}_E \cup_F \mathbf{f}''_E) &= \\ c' + 10 > c' + 7 & \\ = c + \text{FPC}_T(\mathbf{t}_E) + \text{FPC}_F(\mathbf{f}'_E \cup_F \mathbf{f}''_E) &= \\ \text{FPC}(\mathbf{a}' \cup_A \mathbf{a}'') &. \end{aligned}$$

Hence, although application  $\mathbf{a}$  is clearly less in functional size than  $\mathbf{a}'$  according to IFPUG FPA rules, the extension with an identical, known amount of functionality reversed the relation between the extended applications. Therefore, IFPUG Function Point Analysis violates the axiom of monotonicity (2). A consequence of this observation is that if an application has been developed in parts and the functional sizes of these parts have

Sub-group	Personal data	Job description	Hourly payment data	
Data elements	Name	Job name	Hourly rate	
	Birth date	Job level	Standard hourly rate	
	Social security number	Department		
	Address	Group		
	Number of dependents	Supervisor		
	Dependent name	Location		
	Dependent birth date			
	Dependent social security no.			
	Tax class			
	Benefits plan			
	Health care plan			
	Total	11	6	2

Table 1: The data group type  $\mathbf{f}_E$  has 19 data elements and three sub-groups.

been measured with IFPUG FPA, one can in general not predict the functional size of the whole application from the measurement values of the parts.

Mark II FPA, on the other hand, assumes monotonicity (2). An increase in the functional size of one part of an application — from  $\mathbf{a}$  to  $\mathbf{a}'$  in (2) — has a predictable influence on the functional size of the whole application — extended from  $\mathbf{a} \cup_A \mathbf{a}''$  to  $\mathbf{a}' \cup_A \mathbf{a}''$ .

The Full Function Points approach, version 1.0, also violates the axiom of monotonicity (2), because data group types are measured with the same rules as in IFPUG FPA. With the changes made in version 2.0, which is currently in draft status, however, the Full Function Points approach does assume monotonicity.

Sub-group	Personal data	Job description	
Data elements	Name	Job name	
	Birth date	Job level	
	Social security number	Department	
	Address	Group	
	Number of dependents	Supervisor	
	Dependent name	Location	
	Dependent birth date		
	Dependent social security no.		
	Tax class		
	Benefits plan		
	Health care plan		
	Total	11	6

Table 2: The data group type  $\mathbf{f}'_E$  has 17 data elements and two sub-groups.

Sub-group	Salary data
Data elements	Salary
	Hours per week
Total	2

Table 3: Two additional data elements in a new sub-group are defined in  $\mathbf{f}''_E$ .

We thus have characterized another property that gives us insight in the differences between the three FPA variants.

## 6 Conclusion

In this paper, we have characterized Function Point Analysis as a two step measurement process, defining both a data oriented abstraction of software applications and a measure function on this abstraction. We have proposed a generalized representation of this abstraction that applies to three variants of Function Point Analysis. A formalization of this generalized view has been proposed in

the generalized Function Point Structure.

We have thus introduced a new point of view on Function Point Analysis. Our characterization enables us to study the empirical assumptions made by the variants of Function Point Analysis and the properties of the different measures. We have presented two additional conditions to the weak order. These conditions have been used to gain new insights in the behavior of the FPA variants.

The generalized representation of the data oriented abstraction and its formalization in the generalized Function Point Structure can also serve as a basis for the comparison of the different FPA variants under controlled circumstances.

## References

- [1] Alain Abran and Pierre N. Robillard. Function points: A study of their measurement processes and scale transformations. *Journal of Systems and Software*, 25(2):171–184, May 1994.
- [2] A. J. Albrecht. Measuring application development productivity. In *IBM Applications Development Symposium*, pages 83–92, October 14–17 1979.
- [3] Thomas Fetcke. Softwaremetriken bei objektorientierter Programmierung. Diploma thesis, Techn. Univ. Berlin, April 1995.
- [4] *Function Point Counting Practices Manual*. International Function Point Users Group, Westerville, Ohio, 1994. Release 4.0.
- [5] Denis St-Pierre, Marcela Maya, Alain Abran, Jean-Marc Desharnais, and Pierre Bourque. Full function points: Counting practices manual. Technical Report 1997-04, Software Engineering Management Research Laboratory and Software Engineering Laboratory in Applied Metrics, September 1997.
- [6] Charles R. Symons. Function point analysis: Difficulties and improvements. *IEEE Transactions on Software Engineering*, 14(1):2–11, 1988.
- [7] *Mk II Function Point Analysis Counting Practices Manual*. United Kingdom Software Metrics Association, September 1998. Version 1.3.1.
- [8] Horst Zuse. *A Framework of Software Measurement*. de Gruyter, 1998.