Part One

FSSM: Introduction

Chapter 1

Introduction to Functional Software Size Measurement

1.1 INTRODUCTION

For software projects, it is very important to determine the size of the software for development effort estimation, project planning, and cost/productivity analysis.

How much effort will be spent on the software development – analysis, design, coding, testing, and documentation – for a particular application depends on the size of the software to be developed.

The size of the software to be developed depends on the functionality required by the application, and the effort is spent to develop all the required functionality.

The functionality required by the application is described in the Functional User Requirements (FUR)[1] which are the specifications of the required functionality and hence are also known as the Functional Requirements Specifications (FRS). These terms – FUR (Functional User Requirements)[1] and FRS (Functional Requirements Specifications) – denote the same thing, that is, the specifications of the user functionality required by the application.

1.2 FUNCTIONAL SIZE MEASUREMENT AND EFFORT ESTIMATION

Functional Size Measurement (FSM)[1] is an effective method for measuring the size of the software because it is meant to measure the functionality which is described in the FRS and on which the effort is spent for analysis, design, coding, testing, documentation, and maintenance.

All the functionality of the application is distributed in different constituent components of the software. These components are defined as Base Functional Components (BFCs)[1] in the ISO/IEC 14143-1[1] standards. Measurement of the size of software is achieved by measuring the size of the BFCs[1] and computing the total size based on the sizes of the BFCs[1].

The constituent components, that is, the BFCs[1], of the software comprise sub-types of the BFCs[1], defined as the Base Functional Component Types (BFC Types)[1] in the

Functional Software Size Measurement Methodology with Effort Estimation and Performance Indication, First Edition. Jasveer Singh.

Companion website: http://booksupport.wiley.com

ISO/IEC 14143-1[1] standards. Measurement of the size of the BFCs[1] is achieved by measuring the size of the BFC Types[1] and computing the size of the BFCs[1] from the sizes of the BFC Types[1].

After deciding the boundaries which define the limits of the software measurement to a particular area of functionality specified in the FUR[1], the functional size of the BFC Types[1] is calculated by assigning the Units of Functional Size[2] to the BFC Types[1]. The functional size unit is defined in the methodologies mostly as Function Point[4]. Function Points[4] are thus assigned to various BFC Types[1] and are summed up for the entire functionality within the defined boundaries to produce the total software size of the corresponding BFCs[1], and then the sum of the Function Points[4] of the BFCs[1] produces the total software size.

Because of some substantial limitations (details of which are described in this and later chapters), in some of the popular and widely used software size measurement methodologies available, various multiplication factors which are different according to the type/field and functional contents of applications are used subsequently in order to estimate the effort required for development. The use of multiplication factors is required because a very limited part of all the measurable BFC Types[1] of a part of all the measurable BFCs[1] is measured, and from that small partial measurement, an attempt is made to estimate the effort of development for the whole software based on the assumption that the multiplication factors will take care of the unmeasured part and will thus produce the correct effort estimation. This approach may give erroneous results as clarified in further description. The FSSM is specially designed to overcome such limitations.

1.3 IMPORTANT CONSIDERATIONS FOR THE SOFTWARE SIZE MEASUREMENT AND EFFORT ESTIMATION

Care should be taken while selecting a methodology to use for the Functional Size Measurement[1]. The methodology should be such that it measures all the major, relevant functionality on which the effort is required to be spent on development; otherwise – in case of measurement of only a small part of the software functionality – the significance of measurement is lost. The methodology should be simple and easy to use as well. Hence, there is a need of an integral software size measurement methodology which is presented in this book.

The following are some important facts that should be considered regarding the Functional Size Measurement[1], and methodologies to measure the functional size:

1. Software is composed of several distinct components which are inter-related but do not have any obligatory fixed size proportion relationship with respect to one another in any software application. That means, if there are four components: c1, c2, c3 and c4, their size may have any proportion according to the functional requirements concerning these components in different applications. So the magnitude of c1 may be much greater (measured by any method, e.g., counts, units) than c2 in one software application but it may be the opposite in another application. Similarly, the size proportion of c1 to other components in one application may be entirely different than in another application.
2. Each component has several features. For example, component c1 may have two features: c1f1, c1f2; component c2 may have three features: c2f1, c2f2, c2f3; component c3 may have four features: c3f1, c3f2, c3f3, c3f4; and component c4 may

have one feature: c4f1. The size of different features varies according to the application. For most of the features, there is no fixed size relationship of different features of a component amongst themselves and with the features of other components. For example, in one application, the size of one feature c1f1 may be about 10 times bigger (measured by any method, e.g., counts, units) than the size of c2f3, but in another application, it may be of approximately the same size or even of opposite proportion.

3. If an application is made of several sub-programs or modules, the sub-programs have different proportions of different features of the components and may not have any similar size proportionate relationship with respect to one another or with respect to the whole software.

4. Size and complexity of a software component are interrelated.

What is meant by the complexity of the software components here is the level and degree of effort and time required for understanding, analyzing, designing, coding, testing, and maintaining the components.

The more the effort and time are required for these activities, the more complex the component is. For example, regarding data component, 1 table with 10 columns in a database application, or 1 data record with 10 fields in a program is quite simple, but 20 tables with 15–20 columns each, or 20 data records with 15–20 fields each, respectively, are more complex, and 80 tables with 20 columns each, or 80 data records with 20 fields each, respectively, are highly complex as far as the effort and time required for their understanding, analyzing, designing, coding, testing, and maintaining are concerned. The same is true for other components – functionality execution, user interfaces, and message communication.

As a general rule, the bigger the size of the component, the more the effort and time required to understand, analyze, design, code, test, and maintain it; and the more the effort and time required to understand, analyze, design, test, and maintain it, the more complex the component is.

5. Effort for software life cycle activities – development (analysis, design, coding, and testing), maintenance, documentation, etc. – is spent on all the features of all the components. Of course, not all the minute details of all the functionality can be taken into account for measurement, but it is important that the most relevant, major components and all their important features should be considered for size determination, effort estimation, and performance indication. If an accurate estimation of the total software size is needed for correct project planning, the size of all the measurable features for all the components should be computed. Hence, no major and relevant component and at the same time, no major and relevant feature should be ignored for measurement and estimation.

6. If we consider various application programs in different domains and see the magnitude of the presence of different features of the constituent components, it will vary from one application to the other. Not only that, the magnitude will vary in different sub-programs or modules of these applications programs with respect to one another.

The relationship of size between the components and their features depends on only the functionality required and it varies from one application to the other, and from one sub-program (module) to the other of any particular application.

For example, in the signaling part of the application of telecom call handling (call handling would normally consist of signaling, call control, number analysis,

routing, charging, subscriber data, etc.), there will be much more amount of input/output operations from the input/output devices and much more message communication to/from other modules than the disk memory read/write operations. On the other hand, subscriber data will have more disk memory read operations to access the subscriber data, and message communication to/from call control module than computational/logical and decision operations. Routing may have more proportion of decision operations to decide about the routing.

In the radar application which will normally consist of input/output for image data, image extraction, filtering, etc., there will be high amount of input/output operations, high amount of computational operations, and medium amount of disk memory read/write operations in the complete application.

In a data warehouse report preparation application, there will be high amount of disk memory read/write operations and high amount of computational operations.

Table 1 shows the approximate magnitude (size) of some software constituents – components, features, and operations – and their size and complexity in terms of very low, low, medium, high, and very high depending on the number of functional operations or number of functional data elements being almost none, very few, a few, many, and too many, respectively, for the sub-programs of two applications: Telecom Call Handling and Radar. In this table, the sub-programs (or modules) of these applications are considered separately and individually, and their interactions with the other sub-programs (or modules) of the same application program are considered. The table contents are a rough estimate based on practical experience and assessment, and not based on any measurements.

Table 2 shows the approximate magnitude (size) of different components and features of components in different applications. In this table, the whole applications are considered and not separate sub-programs (modules) of the applications individually. For example, for the application Telecom Call Handling, all the sub-programs (modules) for signaling, call control, number analysis, routing, and charging are considered together as a whole application. Similarly, for radar application, all the sub-programs (modules) for input data, image extraction, filtering, and output display are considered together as a whole application. Only one instance of the program execution is considered and not the execution and operations for multi-users with multi-instances of the program running simultaneously. The table contents are a rough estimate based on practical experience and assessment, and not based on any measurements.

The purpose of Tables 1 and 2 is just to show how various applications differ in the size of the components and in the size of the features of the components.

7. If only a few features of one component or of a few components are measured (by any method, e.g., counts, units) and based on that the complete software size is calculated by using various multiplication factors, it may give unreliable results because there is no fixed proportional relationship of the size of one feature to the other feature(s) of the same component or of the other components for the majority of features as explained before. For example,

Example 1 In one application, there may be 50 data tables for which there may be one memory data read and write operation each from/to disk for each table, so the count for memory data read/write operations will be 100. In another application, there may be 2 data tables of which there may be 25 memory data read and write operations each from/to disk for each table, so the count for memory data

Table 1 Software Constituents' Approximate Relative Magnitude (Size) in Different Applications Sub-programs

Approximate Relative Magnitude (Size) of Some Software Constituents in the Sub-programs of Two Different Applications								
Sl. No.	**Software Application Sub-program**	**Memory Read/Write Operations**	**User Interface Input/ Output Operations**	**Message Communication with Internal Modules, External Application Programs**	**Functionality Execution: Computa-tional Operations**	**Functionality Execution: Logical Operations**	**Functionality Execution: Actions, Decisions, Repeat Operations, Execution Flow Control Operations**	**Functional Data Size and Complexity: Diversity, Variety and Width (Number of Classes, Their Attributes and Fields)**
1.	Telecom Call Handling: Signaling	Very low	High	High	Very low	Low	High	Medium
2.	Telecom Call Handling: Subscriber Data	High	Very low	Medium	Very low	Medium	High	High
3.	Telecom Call Handling: Routing	High	Very low	Medium	Medium	High	High	High
4.	Radar: Input Data	Low	High	High	Low	Low	Low	Medium
5.	Radar: Image Extraction	Low	Low	High	High	High	High	Medium
6.	Radar: Filtering	Low	Low	High	High	High	High	Medium
7.	Radar: Output Image	Medium	High	High	Low	Low	Low	Medium

Table 2 Software Constituents' Approximate Relative Magnitude (Size) in Different Applications

Approximate Relative Magnitude (Size) of Some Software Constituents in a Few Applications						
Sl. No.	**Software Application**	**Memory Read/Write Operations**	**User Interface Input/Output Operations**	**Message Communication with Internal Modules, External Application Programs**	**Functionality Execution: Actions, Decisions/ Repeat/Execution Flow Control Operations, Computational/Logical Operations**	**Functional Data Size and Complexity: Diversity, Variety and Width (Number of Classes, Their Attributes and Fields)**
1.	Telecom Call Handling	Low	Medium	Medium	High	High
2.	Telecom Billing	High	Low	Low	High	High
3.	Video Games	Low	High	Low	High	Medium
4.	ERP – Inventory, Sales, Marketing, HR Management	High	High	High	High	Very high
5.	Radar – Image Extraction, Filtering, Input, Output	Low	High	Medium	High	High
6.	Online Shopping	High	High	Medium	Low	High
7.	Data Warehousing	High	Medium	Medium	High	Very high
8.	Text Processing	High	High	Low	Low	Low
9.	Computer Numerical Control (CNC) Machine	High	High	Low	High	Medium

read/write operations will be 100. In both the applications, the number of memory data read/write operations is the same but the data complexity and effort required from the point of view of development is much more in the former application. If we do the measurement of only memory data read and write operation, we would not know how big and complex the data part is, and whether database expertise is required for the development.

Example 2 In an application, there may be 2 user screens, one for input data and the other for output data, being used for input/output data 50 times each, so there will be 100 input/output operations. In another application, there may be 50 user screens for input data being used for 1 input operation each and 50 user screens for output data being used for 1 output operation each, so the total number of input/output operations will be 100. In both the applications, the number of input/output operations is the same but the user interface design in the latter application is more effort-consuming than the former. Only the size of input/output operations does not give a right indication about the input/output interface size and complexity, and thus the effort required for designing the screens, that is, screen structures, cannot be estimated based on the size of the input/output operations.

Moreover, in the aforementioned examples, the size of memory data read/write and input/output operations, respectively, does not indicate the magnitude of other operations in the program, for example, computational, logical, decision, repeat, execution flow control, and message exchange operations, and thus the effort required for these operations cannot be estimated.

8. The existing methodologies of software size measurements measure only a few features of a few components and try to estimate the total software size with the help of some multipliers based on type/field of applications and past experience with similar projects. Effort estimation done in this manner may be inaccurate in most of the cases.

9. Different skills and expertise may be needed for different activities of the software life cycle; for example, for an application which uses huge amount of data deploying databases, database skills would be required. Hence, it is necessary/desirable to know the size and complexity of different components of the software which can help in project planning activities from the point of view of human expertise and skills required.

10. Real effort depends on many factors such as expertise, experience, knowledge, productivity, and efficiency of the project team members but the average effort which has, more or less, approximate fixed proportionate relationships with the type of components to be developed, tested, and maintained can be calculated taking into consideration the average productivity, efficiency, and knowledge.

11. Static and dynamic (run-time) characteristics of any software application may be different. For example, in a program, there may be one disk memory read operation which is in a loop of one thousand times. While counting the Function Points[4] for the memory read operation, it will be counted as one Function Point[4] which is correct from the point of view of software development because it will lead to the effort for writing the statement(s) for one such operation, but at run-time, it will create high disk memory read traffic. If it is desired to assess the run-time characteristics based

on the Function Points[4], careful selection of the right kind of parameters (Function Points[4]) will be required.

12. Real run-time performance characteristics can best be obtained by capturing data at real run-time and analyzing that to find out what type of operations are being performed because the analysis based on static structure of data may not indicate the true performance due to the presence of decision operations. For example, if in a program there is one decision statement which leads to many:
a. memory data write operations if it is evaluated true;
b. data output operations for data display if it is evaluated false;
and if, at the execution time, the possibility of its being true is about 75%, the result will be high memory write traffic in about 75% of the cases and high output traffic for data display in the other 25% of the cases whenever the program is run. Based on the static structure analysis only, we will assess the program to have high memory as well as high output traffic which is also correct from the point of view of design because care will have to be taken to provide suitable design for both high memory traffic as well as high output traffic since both the situations can occur.

1.4 INTRODUCTION TO THE FUNCTIONAL SOFTWARE SIZE MEASUREMENT METHODOLOGY WITH EFFORT ESTIMATION AND PERFORMANCE INDICATION (FSSM)

The methodology presented in this book – Functional Software Size Measurement Methodology with Effort Estimation and Performance Indication (FSSM) – is based on the principles of Functional Size Measurement (FSM)[1] and is fully compliant with the ISO/IEC 14143-1[1] standards. It also takes into account all the important considerations concerning size measurement and effort estimation mentioned earlier in Section 1.3.

In this methodology, the size of the software is measured based on the Functional Requirements Specifications which contain mainly the:

a. description of the functionality, error handling, messages, and user interfaces;

b. logical data model providing high-level information about the data.

Among the available software size measurement methodologies at present, some of them give an estimation of the software size by counting the Function Points[4] and are extensively used but they use very limited software measurability features (only a small percentage of the complete features) for measurement. For example, some currently available methodologies take into account only some of the following types of operations for the data movements:

a. memory read/write operations used for reading/writing the data from/to the data storage devices;

b. input/output operations used for receiving/sending the input/output data from/to the input/output devices;

c. messages to external application programs;

and data fields.

In other methodologies, other counts such as the count of business rules are used but in no methodology, all the following aspects which are the main constituents of

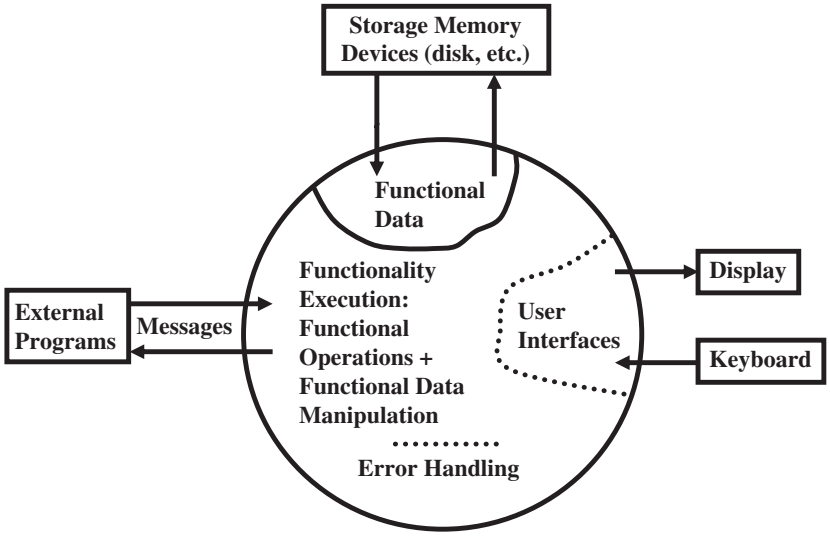

Figure 1 Software size determination model. Adapted from Software Comprehensive Count with Quality Indicators (SCCQI), Jasveer Singh, UKSMA/COSMIC Conference 2012[5], SMEF 2012[6], NESMA Autumn Meeting 2012[7].

software, as shown in Figure 1, are considered and taken into account for the software size measurement[5–7]:

- **a.** size and complexity of the functional data;
- **b.** size, type, and complexity of all the functionality;
- **c.** magnitude of the functional data handling in all the functionality;
- **d.** error handling;
- **e.** number, size, and complexity of the user interfaces;
- **f.** quantity, size, and complexity of the messages for communication with the external application programs and/or sub-programs of the same application program.

All these aspects – the main constituents of software – are considered and taken into account for the software size measurement in the FSSM.

In most of the existing methodologies, a number of constants with complicated assumptions based on the area of the software application are used as multiplication factors, which makes the software size measurement and effort estimation dependent on assumptions. Project planning done based on such estimations may not result in correct time schedules in many cases. In the FSSM, no such multiplication factors are required.

Because of the approach in the existing methodologies that neither all the major components of the software nor all the measureable features of the components are considered for measurement, it cannot be possible to assess the complete size of the software reliably by using them. In the FSSM, these limitations are not there.

It is clear that all the functionality of the application is distributed in different constituent components of the software, these components are called the Software's Measurable Components (SMCs) in the FSSM and are defined as Base Functional Components (BFCs)[1] in the ISO/IEC 14143-1[1] standards. The terms BFC (Base Functional Component)[1] and SMC (Software's Measurable Component) denote the same thing, that

is, the constituent components of the software, and they are used interchangeably in this book henceforth. Measurement of the size of software is achieved by measuring the size of the SMCs (i.e., BFCs)[1] and computing the total size.

The constituent components, that is, the BFCs[1], of the software comprise sub-types of the BFCs[1], defined as the BFC Type[1] in the ISO/IEC 14143-1[1] standards. The BFC Types[1] are, in fact, the features of the SMCs. These features of the SMCs are different for each of the SMC, and are denoted as Software Component's Measurable Features (SCMFs) in the FSSM, that is, the BFC Types[1]. Thus, each Software's Measurable Component (SMC), that is, the BFC[1], has several Software Component's Measurable Features (SCMFs), that is, the BFC Types[1]. The terms BFC Type (Base Functional Component Type)[1] and SCMF (Software Component's Measurable Feature) denote the same thing, that is, different features of the constituent components of the software, and they are used interchangeably in this book henceforth. Measurement of the size of the SMCs (i.e., the BFCs[1]) is achieved by measuring the size of the SCMFs (i.e., the BFC Types[1]), and then computing the size of the SMCs based on the size of the SCMFs. The complete size of the software is calculated based on the size of the SMCs.

The functional size of the SCMFs (BFC Types[1]) is calculated by assigning the Units of Functional Size[2], called Functional Size Units (FSUs) in the FSSM, to the SCMFs. The Functional Size Unit (FSU) in the FSSM is defined as the Software Component's Feature Point (SCFP). Hence, the SCFPs (Software Component's Feature Points) are assigned to various SCMFs (Software Component's Measurable Features) of different SMCs (Software's Measurable Components). All the assigned SCFPs are summed up correspondingly for each SCMF (i.e., the BFC Type[1]) of different SMCs (i.e., the BFCs[1]), and the sums are called the Software Component's Measurable Feature Point Counts (SCFPCs). The SCFPCs form the basis of the Software Component's Feature Measurements (SCFMs). The SCFMs are used for further measurement calculations, estimations, and indications, which are the Software Component's Measurements (SCMs), Software Size and Effort Estimations (SSEEs), and Software Performance Quality Indicators (SPQIs) of 2 types – Software Structural Indicators (SSIs)[5–7] and Software Operational Indicators (SOIs)[5–7] – respectively, as explained in the subsequent chapters.

1.5 CHAPTER SUMMARY

Software is composed of different components which have many features associated with them. The sizes of the components and their features are dependent on the functionality required, and they do not have any fixed proportional size relationship amongst themselves. Functional Size Measurement (FSM)[1] measures the size of the software based on the functionality.

It is highly desirable and advisable that all the relevant and major software components, that is, the BFCs[1], and their features, that is, the BFC Types[1], are taken into account properly in the FSM (Functional Size Measurement)[1] methodology in order to have a realistic measurement of the software size and, consequently, a correct estimation of the effort required for software development to be used for project planning/cost analysis purposes.

In the Functional Software Size Measurement Methodology with Effort Estimation and Performance Indication (FSSM):

a. The Software's Measurable Components (SMCs) are the constituent components of the software. They are the same as the BFCs (Base Functional Components)[1] defined in the ISO/IEC 14143-1[1] standards.

b. The Software Component's Measurable Features (SCMFs) are the different features of the SMCs and are equivalent of the BFC Types[1] defined in the ISO/IEC 14143-1[1] standards.

c. All the major and relevant Software's Measurable Components (SMCs), that is, the BFCs[1], and Software Component's Measurable Features (SCMFs), that is, the BFC Types[1], are taken into account to provide the software size measurements, effort estimations, and performance indications.

d. The Software Component's Feature Points (SCFPs) are assigned to the SCMFs in the Functional Requirements Specifications (FRS) which are the same as the Functional User Requirements (FUR)[1] defined in the ISO/IEC 14143-1[1] standards.

e. The SCFP is the Functional Size Unit (FSU), which is the Unit of Functional Size[2] according to the ISO/IEC 14143-2[2] standards.

f. Summing up the assigned SCFPs correspondingly for different SCMFs produces the Software Component's Feature Point Counts (SCFPCs).

g. SCFPCs form the basis of the Software Component's Feature Measurements (SCFMs).

h. All the Software Component's Measurements (SCMs), Software Size and Effort Estimations (SSEEs), and Software Performance Quality Indicators (SPQIs) of 2 types – Software Structural Indicators (SSIs)[5–7] and Software Operational Indicators (SOIs)[5–7] – are computed from the SCFMs.

i. Full compliance with the ISO/IEC 14143-1[1] standards exists.

EXERCISES

1.1 Which of the following in the FSSM is the equivalent of the BFC (Base Functional Component)[1] of the ISO/IEC 14143-1[1] standards?

A. SMC (Software's Measurable Component).

B. SCMF (Software Component's Measurable Feature).

C. SCFP (Software Component's Feature Point).

1.2 Which of the following in the FSSM is the equivalent of the BFC Type (Base Functional Component Type)[1] of the ISO/IEC 14143-1[1] standards?

A. SMC (Software's Measurable Component).

B. SCMF (Software Component's Measurable Feature).

C. SCFP (Software Component's Feature Point).

1.3 Which of the methods below can be considered as the best practice for software size measurement?

A. Do not use any formal software size measurement methodology and estimate the effort by experience of the experts.

B. Use a software size measurement methodology which measures only a tiny part of the software which has no fixed proportion to the other software parts, nor to the complete software, and estimate the development effort based on the partial and deficient measurement.

C. Use a software size measurement methodology which measures all the major and relevant features of all the components of the software for complete software measurement, and estimate the development effort based on that.

1.4 Which of the following is true for all the software applications?

A. The number of input/output operations is always approximately equal to the number of memory read/write operations in all the software applications.

B. The number of computational operations is always double the number of input/output operations in all the software applications.

C. There is no fixed size proportion relationship between the various types of functionality operations in the different software applications.

1.5 Which of the following relationships exist between the effort required for analyzing, designing, and coding of the functionality and the effort required for data analysis, design, and coding of the data structures (or data tables) in different software applications?

A. Both the efforts – functionality and data – are always equal in all the software applications.

B. The effort for functionality is always 3 times the effort for data in all the software applications.

C. The efforts vary from application to application depending upon the functional requirements.