ICEAA 2022 “Simplifying Software Sizing with Simple Function Points”

Authors: Carol Dekkers, Dan French

Abstract:

Professional software cost estimators recognize that one the most elusive, yet fundamental components of parametric software cost estimation is the size of the software under development. While many methods have been proposed over the years to quantify software size, none has been as stable or independent of changing technologies as functional size measurement (FSM), first introduced at IBM in the late 1970’s. FSM and its unit of measure, function points, derives software size based on a standardized assessment of its functional requirements. Today, the most popular and globally accepted FSM approach is the International Function Point Users Group (IFPUG) Function Point Analysis (FPA) version 4.3.1. In October 2021, the IFPUG released a new and standardized approach called Simple Function Points (SFP) version 2.1, based on an approach developed by Dr. Roberto Meli of Italy in 2010.

This paper introduces the SFP methodology, demonstrates its use, and highlights the challenges and opportunities for software cost estimators who need to estimate software size from high level software requirements. We will also explore the key differences between SFP and traditional IFPUG FP, including guidance for cost estimators using Function Point measures as the basis for their software cost *estimates.*

Introduction and brief history of IFPUG function points:

The IFPUG function point analysis methodology was developed by IBM in the 1970’s in response to customer concerns that newer, more efficient software languages (such C, SQL and Pascal) resulted in a smaller volume of computer code (quantified at the time by the number Source Lines of Code or SLOC) and thus, appeared to be of less “value” to their customers. With the advent of higher-level languages, they found that they were increasingly experiencing cost and schedule overruns for software projects based on SLOC and sought to find a better means of assessing software size that would be independent of the development technology.

To address this issue, IBM assembled a team of software engineers, led by Allan Albrecht, with the goal of developing an alternative software size measure, agnostic of programming language and platform. The first iteration of “Function Points” was formally presented in Albrecht’s paper “Measuring Application Development Productivity”, at an IBM Guide/Share conference. The industry response was so positive that the rest, as they say, is history.

In 1984, the International Function Point Users Group (IFPUG) was founded as the not-for-profit custodians of the Function Point sizing methodology and the first IFPUG Function Point Counting Practices Manual (CPM) version 1.0 followed in 1986.

The IFPUG Function Point methodology has slowly evolved and standardized over the years, but the original Albrecht-based components and rules still apply.

Following the 1998 publication of the International Organization for Standardization /International Electrotechnical Commission (ISO/IEC) functional size measurement framework suite of standard ISO/IEC 14143-1: Concepts of Functional Size Measurement (FSM), et al, IFPUG’s Function Point Analysis method became the first ISO/IEC standardized Functional Size Measurement Method: ISO/IEC 20926, of which the current instantiation is known as ISO/IEC 20926: IFPUG Functional Size Measurement Method version 4.3.1.

Over the years, the function point methodology has matured and is now codified into the ISO/IEC standard (30 pages) supported by a formal counting practices manual (CPM) with several hundred pages of terms, application guidelines, and examples of practical implementation FP counts.

Today, it is well recognized by ICEAA and other leading software cost estimating experts within the US government and internationally that software size is one of the major drivers of software development cost and schedule. Additionally, as more and more organizations cope with tighter Information Technology (IT) budgets, coupled with increases in project overruns and failures, there is a major need to develop better, fact-based, and reliable software estimates. While IFPUG FPA holds promise to revolutionize the software cost estimating industry, the time and resources required to properly train and implement function point-based estimating and measurement using the formal IFPUG methodology presents barriers. Additionally, the formal IFPUG method relies on detailed functional requirements that may not yet be fully specified when the estimate is needed.

The emergence of Simple Function Points (SFP)

The reasons stated above, in addition to the need to create a functional size estimate early in the development lifecycle, provided the impetus for a group of Italian researchers, led by Dr. Roberto Meli, to develop a simplified approach to functional size measurement. In 2009 they debuted a method they called Early and Quick Function Points (E&Q FP), based on the IFPUG method. E&Q FP replaced the function identification and complexity steps involved in the formal IFPUG method with a more generic and simplified FP scoring system and reduced the dependence on specific requirements details such as the number of data fields or files involved in countable components, thereby enabling a quicker estimation of functional size from higher level requirements documents. The method also allowed for applying the traditional IFPUG formal counting rules when such details were available.

E&Q FP eventually evolved into the Simple Function Point method (SiFP) in 2010 and was subsequently acquired by the IFPUG in 2019. In October 2021, the IFPUG standardized terminology and formally released the method as IFPUG Simple Function Point (SFP) version 2.1.

SFP simplifies the IFPUG method by reducing the functional size measurement process to the assessment of two IFPUG-compatible base functional components: Elementary Processes (EP) and Data Groups (DG), each with a single function point value: 4.6FP for EPs and 7 for DGs. As such, the steps of determining the primary intent to categorize five distinct types of functions, and the subsequent step of then categorizing them based on their relative complexity (low, average, or high) before assigning function point values.

Having recently released and formalized the SFP as an IFPUG product, work is underway to develop formal IFPUG SFP training programs and a formal SFP-based certification.

ICEAA 2022 “Simplifying Software Sizing with Simple Function Points”

Authors: Carol Dekkers, Dan French

Functional Size Concepts and Terminology

The same concepts and definitions pertaining to functional size measurement and functional size are identical to both IFPUG Function Points (FP) and IFPUG Simple Function Points (SFP). This section provides an overview of salient terms for readers unfamiliar with this software sizing approach.

There are a few key terms and definitions applicable when discussing IFPUG FP and IFPUG SFP. Note all terms and definitions are in accordance with the IFPUG Counting Practices Manual (CPM) version 4.3.1 and the Simple Function Point (SFP) Manual version 2.1. Those taken directly from the official IFPUG documents are included in italics below.

According to the IFPUG CPM, **functional size** is the *“measure of the functionality that a <software> application provides to the user, determined by the application function point count.”* (IFPUG, 2010)
Functionality or functions, in turn, are the user specified functions or business practices and procedures that the software performs, as specified by the **Functional User Requirements (FUR).**

Functional User Requirements (FUR) *- A sub-set of the user requirements; requirements that describe what the software shall do, in terms of tasks and services.* **FUR** are those requirements that describe what the software will do: for example, what data to store, what reports to produce, which data to display, what information to send to other systems, to name a few.

Functional size measurement (FSM) is a methodological approach to determining the Functional Size from evaluating a software’s FUR and assigning a specified number of function points to each.

Note that FUR are distinct from, and should not be mistaken for, other types of software requirements: technical, quality, or **non-functional requirements (NFR)**, that describe other aspects of the software including how the software must perform (NFR), the quality of the software (also NFR), the development environment (technical) or the programming language. A few further examples of software requirements that are NOT functional requirements include: the hardware or hosting platform(s), quality requirements, response time (to meet service level agreements), data capacity, industry or organizational standards and policies, and processing loads. Many of these requirements can be measured using a different methodology and units of measure, such as the IFPUG Software Non-Functional Assessment Process (SNAP) and associated SNAP points.

(Application or software) ***Boundary*** *- The boundary is a conceptual interface between the software under study and its users.*

User *- A user is any person or thing that communicates or interacts with the software at any time.* A user could be a physical person, other software or hardware, or anything that sends or receives data that crosses the software’s application boundary.

Elementary process (EP) - *“An Elementary Process is the smallest unit of activity, which is meaningful to the user, that constitutes a complete transaction, it is self-contained and leaves the business of the application being measured in a consistent state”. The term* ***transaction*** *here does not mean a physical collection of software instructions grouped according to a technical criterium (a Non-Functional Requirement). An elementary process* *is, instead, a logical aggregation of processing steps which is meaningful from a logical user perspective, and it is fulfilling a Functional Requirement.*

Logical file (LF) "*A Logical File represents functionality provided to the user to meet internal and external data storage requirements. It is a user recognizable group of logically related data or control information maintained and/or referred within the boundary of the application being measured." The term file here does not mean physical file or table. In this case, file refers to a logically related group of data and not the physical implementation of those groups of data.*

Additionally for the formal IFPUG FP methodology, the following definitions apply:

An ***internal logical file (ILF)*** *is a user recognizable group of logically related data or control information maintained within the boundary of the application being measured. The primary intent of an ILF is to hold data maintained through one or more elementary processes of the application being measured.*

An ***external interface file (EIF)*** *is a user recognizable group of logically related data or control information, which is referenced by the application being measured, but which is maintained within the boundary of another application. The primary intent of an EIF is to hold data referenced through one or more elementary processes within the boundary of the application measured. This means an EIF counted for an application must be in an ILF in another application.*

An ***external input (EI)*** *is an elementary process that processes Data or control information sent from outside the boundary. The primary intent of an EI is to maintain one or more ILFs and/or to alter the behavior of the system.*

An ***external output (EO)*** *is an elementary process that sends data or control information outside the application's boundary and includes additional processing beyond that of an external inquiry. The primary intent of an external output is to present information to a user through processing logic other than or in addition to the retrieval of data or control information. The processing logic must contain at least one mathematical formula or calculation, create derived data, maintain one or more ILFs, and/or alter the behavior of the system.*

An ***external inquiry (EQ)*** *is an elementary process that sends data or control information outside the boundary. The primary intent of an external inquiry is to present information to a user through the retrieval of data or control information. The processing logic contains no mathematical formula or calculation and creates no derived data. No ILF is maintained during the processing, nor is the behavior of the system altered.*

Data Element Type (DET) *- A unique, user recognizable, non-repeated attribute.*

File Type Referenced (FTR) *- A data function read and/or maintained by a transactional function.*

Record Element Type (RET) *- A user recognizable sub-group of data element types within a data function*

ICEAA 2022 “Simplifying Software Sizing with Simple Function Points”

Authors: Carol Dekkers, Dan French

Evolution of Simple Function Points (SFP)

With the initial introduction of function points in the mid-and late1980’s, many software development organizations, who had been struggling with delivering high fidelity software estimates and metrics using SLOC, were quick to adopt the new approach to software sizing based on the functionality provided to its users. There were adjustments made to the methodology in the 1990’s and early 2000’s resulting in new versions of the counting practices manual to address issues and concerns users had in the application of the rules. Additionally, the implementation of non-mainframe software platforms provided challenges to applying the rules, in particular interpretation of the application boundary. However, with the release of version 4.1 of the CPM in 1996 the rule set was stabilized.

While IFPUG worked to address shortfalls in the process, there were still challenges with developing function point counts when the requirements were not detailed and the ability to identify key components such as DETS or FTRs was not possible. There were also claims (false) that the use of function points was not possible until detailed design requirements were available, function points could not be counted until the software was in production, or that certain platforms, application types, or some software development methodologies could not be counted.

The assertions, mostly incorrect, did demand that there be a way to address the concerns around lack of details needed to properly identify and classify functions, particularly in the early phase of software development. IFPUG and others promulgated differing approaches which primarily consisted of assuming the average complexity for all functions.

In 2007, DPO in Italy introduced the more refined concept of Early and Quick Function Points (E&Q FP). Based on the IFPUG methodology, they replaced the “assume average complicity” concept with a more refined approach. While IFPUG-based, the need for detailed requirements was not necessary, the analyst is still required to have a reasonable amount of detail in the requirements to properly discern which of the various levels of aggregation is appropriate as well as which functional type it is, which correlates highly with the IFPUG functions.

The range of the function sizes using are determined based on the aggregation level employed. For 1st aggregation level it uses the traditional function point sizes and ranges from 3FP to 15FP. 2nd aggregation level ranges from 4.0 to 8.1 FP, 3rd aggregation level goes from 14.1 to 101.8 and the 4th aggregation level spans 111.5 to 617.4 FP. See appendix A for E&QFP aggregation levels and sizing table.

While the E&QFP approach provided a mechanism for counting function points based on the level of detail of the functional user requirements, including a way to count FP where the FURs were at a high level, some function point analysts still had difficulty with determining how to count to the appropriate level of size, aggregation and determining the appropriate sizes for Typical Processes (TP), General Processes (GP), General Data Groups (GDG) and Macro Processes (MP). Given the ranges of these functions, misclassification could still lead analysts to over or undercounting the software size.

This led Dr. Meli to further refine and simplify the methodology and develop Simple Function Points (SFP). The method approximates the IFPUG function point methodology but does not require the identification of DETS, RETS, or FTRs and consists of only two types of functions: Elementary Process (EP) replacing EI, EO, and EQ and Logical File (LF) replacing ILF and EIF.

IFPUG FP and IFPUG SFP – Similarities and Differences

The SFP concept embraces the same concepts and definitions that the traditional IFPUG method does with regards to the definition of boundary, functional and technical requirements, maintenance, enhancement, user, logical file and elementary process but removes the need for the analyst to identify and classify the different transactional and data function types into EI, EO, EQ, ILF and EIF.

Rather, in SFP, functional user requirements are identified and classified as transactional Elementary Process (EP) functions or logical (data) file (LF) functions. SFP also eliminates the complexity rating of each function (as Low, Average or High) based on their component range of DETS, FTRS or RETS. This allows for the functional size to be quantified more easily based on high-level, not-yet-detailed functional requirements, and also speeds up the assessment process by eliminating the need to assess the functionality based on the various transaction and date types, and their component DETS, FTRS and RETS.

Table 1 summarizes the similarities and differences between IFPUG FP and IFPUG SFP.

Table 1: IFPUG FP compared to IFPUG SFP

Concept	IFPUG FP	IFPUG SFP
IFPUG standardized glossary	Yes	Yes, same
Intent to measure functional size based on FUR	Yes	Yes, same
Method owned by IFPUG	Yes	Yes
IFPUG FP measurement steps: 1. Gather available documentation 2. Purpose/scope/boundary, identify FUR 3a. Measure data functions 3b. Measure transactional functions 4. Calculate functional size 5. Document and report	Yes, but steps 3a and 3b involve additional sub-steps: subclassification into 3 types of transactional functions and 2 types of data functions, and a complexity classification (into Low, Average, or High) to get FP values	Yes
Base functional components (BFC): transactional functions and data functions	Yes: Transactional functions are subdivided into EI, EO, EQ, and Data functions are subdivided into ILF, EIF	Yes: Transactional functions are called “Elementary Processes” and Data Functions are called “Logical Files”

Number of different FP values allocated across function types	3 FP values allocated as Low, Average or High across 5 function types (total of 8 different values)	2 SFP values allocated, one each to two function types
Range of FP values by category	Transactional functions are worth between 3 and 7 FP depending on type and complexity. Logical files are worth 7 to 15 FP depending on type and relative complexity	All transactional functions are considered to be EP and assigned 4.6 SFP. All data functions are considered to be logical files and assigned 7 SFP
Unit of measure	Function Points (FP)	Simple Function Points (SFP)
Convertibility	1 FP = 1 SFP	1 SFP = 1 FP

When to use IFPUG SFP vs IFPUG FP

The determination of which method to use can be influenced by a number of factors including skill level, expertise and training of the analyst, fidelity and availability of detailed functional requirements, and the business need for the count. It is always advisable that when a count is being performed that the analyst(s) conducting the count are properly trained and preferably IFPUG certified regardless of method used. Having a count performed by untrained or poorly trained analysts will likely result in a function point count that is significantly over or under counted. Ideally, the analyst is a Certified Function Point Specialist (CFPS) or Certified Function Point Practitioner (CFPP). While IFPUG currently does not have training or certification available for the SFP methodology, a task force has been formed to deliver these by the end of 2022.

If the analyst(s) is/are not trained then it is advisable, regardless of the phase of the project or requirements state, to us the SFP method. Likewise, if the requirements and supporting documentation (Entity Relationship Diagrams (ERD), Data Schema, Interface Requirements Documents (IRD) are not defined to the point where DETS, FTRS or RETS can be confidently identified. Typically, this is the case early in the software development lifecycle such as at the proposal or project definition phase. If there are cost or time constraints, or there is only a need for a Rough Order of Magnitude (ROM) estimate, then the SFP method can be used. If the sizing will be updated as the project progresses throughout the life cycle it is recommended that when there are sufficient details available to use the traditional IFPUG method, that it be done. Particularly if doing a baseline or application count.

Where there are trained analysts and sufficiently detailed requirements and other documentation available and there is sufficient time and resources, then it is recommended to use the traditional IFPUG Function Point methodology. It is advisable to use as well when a high degree of accuracy and fidelity for the sizing and estimate are required.

Can IFPUG FP or IFPUG SFP be used to Size Agile Software Development?

With regards to Agile, DevOps and other non-waterfall development methodologies and frameworks, there is a misconception that function points cannot be used, either SFP or traditional IFPUG. This is incorrect. In addition to function points being language, platform and technology agnostic, they are also agnostic to development methodology. It is likely that the requirements, typically documented as use cases in the Agile world, are not of sufficient detail and it would be likely that SFP would be the more appropriate sizing to use. FPs can size product backlogs, use cases, epic and features and provide the advantages of using rule-based sizing method over the subjective sizing typically employed in Agile such as story points. They are particularly useful for providing more accurate metrics such as sprint velocity, productivity and cost/fp.

Dos and Don’ts of Function Point analysis

There may be various circumstances which determine the function point sizing method used by the analyst(s) but regardless of whether simple function points are used or traditional IFPUG function points the following provide guidance on the dos and don’ts of function point analysis:

Do:

- Use properly trained analysts, if at all possible, even if it requires hiring an outside analyst
- Properly document the function point count and all source documentation
- Use traditional IFPUG function points if a high degree of accuracy in sizing is required for estimating or legal reasons and there is sufficiently detailed requirements to support it
- Use SFP when it is necessary to develop a quick sizing estimate with little documentation available

Don’t:

- Use SFP just because it is easier or quicker, make sure that it will also meet other business needs for the count
- Use SFP if using a parametric estimating tool to develop cost and schedule estimates as none currently on the market support native SFP sizing
- Don’t use traditional IFPUG function point sizing when there is limited time or lack of resources to properly conduct the count
- If sizing a waterfall method project and the early phase sizing estimates are done using SFP it is recommended to transition to traditional IFPUG function points sizing when there is the available documentation to support, it.
- Depending on the business need, it is not recommended to use SFP for application counts, because all of the prerequisite details to do a formal IFPUG FP count should be available and known.

ICEAA 2022 “Simplifying Software Sizing with Simple Function Points”

Authors: Carol Dekkers, Dan French

Example Case Study to Demonstrate Functional Size Estimation

Consider that we have a high-level CONOPS (Concept of Operations) document that outlines the following hypothetical functional requirements for a simple online book sales system:

a. Create, read, update, delete (CRUD), and store customer records.
b. System administrator functions to create, read, update, delete (CRUD), and store book catalog entries for available books.
c. Customers can display and browse book catalog by author or title.
d. Customers can select and see details about an individual book.
e. Customers can create an order for one or more books by selecting them from the catalog and placing them in a shopping cart that will be saved as an order.
f. The system will display the order summary with the total amount calculated from the prices of all books.
g. Customers can complete their order by paying with a credit card.
h. Software will generate an order summary to the customer.
i. Software will generate an order request to the sales staff at the store.

Table 2 presents the high-level summary of using both IFPUG FP (assuming all functions are average complexity) and IFPUG SFP. The total over the entire case study came out to be close for the IFPUG avg FP estimate and the IFPUG SFP estimate being 90 FP and 93 SFP. If there were more detailed requirements such as complex reports that would be scored as a high complexity EO (External output), there would be a larger variation between the methods because the value of a H EO is 7 FP versus the IFPUG SFP single EP score of 4.6 SFP.

Note that the following acronyms are used in Table 2:

For IFPUG avg (average) functions:

- *A EI or A EQ= average External Input or average External Query worth 4 FP*
- *A EO = average External Output worth 5 FP*
- *A ILF = average complexity Internal Logical File worth 10 FP*

For IFPUG Simple Function Point (SFP) functions:

- *EP = elementary process worth 4.6 SFP*
- *LF = logical file worth 7 SFP*

Table 2: Comparison of IFPUG FP (avg) and IFPUG SFP for CONOPS Case Study

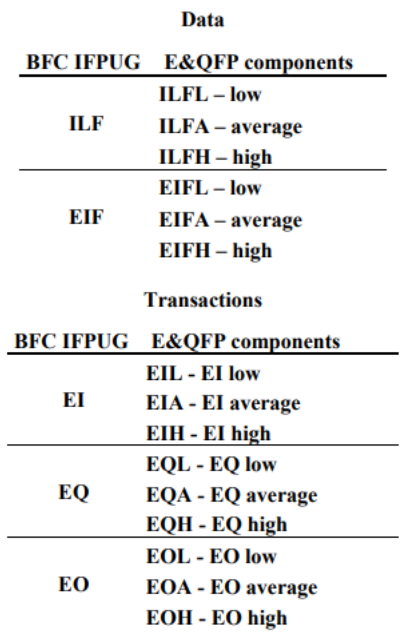
Functional Requirement	IFPUG avg functions	IFPUG FP value	IFPUG SFP functions	IFPUG SFP value
a. CRUD, store customer records.	3A EI, A EQ 1A ILF	26 FP	4 EP, 1 LF	25.4 SFP

b. CRUD, store book catalog	3A EI, A EQ 1A ILF	26 FP	4 EP, 1 LF	25.4 SFP
c. Display books by author or title	1A EQ	4 FP	1 EP	4.6 SFP
d. Select and display book details	1A EQ	4 FP	1 EP	4.6 SFP
e. Select books to create order	1A EI, 1A ILF	14 FP	1 EP, 1 LF	11.6 SFP
f. Display order summary (calcs)	1A EO	5 FP	1 EP	4.6 SFP
g. Pay for order with credit card	1A EI	4 FP	1 EP	4.6 SFP
h. Order summary to customer	1A EO	5 FP	1 EP	4.6 SFP
i. Order request to sales staff	1A EO	5 FP	1 EP	4.6 SFP
TOTAL	**8A EI, 3A EO, 4A EQ, 3A ILF**	**93 FP**	**15 EP, 3 LF**	**90 SFP**

Conclusion

The IFPUG function point methodology is a tried-and-true software sizing method that is an ISO/IEC Functional Size Measurement standard and is especially suitable to size software when detailed functional requirements are known. The evolution of the Simple Function Point methodology (IFPUG SFP V2.1) presents a simplified approach to functional sizing that is especially useful for early estimation when functional requirement details are not yet specified or available. IFPUG SFP facilitates using IFPUG FP concepts when conditions and circumstances warrant the use of a rules-based sizing method but requires one that can be readily used quickly for high-level requirements. IFPUG SFP provides such a method true to IFPUG FP, with the added benefits that it is easier to learn and provides a reasonable level accuracy in a more timely and efficient manner than using the formal IFPUG FP methodology.

Appendix A Early and Quick Function Point Aggregation Levels and Values (DPO):



Data

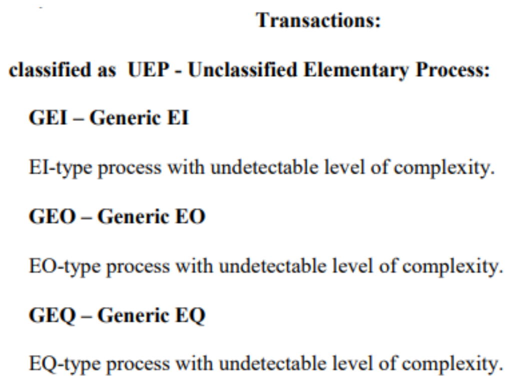
BFC IFPUG	E&QFP components
ILF	ILFL – low
	ILFA – average
	ILFH – high
EIF	EIFL – low
	EIFA – average
	EIFH – high

Transactions

BFC IFPUG	E&QFP components
EI	EIL - EI low
	EIA - EI average
	EIH - EI high
EQ	EQL - EQ low
	EQA - EQ average
	EQH - EQ high
EO	EOL - EO low
	EOA - EO average
	EOH - EO high

Table 2: Early and Quick 1st level aggregation (DPO)[2]

If less mature requirements are available, then the analyst can genericize the functions to the 2nd aggregation level:



Transactions:

classified as UEP - Unclassified Elementary Process:

GEI – Generic EI

EI-type process with undetectable level of complexity.

GEO – Generic EO

EO-type process with undetectable level of complexity.

GEQ – Generic EQ

EQ-type process with undetectable level of complexity.

UGO - Unspecified Generic Output (EO/EQ)
“doubtful” or “uncertain” output process for which there are no details available to differentiate between EO and EQ.

UGEP - Unspecified Generic Elementary Process (EI/EO/EQ)
“doubtful” or “uncertain” elementary process for which there are no details available to single out the primary goal, namely the presence of EI, un EO or un EQ.

GILF – Generic ILF

Sets of data recognizable by users as ILF-type of an uncertain complexity

GEIF - Generic EIF

Sets of data recognizable by users as EIF-type of an uncertain complexity.

UGDG - Unspecified Generic Data Group

Unspecified logical file (either ILF or EIF) of uncertain complexity.

Table 3: Early and Quick 2nd level aggregation (DPO)[3]

Transactions

When it is not possible to accurately identify a specific UBFC or the precise amount of UBFC that makes up a specific software component it is possible to use a 3rd level component.

Typical Process (TP)
It consists of a set of four typical functional processes: Insert, Edit, Delete, Display a record data, recognised as CRUD – (Create, Read, Update & Delete) and generally centred around a specific data store. Normally it corresponds to the general definition “Management of a data store”, “Management of …”.

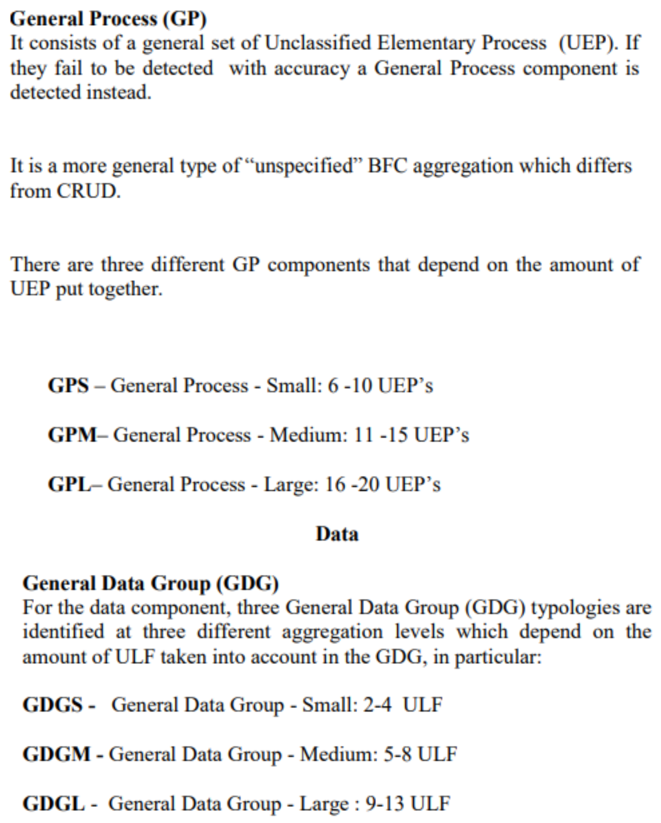
When detectable, the Typical Process helps save measurement time without losing out in accuracy in the four base components shortlisted.

There are three TP classes:

TPS – Typical Process - Small: CRUD

TPM – Typical Process - Medium: CRUD + List (EQ)

TPL – Typical Process - Large: CRUD + List (EQ) + Report (EO)



General Process (GP)
It consists of a general set of Unclassified Elementary Process (UEP). If they fail to be detected with accuracy a General Process component is detected instead.

It is a more general type of “unspecified” BFC aggregation which differs from CRUD.

There are three different GP components that depend on the amount of UEP put together.

GPS – General Process - Small: 6 -10 UEP’s

GPM– General Process - Medium: 11 -15 UEP’s

GPL– General Process - Large: 16 -20 UEP’s

Data

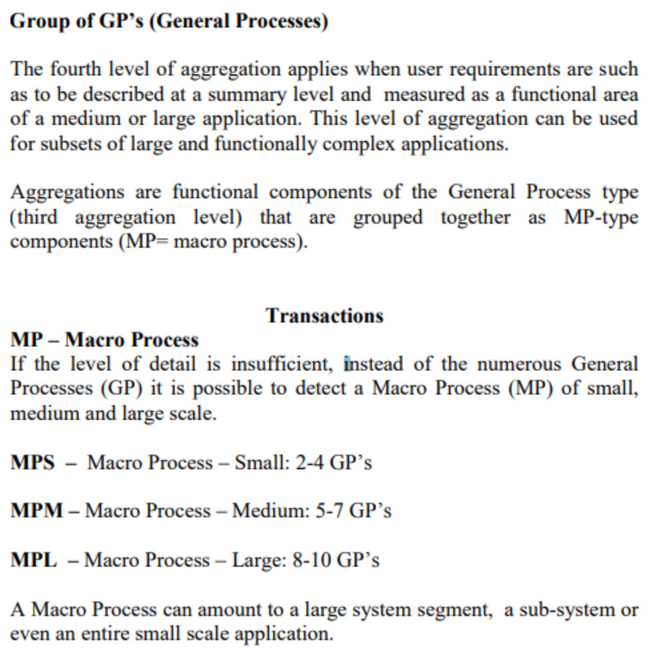
General Data Group (GDG)
For the data component, three General Data Group (GDG) typologies are identified at three different aggregation levels which depend on the amount of ULF taken into account in the GDG, in particular:

GDGS - General Data Group - Small: 2-4 ULF

GDGM - General Data Group - Medium: 5-8 ULF

GDGL - General Data Group - Large : 9-13 ULF

Table 4: Early and Quick FP 3rd level aggregation (DPO)[4]



Group of GP’s (General Processes)

The fourth level of aggregation applies when user requirements are such as to be described at a summary level and measured as a functional area of a medium or large application. This level of aggregation can be used for subsets of large and functionally complex applications.

Aggregations are functional components of the General Process type (third aggregation level) that are grouped together as MP-type components (MP= macro process).

Transactions

MP – Macro Process
If the level of detail is insufficient, instead of the numerous General Processes (GP) it is possible to detect a Macro Process (MP) of small, medium and large scale.

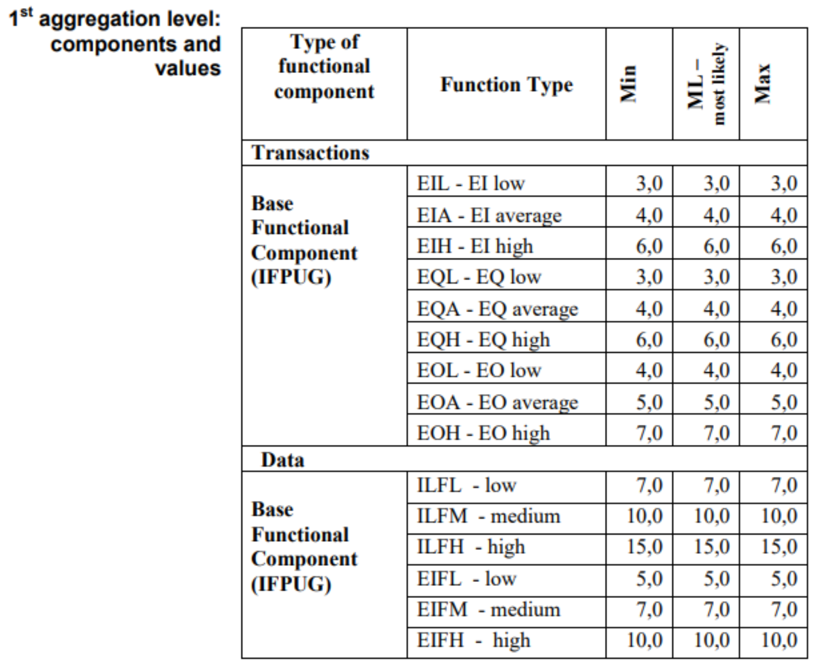
MPS – Macro Process – Small: 2-4 GP’s

MPM – Macro Process – Medium: 5-7 GP’s

MPL – Macro Process – Large: 8-10 GP’s

A Macro Process can amount to a large system segment, a sub-system or even an entire small scale application.

Table 5: Early and Quick FP 4th level aggregation (DPO)[5]

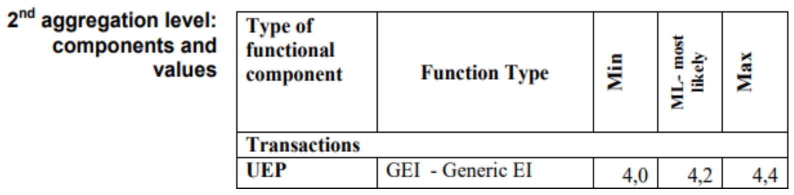


1st aggregation level: components and values

Type of functional component	Function Type	Min	ML – most likely	Max
Transactions				
Base Functional Component (IFPUG)	EIL - EI low	3,0	3,0	3,0
	EIA - EI average	4,0	4,0	4,0
	EIH - EI high	6,0	6,0	6,0
	EQL - EQ low	3,0	3,0	3,0
	EQA - EQ average	4,0	4,0	4,0
	EQH - EQ high	6,0	6,0	6,0
	EOL - EO low	4,0	4,0	4,0
	EOA - EO average	5,0	5,0	5,0
	EOH - EO high	7,0	7,0	7,0
Data				
Base Functional Component (IFPUG)	ILFL - low	7,0	7,0	7,0
	ILFM - medium	10,0	10,0	10,0
	ILFH - high	15,0	15,0	15,0
	EIFL - low	5,0	5,0	5,0
	EIFM - medium	7,0	7,0	7,0
	EIFH - high	10,0	10,0	10,0

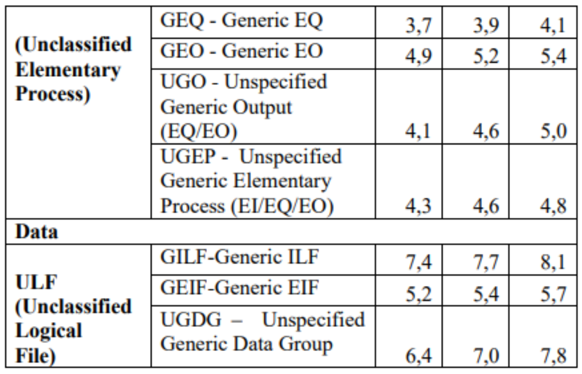
ICEAA 2022 “Simplifying Software Sizing with Simple Function Points”

Authors: Carol Dekkers, Dan French

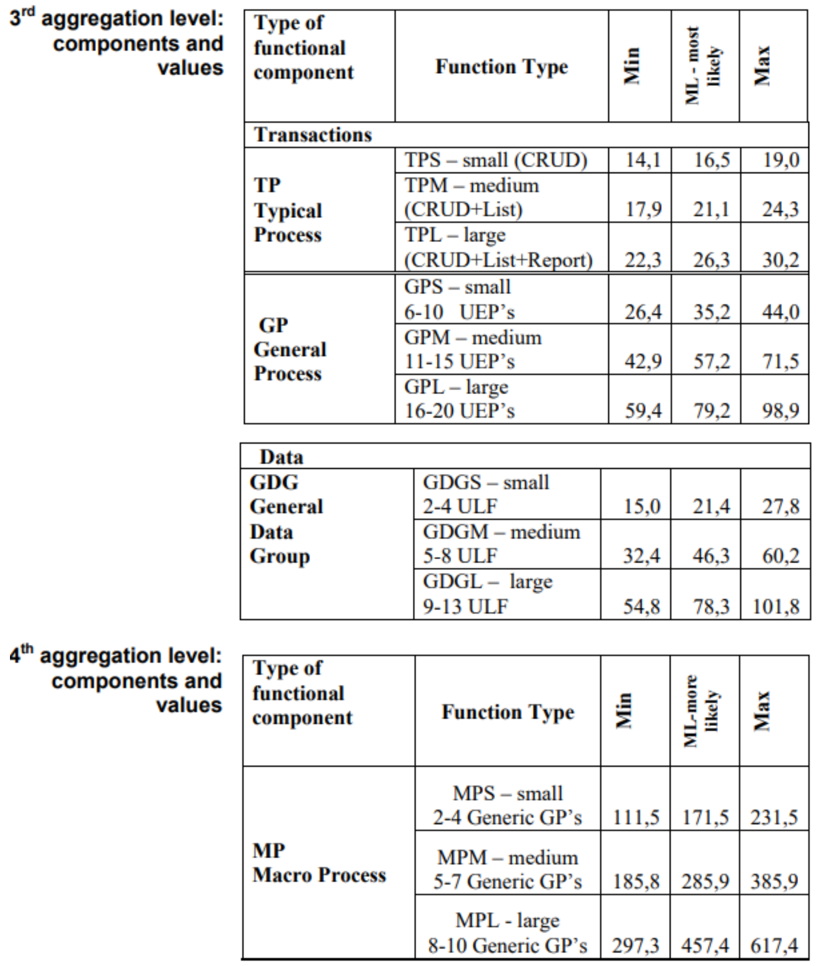


2nd aggregation level: components and values

Type of functional component	Function Type	Min	ML- most likely	Max
Transactions				
UEP	GEI - Generic EI	4,0	4,2	4,4



(Unclassified Elementary Process)	GEQ - Generic EQ	3,7	3,9	4,1
	GEO - Generic EO	4,9	5,2	5,4
	UGO - Unspecified Generic Output (EQ/EO)	4,1	4,6	5,0
	UGEP - Unspecified Generic Elementary Process (EI/EQ/EO)	4,3	4,6	4,8
Data				
ULF (Unclassified Logical File)	GILF-Generic ILF	7,4	7,7	8,1
	GEIF-Generic EIF	5,2	5,4	5,7
	UGDG – Unspecified Generic Data Group	6,4	7,0	7,8



3rd aggregation level: components and values

Type of functional component	Function Type	Min	ML - most likely	Max
Transactions				
TP Typical Process	TPS – small (CRUD)	14,1	16,5	19,0
	TPM – medium (CRUD+List)	17,9	21,1	24,3
	TPL – large (CRUD+List+Report)	22,3	26,3	30,2
GP General Process	GPS – small 6-10 UEP’s	26,4	35,2	44,0
	GPM – medium 11-15 UEP’s	42,9	57,2	71,5
	GPL – large 16-20 UEP’s	59,4	79,2	98,9
Data				
GDG General Data Group	GDGS – small 2-4 ULF	15,0	21,4	27,8
	GDGM – medium 5-8 ULF	32,4	46,3	60,2
	GDGL – large 9-13 ULF	54,8	78,3	101,8

4th aggregation level: components and values

Type of functional component	Function Type	Min	ML-more likely	Max
MP Macro Process	MPS – small 2-4 Generic GP’s	111,5	171,5	231,5
	MPM – medium 5-7 Generic GP’s	185,8	285,9	385,9
	MPL - large 8-10 Generic GP’s	297,3	457,4	617,4

Table 6: Early and Quick FP Range Values by Aggregation Level (DPO)[6]