Improving the Structural Size Measurement Method Through the Assessment of Nested (Multi-Level) Control Structures in UML Sequence Diagram

Hela Hakim[1], Asma Sellami[1], Hanêne Ben Abdallah[2], and Alain Abran[3]

[1] Mir@cl Laboratory, University of Sfax, ISIMS, BP 242. 3021.Sfax-Tunisia.
[2] Higher Colleges of Technology, Dubai, UAE
[3] École de Technologie Supérieure – ETS, University of Quebec (Montréal, Canada)
hakim.hela@yahoo.fr, asma.sellami@isims.usf.tn, hbenabdallah@hct.ac.ae, alain.abran@etsmtl.ca

Abstract. The COSMIC ISO 19761 method is used in software industry as a contributor to estimation improvements and for comparability across projects. The COSMIC method is based on the identification of data movements but it does not consider the data manipulations. To take into account data manipulations associated with data movements at a detailed level of granularity, the measurement of control structures, also referred to as structural size, has been suggested previously. While the previous work focused on the use of constructs (alt, opt, and loop) for one structural level, the multi-level had not been considered. This work proposes refinements to our previous structural size measurement method through the assessment of the nested (multi-level) control structures in the sequence diagram. This refined method proposes detailed measures in different situations where the three constructs can be nested. These measures can be very useful for the software project planning that requires better effort estimation. A web site case study "Digital-Training Center" is used to illustrate and apply the proposed measurement algorithms.

Keywords: Functional size measurement, Structural Size Measurement SSM, COSMIC, ISO 19761, combined fragments, nested combined fragments, mutli-level.

1 Introduction

The software development industry as a whole has a disappointing track record when it comes to completing a project on time and within budget. The Standish Group well-known Chaos Report confirms that only 32% of software development projects are completed successfully within the estimated schedule and budget [21].

Software developers are constantly under pressure to deliver on time and on budget. As a result, many projects focus on delivering functionality at the expense of meeting the details of functionality, defined as the different scenarios describing one functionality [20]. When software details become visible and clients' demands on software quality increase, functionality details can no longer be considered of secondary importance. Many systems fail or fall into disuse precisely because of inadequacies in these details. In an object oriented technology such details are modeled in an UML sequence diagram and some others diagrams related to modeling functionality. In practice, UML sequence diagrams notation is considered as the most popular diagram [7].

Early in the software project lifecycle, details of the specific requirements of the software to be built as well as the staffing needs and other project variables are unclear. The variability in these factors contributes to the uncertainty of project effort estimates. As the sources of variability are further investigated and pinned down, the variability in the project decreases, and thus the variability in the project effort estimates can also decrease.

A number of measurement methods (*e.g.,* Mk II [3], NESMA [4], IFPUG [2], FISMA [5], and COSMIC [1]) are proposed for sizing software functionality and for estimation purposes. Although the design of COSMIC Functional Size Measurement method has been proposed to overcome the shortcomings of the first generation methods, it does not separately assess the data manipulations associated with data movements. For a finer level of granularity of measurement, the structural size measurement with a focus on the nested multi-level control structure has been proposed: it provides development team and managers with a more detailed software size measurements and could lead to a more accurate level of estimates. This work proposes an improvement (refinement) of our previous structural size measurement method for sizing detailed requirements expressed in the form of UML sequence diagrams. In this paper, we focus on how to assess the nested combined fragments (such as opt, alt and loop) within a sequence diagram.

The remainder of the paper is organized as follows: Section 2 presents an overview of the COSMIC method, the Structural Size Measurement method and the UML sequence diagram and its Combined Fragments. It also discusses some related works. Section 3 describes the refined Structural Size (SS) Measurement method. Section 4 illustrates its application combined with the COSMIC through the "Digital-Training Center" case study. Section 5 discusses and presents some limitations of our work. Finally, section 6 concludes the presented work and outlines some of its possible extensions.

2 Background

2.1 COSMIC- FSM: ISO 19761

The COSMIC Functional Size Measurement (FSM) method is becoming popular because of its ability to size different types of software (e.g., business application, real-time software, embedded software, mobile apps, neural networks, etc.) [1]. COSMIC measures the software size from the Functional User Requirements (FUR) in terms of COSMIC Function Point (CFP) units. Some measurement concepts need to be understood before starting the use of COSMIC method: the FURs describe a set of data movements that move data groups consisting of one or more data attributes to and from functional processes. Functional processes are the behavior of the software as viewed by the functional users. A functional user is either a human, an external device or another software that sends or receives data described in FUR. A boundary acts as an interface between the functional users and the software to be measured. In fact, four data movement types occur between functional users, functional processes and persistent storage: Entry (E): The functional user sends data to the functional process through the boundary. eXit (X): The data is sent from the functional process to the user through the boundary. Read (R): The data is moved from persistent storage through the functional process. Write (W): The data is moved to persistent storage through the functional process.

A Functional process always has at least one entry data movement with either a Write (W) to persistent storage or an exit (X) data movement. This will account for at least two data movements and will be sized as two CFP. Thereafter, these data movements are used to get better insights into sizing the software product.

2.2 Structural Size Measurement Method

In our previous work [20], we proposed the Structural Size Measurement (SSM) method. It was designed by following the measurement process recommended in [8].

The main reason for creating such method was the need of detailed measures to quantify data manipulations. As in the COSMIC method where the software functional size is derived by quantifying the FUR [9], the structural size is derived by quantifying the FUR at a detailed level (e.g., at the structural level).

The proposed SSM is applied on the combined fragments of a sequence diagram to measure its structural size (SS). This SS, also named control structural size, refers to the structural size of both conditional control structures CCS and iterative control structures ICS, described respectively through the alt, opt, and loop constructs. The SS of a sequence diagram is defined at a fine level of granularity (*i.e.*, the size of the flow graph of their control structures).

The use of SS requires the identification of two types of data manipulation depending on the structure type CCS (alt and opt combined fragments in the flow graph) and-or ICS (loop combined fragment in the flow graph). Each data manipulation is equivalent to 1 CSM (Control Structure Manipulation) unit. The sequence structural size is computed by adding all data manipulations identified for every flow graph.

2.3 UML Sequence Diagram and its Combined Fragments

The UML Sequence diagrams are a popular dynamic modeling solution in UML because they specifically focus on lifelines, or the processes and objects that live simultaneously, and the messages exchanged between them to perform a function before the lifeline ends [7].

Sequence diagrams can be useful references for software developers who must perform detailed design for each software component, and at different levels of details. Basically, a sequence diagram is drawn to:

- represent the details of a UML use case;
- model the logic of a sophisticated procedure, function, or operation;
- see how objects and components interact with each other to complete a process;
- plan and understand the detailed functionality of an existing or future scenario.

In sequence diagrams, combined fragments are logical groupings, represented by a rectangle, which contain the conditional structures that affect the flow of messages. A combined fragment contains interaction operands and is defined by the interaction operator. The type of combined fragment is determined by the interaction operator in which the type of logic or conditional statement are identified. The interaction operator defines the behavior of the combined fragment that can be used to describe several control and logic structures in a compact and concise manner.

A combined fragment can also contain nested combined fragments or interaction containing additional conditional structures that represent more complex structures that affect the flow of messages. The combined fragments opt, alt, and loop are summarized in Table 1.

Table 1. Description of Alt, opt, and Loop combined fragments

Fragment types	Description
OPT	The combined fragment “opt”: encloses a sequence that might happen or not. The condition under which it occurs can be specified in the guard.
ALT	The combined fragment “alt”: Contains a list of fragments that contain alternative sequences of messages. Only one sequence occurs on any occasion. A guard in each fragment indicates the condition in which it can run. A guard of ELSE indicates a fragment that should run if no other guard is true. If all guards are false and there is no ELSE, then none of the fragments executes.
LOOP	The “loop” combined fragment repeats the condition as it is indicated in the guard a number of times. The “loop” combined fragments have the properties Min and Max that indicate a minimum and maximum number of times to be repeated by the fragment. The default is not a restriction.

2.4 Related Work

Over the past two decades, researchers have proposed object-oriented software metrics to measure the quality of software design and improve the productivity [10]: for example, the CK metrics, MOOD Metrics, etc. [19, 18, 13, 12, 11, 14, 15]. The CK metrics suite can be used for measuring the software complexity [17] serving both as an analyzer and a predictor. To develop better quality software, it is necessary to identify the complexity at module, method and class level (*e.g.*, coupling, cohesion and inheritance that have an effect on complexity). Most of these metrics studies have focused on object-oriented design and are limited only to the static aspect of design (class diagram) size.

In this paper, we focus on how to measure the dynamics aspect of software design (e.g., sequence diagrams) at a fine level of granularity. In other words, measuring the sequence diagram structural size not only at one level (as it is described in our previous work [20]), but also at the multi-level where combined fragments (opt, alt, loop) are nested.

3 Improving the Structural Size Measurement Method through the Assessment of Nested Control Structures

This section presents how the structural size measurement method can be improved by taking into account the in-depth nested (multi-level) control structures in the Sequence Diagram (SD). For this purpose, we propose different algorithms for assessing the nested (multi-level) combined fragments (alt, opt, and loop).

In our earlier work we presented a structural size method for measuring data manipulation expressed by the combined fragments opt, alt and loop based on Sequence diagram descriptions. In this section, we extend our earlier work as follows:

- First, we classify the nested Combined Fragments into three levels and each level is further divided into several cases.
- Second, we propose different algorithms to support the nested multi-level combined fragment in the sequence diagrams and its corresponding flow graph.

3.1 Classifying the Combined Fragments at Different Levels (Muli-Level)

Before sizing the sequence diagram containing the multi-level nested control structures in terms of CSM units, it is important for measurers to distinguish the combination of nested control structures' categories in each level that require more attention. This is to avoid misinterpretation of measurement results.

There are many more cases to be identified for classifying the combined fragments into categories at a multi-level hierarchy (including the following three cases). For simplicity sake, we will focus only on the two first categories.

1. Case when one type of the SD control structures is used at all levels. Note that the number of levels can be one or more.
2. Case when two types of the SD control structures are used at all levels.
3. Case when all the above SD control structures are used at all levels.

1st Category: A Single Control Structure Type used at all Levels

This category includes the alternative ALT Combined Fragment with several alternatives, the choice OPT Combined Fragment, and the iterative LOOP Combined Fragment.

- If the alternative ALT Combined Fragment contains or not one or more nested blocks having the same control structure type (ALT Combined Fragment nested in an ALT Combined Fragment), the following situations should be distinguished:
  - All levels have *n* sequence flows and each alternative has nested ALT Combined Fragment (where *n* is a constant number) (See Algorithm 1.1)

- All levels have *n* sequence flows and some alternatives do not have nested ALT Combined Fragment (See Algorithm 1.1)
- 1st level has *n* sequence flows and all other levels have *p* sequence flows (with *p*! = *n*) (See Algorithm 1.2)
- 1st level has *n* sequence flows and only some other levels have *p* sequence flows (with *p*! = *n*) (See Algorithm 1.2)
- Each level has different number of sequence flows

- If the choice OPT Combined Fragment (two sequence flows where one branch is connected to an end event) and all the nested blocks have the same control structure type (OPT Combined Fragment).
  - Each level contains OPT Combined Fragment restricted to two sequence flows where one sequence flow is linked to end event (because it contains always a single path) (See Algorithm 1.3).

- For an iterative LOOP control structure, we denote the case when each level includes a number of iterative control structures (the number of iterations is arbitrary). The following situations can be observed:
  - Loop with *n* iterations at all levels (See Algorithm 1.4)
  - Loop with *n* iterations in the first level and *p* iterations in the next levels (See Algorithm 1.5)
  - Loop with different iterations in each level (See Algorithm 1.6)

2nd Category: Two different Control Structures used at all Levels

This category may include the following cases:

1st case:

- If the Alt combined fragment in the first level is followed by a nested OPT Combined Fragment, the following situations are considered:
  - All levels having OPT Combined Fragment (See Algorithm 2.1)
  - Some levels having OPT Combined Fragment in which the following cases are established:
    - Each alternative in the first level contains the Opt combined fragment and the other levels may have or not the OPT Combined Fragment (See Algorithm 2.2)
    - Each alternative may or not have the OPT Combined Fragment in all levels (See Algorithm 2.3)

2nd case:

- The OPT Combined Fragment in the first level is followed by a nested ALT Combined Fragment.
- The OPT Combined Fragment in the first level is followed by a nested LOOP Combined Fragment (See Algorithm 2.4).

3rd case:

- The LOOP Combined Fragment in the first level is followed by a nested OPT Combined Fragment.
- The LOOP Combined Fragment in the first level is followed by a nested ALT Combined Fragment.

3rd Category: All different Control Structures Type are used at all Levels

In this category, all possible combinations of the different control structures can be used.

3.2 Sizing the Nested (Multi-Level) Control Structures in the Sequence Diagram

With the information from the sequence diagram and its corresponding flow graph, measurers can quickly identify the combined fragments at different levels (including the nested level). Thereafter, by applying the proposed algorithms (See Tables 2 and 3), the detailed structural size of a sequence diagram can be generated.

Let:

- SDm be a Sequence Diagram containing a nested (multi-level) combined fragments,
- GSS be its corresponding flow graph,
- GSS(SDm) be the graph-based structural size function expressing the Structural Size of the whole Sequence diagram, and
- GSS(F) be the graph-based structural size function expressing the Structural Size of the combined fragment within the Sequence diagram.

The sequence diagram structural size is derived as the sum of the Structural size of all the combined fragments (multi-level) as shown by equation (1):

$$SS(SDm) = \sum_{i=1}^{n} SS(F_i) \quad (1)$$

where:

- n is the number of combined fragments F_i in the SDm,
- $SS(F_i)$: the Structural size of a fragment F_i

Since each fragment represented in a SD can be itself decomposed into a new fragment that refines it, each Fi can contain (or not) one or more nested control structures. The structural sizes of these fragments are derived by using algorithms as illustrated in section 3.2.2. Note that:

- GSSic(F) is the (sub)graph representing the structural size of the iterative control structure SS(ICS),
- GSScc(F) is the (sub) graph representing the structural size of the conditional control structure SS(CCS) that may include alt combined fragment and-or opt combined fragment.

To measure the structural size of a fragment, we propose algorithms based on the defined strategy.

Proposed algorithms for the 1st category.

When a single control structure type is used at all levels, we propose to use Algorithm 1.1 and Algorithm 1.2. Algorithm 1.1 represents how sizing an ALT Combined Fragment nested within another ALT Combined Fragment in the same SDm, where all levels have *n* alternatives (with *n* a fixed number). Algorithm 1.2 illustrates the ALT Combined Fragment nested within another ALT Combined Fragment where the 1st level has *n* alternatives and all other levels have *p* alternatives (with $p! = n$).

Proposed algorithms for the 2nd category

When two types of the SD control structures are used, we propose to use Algorithm 2.1, Algorithm 2.2 and Algorithm 2.3.

1st case:

Algorithm 2.1 expresses the case when the Alt combined fragment is followed by Opt combined fragment in each level. While Algorithm 2.2 and 2.3 represent the case when the alternative Alt combined fragment is followed by some Opt combined fragments, this situation represents two cases:

- Algorithm 2.2 represents that each sequence flow in the Alt combined fragment (i=0) contains Opt combined fragment, and other alternatives in the followed levels may contain or not an Opt combined fragment.
- Algorithm 2.3 represents that each alternative in the Alt combined fragment (i=0) may or not have choice Opt combined fragment in all levels (Algorithm 2.3).

Table 2. Algorithms and measurement formulas for the 1st category

Algorithm 1.1. ALT combined fragmentnested within anotherALT combined fragment in the SDm (*All levels have n alternatives (with n a fixed number) and each node has a nested Combined Fragment F*

Begin
inti=0; // i = [0,1] i is the root or the1st level
alt=n; // n is the number of nodes in each level i
i=1;
int j=1// j = [1..h] where h is the number of nodes in the 1st level
i=1; //Xij
For each level i
For each node x_{ij}
out-degree (x_{ij}) = n

$$SS_{cc}(x_{ij}) = \text{GSScc}(x_{\text{ij}})$$

End for
i++;
int m = [1..d]; where d is the number of nodes in the second level(i=2)
For each level i
For each node y_{im}
out-degree (y_{im}) = n

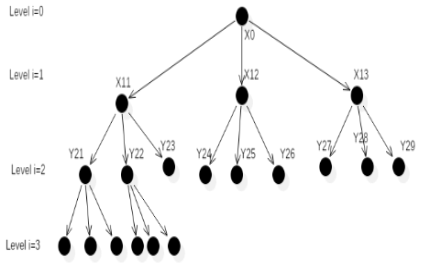
$$SS_{cc}(y_{im}) = \text{GSScc}(y_{\text{im}})$$

End for
End for

$$SS(F) = \sum_{m=1}^{d} \left(SS_{cc}(y_{im}) * (SS_{cc}(x_{ij})/n)^{i}\right)$$

= n*d
=n*n^{i}
=n^{i+1}
End for
End
Note that d= n^{i} if there is a nested Alt combined fragment in each alternative
SS(F)= n^{i+1}
Else

$$SS(F) = \sum_{m=1}^{d} \left(SS_{cc}(y_{im}) * (SS_{cc}(x_{ij})/n)^{i}\right)$$



Flow graphs modeling Algorithm 1.1

Algorithm 1.2 Alt combined fragment nested in an Alt combined fragment*: 1st level has n alternatives and all other levels have p alternatives (with p! = n)*

Begin

int i=0 ; // i=[0,1] i is the root or the1st level

alt=n; // n is the number of nodes in each level i

i=1;

int j=0// j = [0..h] where h is the number of nodes in the first level

i=1;//xij

For each level i

For each node x_{ij}

out-degree (x_{ij}) = n

$SS_{cc}(x_{ij}) = \text{GSS}(x_{ij})$

End for

i++;

int m = [1..d]; where d is the number of nodes in the second level (i=2)

For each level i

For each node y_{im}

out-degree (y_{im}) = p

$SS_{cc}(y_{im}) = \text{GSS}(y_{im})$

End for

End for

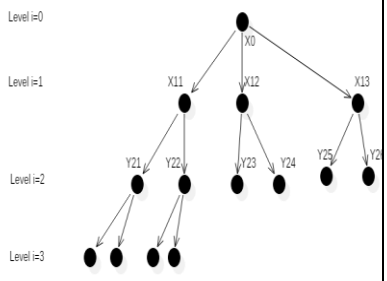
$$SS(F) = \sum_{m=1}^{d} \left(SS_{cc}(y_{im}) * (SS_{cc}(x_{ij})/n)^{i}\right)$$

= (p*d) CSM

End for

End

$$SS(F) = \sum_{m=1}^{d} \left(SS_{cc}(y_{im}) * (SS_{cc}(x_{ij})/n)^{i}\right)$$



Flow graphs modeling Algorithm1.2

Algorithm 1.3 Opt combined fragmentand all the nested blocks have the same control structure type (Opt combined fragment)

Begin

int i=0; // i = [0, l] i is the root or the1st level

alt=n=1; // n is the number of nodes in each level i

i=1;

For each level i

For each node x_i

out-degree (x_i) = n=1

$SS_{cc}(x_i) = \text{GSS}(x_i)$

End for

End for

i++;

For each level i

For each node y_i

out-degree (y_i) = 1

$SS(y_i) = \text{GSS}(y_i)$

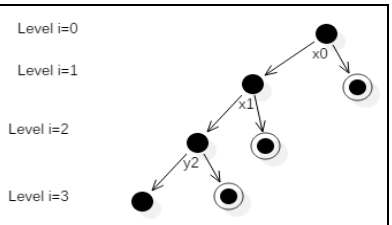
End for

End for

$SS(F) = \left(SS_{cc}(y_i) * (SS_{cc}(x_i)/n)^{i}\right)$

= 1 CSM

End



Flow graphs modeling Algorithm 1.3

Algorithm 1.4 Each level contains iterative control structures (LOOP Combined Fragment) where the number of iterations is fixed (r)

inti=0; // i = [0..l] where i is the root or the1st level
int itr=r; // r is the number of iterations
i=1;
For each level i having r number of iterations
For each node x_i
out-degree (x_i) = r

$$SS_{ic}(x_i) = \text{GSS}(x_i)$$

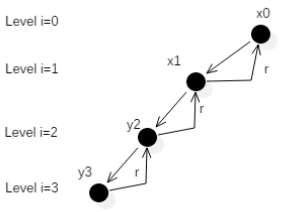
End for
i++;
For each level i having r number of iterations (r is a fixed number equal to the fixed number in the previous level)
For each node y_i
out-degree (y_i) = r

$$SS_{ic}(y_i) = \text{GSS}(y_i)$$

End for
End For

$$SS(F) = GSS(x_i) * GSS(y_i) = r^i$$

End for



Flow graphs modeling Algorithm 1.4

Algorithm 1.5 Each level contains iterative control structures (LOOP Combined Fragment) where the number of iterations is fixed in the first level and s iterations in the next levels

inti=0; // i = [0,1] i is the root or the1st level
int it r= r; // r is the number of iterations in the first level
int itr= s; // s is the number of iterations in other levels
i=1;
For each level i having r number of iterations
For each node x_i
out-degree (x_i) = r

$$SS_{ic}(x_i) = \text{GSS}(x_i)$$

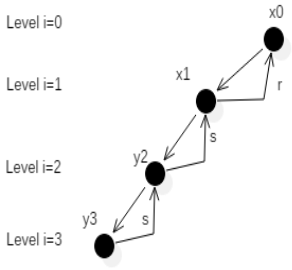
End for
i++
For each level i having s number of iterations which is different to the previous level
For each node y_i
out-degree (y_i) = s

$$SS_{ic}(y_i) = \text{GSS}(y_i)$$

End for
End For

$$SS(F) = GSS(x_i) * GSS(y_i) = r * s^i$$

End for



Flow graphs modeling Algorithm 1.5

Algorithm 1.6 Each level contains iterative control structures (LOOP Combined Fragment) where the number of iterations is different from each level

int itr=r; // r is the number of iterations in the first level
int itr=s, t; // s and tare respectively the number of iterations in the next levels
i=1;
For each level I having r number of iterations
For each node x_i
out-degree (x_i) = r

$$SS_{ic}(x_i) = \text{GSS}(x_i)$$

End for
i=2;
For each level i having s number of iterations (s is different from the previous level)
For each node y_i
out-degree (y_i) = s

$$SS_{ic}(y_i) = \text{GSS}(y_i)$$

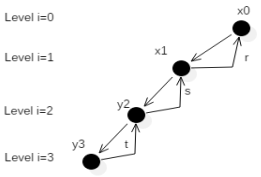
End for
End For
i++;
For each node y_i
out-degree (y_i) = t

$$SS_{ic}(y_i) = \text{GSS}(y_i)$$

End for

$$SS(F) = GSS(x_i) * GSS(y_i) = r * s * t$$

End for



Flow graphs modeling Algorithm 1.6

Table 3. Algorithms and measurement formulas for the 2nd category

Algorithm 2.1: Alt combined fragment in the first level is followed by nested opt Combined Fragment F in all levels

Begin
inti=0 ; // i=0…l is the number of levels
Alt=n ; // n is the number of nodes in the level i=1;
int j=1// j=1..h where h is the number of nodes in 1st level
For each level i having choice Opt combined fragment
For each alternative in level i
For each node x_{ij}
out-degree (x_{ij}) = 1

$$SS_{cc}(x_{ij}) = \text{GSS}(x_{\text{ij}})$$

End for
i++
int m=1….d; where d is the number of nodes in 2nd level
For each level i
For each node y_{im}
out-degree (y_{im}) = 1

$$SS_{cc}(y_{im}) = \text{GSS}(y_{\text{im}})$$

End for
End For

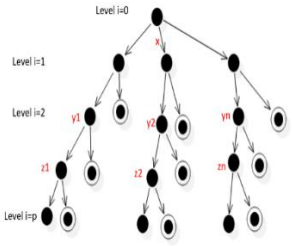
$$SS_F = \sum_{m=1}^{d} \left(SS_{cc}(y_{im}) * (SS_{cc}(x_{ij}))^i\right)$$

$$SS_F = SS_{cc} = \sum_{i=1}^{P} \left(\text{GSS}(x_i) + \text{GSS}(y)\right)_i$$

End for

$$SS_{seq} = \sum_{j=1}^{n} \left(\sum_{i=1}^{P} \left(\text{GSS}(x) + \text{GSS}(y)\right)_i \right)_j$$

End for
End



Flow graphs modeling Algorithm 2.1

Algorithm 2.2: Each sequence flow in the Alt combined fragment (i=0) contains Opt combined fragment, and other alternatives in the followed levels may have or not an Opt combined fragment

Begin
Int i=0 ; // i is the number of levels
outdegree(x)=n;
j=1..n ; //j number of nodes from 1 to n
i=1..p; i//number of levels from 1 to p
Alt=n ; // n is the number of nodes
For (j=1, j<n, j++)
If the opt Combined Fragment F==True
out-degree (y) = 1

$$SS_{cc} = \mathrm{GSS}(y)$$

End if
i++
For each level i contains or not an Opt combined fragment
For each node z
Ifthe opt Combined Fragment CF=True
out-degree (z) = 1

$$SS_{cc} = \mathrm{GSS}(z)$$

$$SS_{cc} = \sum_{n \in li, p \in l_{i+1}; S_{cc}} \mathrm{GSS}(y) + \mathrm{GSS}(z)$$

Else

$$SS_{cc} = \mathrm{GSS}(z) = 0$$

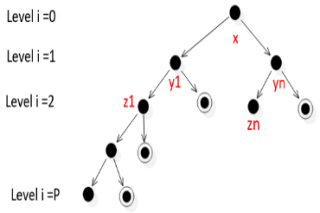
$$SS_{cc} = \sum_{n \in li, p \in l_{i+1}} \mathrm{GSS}(y) + \mathrm{GSS}(z) = GSS(y) + 0$$

$$= GSS(y)$$

End if
End for
End for

$$SS_F = \sum_{i=1}^{P} (SS_{cc})_i$$

End For
End



Flow graphs modelingAlgorithm2.2

Algorithm 2.3: Each sequence flow in the Alt combined fragment may or not have Opt combined fragment in all levels

Begin
inti=0; // i is the number of levels

outdegree(x)=n;
inti=0 ; // I is the number of level
outdegree(e)=n;
j=1..n ; //j number of nodes from 1 to n
i=1..p; i//number of levels from 1 to p
Alt=n ; // n is the number of nodes
For (j=1, j<n, j++)
If Opt combined fragment=True
out-degree (x) = 1

$$SS_{cc} = \mathrm{GSS}(x)$$

End if
i++
For each level i contains or not opt
For each node y
If Opt combined fragment=True
out-degree (y) = 1

$$SS_{cc} = \mathrm{GSS}(y)$$

$$SS_{cc} = \sum_{n \in li, p \in l_{i+1}; BS_{cc}} \mathrm{GSS}(x) + \mathrm{GSS}(y)$$

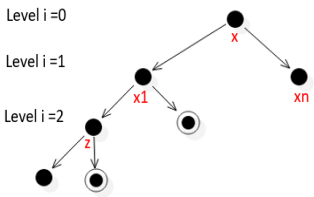
Else

$$SS_{cc} = \mathrm{GSS}(y) = 0$$

$SS_{cc} = \sum_{n \in li, p \in l_{i+1}; SS_{cc}} \mathrm{GSS}(x) + \mathrm{GSS}(y) =$
$GSS(x) + 0 = GSS(x)$
End if
End for
End For

$$SS_F = SS_{cc} = \sum_{i=1}^{P} (SS_{cc})_i$$

End For
End



Flow graphs modelingAlgorithm2.3

Algorithm 2.4: LOOP Combined Fragment nested in the Alt combined fragment: Each alternatives *contains LOOP Combined Fragment (with u a fixed number of iterations)*

Begin
int i=0; // i is the number of levels
outdegree(x)=0;
Alt=n; // n is the number of nodes
For each level i
For each node x
out-degree (x) = n*i

$$SS_{cc} = \text{GSS}(x)$$

end for
i++
For each level i having *u* number of iterations
For each node y
out-degree (y) = u

$$SS_{ic} = \text{GSS}(y)$$

end for
i++
For each level I (i=2) having w number of iterations (wis a random number and it is NOT equal to the previous levels)
For each node z //z represents all nodes in any levels except level 0 and 1.
out-degree (z) = w

$$SS_{ic} = \text{GSS}(z)$$

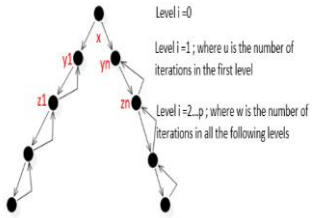
End for
End For

$$SS_{ic} = GSS(y) * GSS(z) = w^i$$

End for

$$SS_{Fi} = SS_{(ic_)} + SS_{(cc_)}$$

end for
End



Flow graphs modeling Algorithm 2.4

4 Case Study: Illustrative Example

This section illustrates the application of our proposed method through the case study "Digital-Training Center" web site. It considers the interactions among trainings in this web site, as an example of UML2.x SD modeling the behaviors of a distributed system (see Fig.1 and Fig.2). The web site has independent components: the training officer, the home page, the custom page, the training page and the authentication page.

We model the update of the training catalog by the training officer (Fig. 2): the training officer must authenticate at first (Fig. 1). He enters the web site URL and he can be redirected directly to the custom page if he has chosen, in the last connection, the option to remain connected for a limited duration. Otherwise, he is redirected to the authentication page. There are three possible cases:

1. successful authentication;
2. missing information - in this case the site asks the user to complete them;
3. wrong login and-or wrong password - in this case the site asks the training officer to correct them.

In the second case, domain name system (DNS) attacks may accidentally occur. Fig. 2 depicts the update of the training catalog. The training officer requests to update the training catalog: he is redirected to the training page where he has three possible alternatives: i) he can add new training and create a calendar session; ii) he can choose a training and update a session, iii) he can remove a training - the list of participants to this training is displayed.

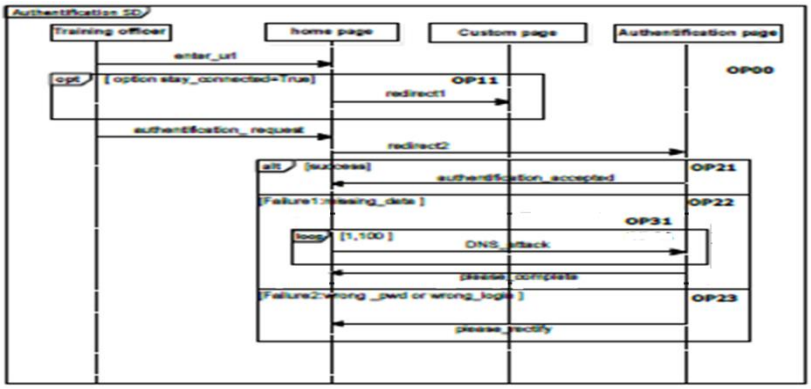


Fig. 1. Authentication SD

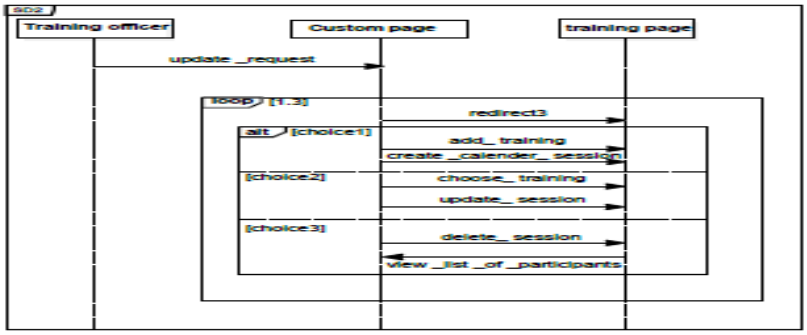


Fig. 2. Update training catalog diagram SD

Table 4. Detailed COSMIC FS and SS Measurements - Digital-Training Center Web Site

SD	Description of interactions	FS(SD)	SS(SD)	SS(SDm)	Total
Authentication (Fig1)	enter_url	1 CFP			**8 CFP** **102 CSM**
	redirect1	1 CFP	1 CSM		
	authentification_request	1 CFP			
	redirect2	1 CFP			
	authentification_accepted	1 CFP		100+1=101CSM	
	DNS_attack	1 CFP			
	please_complete	1 CFP			
	please_rectify	1 CFP			
Update training catalog (Fig2)	update _request	1CFP	0 CSM		**8CFP** **9CSM**
	redirect3	1 CFP		3*3=9 CSM	
	add_ training	1 CFP			
	create_calender	1 CFP			
	choose_ training-session	1 CFP			
	update_ session	1 CFP			
	delete_ session	1 CFP			
	view _list _of _participants	1 CFP			

5 Discussion and Limitations

User requirements are the basis core of software projects and sizing requirements is a crucial task for software project planning. Hence, if these requirements are poorly described (in the analysis phase of the SLC) and modeled (in the design phase), the software development/maintenance planning will be vulnerable. In practice, well detailed User requirement are not always available since they require much time and comprehension. However, the more time spent at the beginning of the process, the less time will be required later. Consequently, detailed descriptions of user requirements are needed: these detailed requirements can be easily represented in the form of sequence diagrams that can be measured.

In our work, the proposed algorithms dealing with the nested (muti-level) Combined Fragments in the Sequence Diagram have been illustrated through the case study "Digital-Training Center". To explore the efficiency of our refined measurement-based algorithms, these algorithms must next be applied and tested in an industrial setting, particularly in projects having a complex functionality that requires to be modeled with these nested multi-levels.

6 Conclusion and Future Work

This paper extends our previous work [20] on a structural size measurement method by proposing additional algorithms for measuring the combination of all nested (multi-level) combined fragments. This extension is based on the most popular combined fragments (ALT, OPT and LOOP) identified in the sequence diagrams (SD) as a whole (without parsing the SD). More specifically, we proposed several algorithms to measure the different categories of partial order between the events. We also proposed some additional contributions for refining our proposed structural size method by focusing on the different combinations of nested combined fragments of the SD in order to support nested multi–level Combined Fragment. In our future work, we will focus on automating the combination of functional and the extended Structural Size Measurement methods. Furthermore, we plan to investigate the combination of these measures to improve projects and tests effort estimation models by taking into account both Functional size measurement and this refined Structural Size Measurement. Moreover, this combination of measures should be explored by scrum masters to evaluate and prioritize user stories and make appropriate decisions within an agile context.

References

1. ISO/IEC 19761, Software Engineering – COSMIC: A Functional Size Measurement Method, International Organization for Standardization, ISO, Geneva, (2011).

2. ISO/IEC 20926, Software and Systems Engineering – Software measurement – IFPUG Functional Size Measurement Method Geneva, International Organization for Standardization, ISO (2009).

3. ISO/IEC 20968, Software Engineering – Mk II Function Point Analysis – Counting Practices Manual, International Organization for Standardization, ISO, Geneva, (2002).

4. ISO/IEC 24570, Software Engineering – NESMA Functional Size Measurement Method Version 2.1 – Definitions and Counting Guidelines for the Application of Function Point Analysis, International Organization for Standardization, ISO, Geneva, (2005).

5. ISO/IEC 29881, Information Technology – Software and Systems Engineering – FiSMA 1.1 Functional size Measurement Method, International Organization for Standardization, ISO, Geneva, (2008).

6. ISO/IEC 14143-1, Information Technology – Software Measurement-Functional Size Measurement–Part 1: Definition of Concepts, International Organization for

Standardization, ISO, Geneva, (2007).

7. OMG, Documents Associated with Unified Modeling Language (UML), V2.4.1, (2011)
8. Abran A, Software Metrics and Software Metrology, John Wiley & Sons, Inc. IEEE Computer Society Press, (2010).
9. COSMIC Group, Non-Functional & Project Requirements with COSMIC: Experts Guide, , (2020)
10. Booch. G. Object-Oriented Design and Application, Benjamin/Cummings, Mento Park, CA. (1991)
11. Brotoeabreu. F. D "Metrics Set", in Proc. ECOOP'95 Workshop Metrics. (1995)
12. Briand. L. C, Daly. J. W and Wust. J. K. "A Unified Framework for Cohesion Measurement in Object Oriented Systems", Empirical Software Eng., 1, 1, 65- 117. (1998).
13. Bieman. J. M and B.K. Kang. "Cohesion and Reuse in an Object-Oriented System", in Proc. Symp. Software Reliability, 259-26. (1995)
14. Chae, H.S, Kwon. Y. R and Bae.D.H. Cohesion Measures for Object-Oriented Classes", Software practice and Experiences, 30, 12, 1405-1431. (2000)
15. Churcher. N. I and Shepperd. M. J. Comments on "A Metric Suite for Object-Oriented Design", IEEE Trans. on Software Engineering. 21, 263-265. (1995)
16. COSMIC: The COSMIC Functional Size Measurement Method, Version5.0., Measurement Manual, COSMIC Group https://cosmic-sizing.org/measurement-manual/ (March 2020)
17. El-Emam, K. Object-oriented metrics: A review of theory and practice. In: Erdogmus, H., Tanir, O. (eds.) Advances in Software Engineering, pp. 23–50. Springer, New York. (2002)
18. Jacobson, I., Griss, M. and Johnsson, P, Software Reuse, Architecture, Process, and Organization for Business Success. Addison-Wesley. (1997)
19. Lorenz, Mark & Kidd Jeff: "Object-Oriented Software Metrics", Prentice Hall, 1994
20. Sellami, A., Hakim, H., Abran, A., and Ben-Abdallah, H., "A Measurement Method for Sizing the Structure of UML Sequence Diagrams", Information and Software Technology – Elsevier IST - Elsevier, Vol. 59, March, pp. 222-232 http://dx.doi.org/10.1016/j.infsof.2014.11.002 (2015)
21. Standish Group, The CHAOS Report, Boston (2009)