AUTHOR QUERY FORM

ELSEVIER	**Journal: DATAK** **Article Number: 1386**	**Please e-mail or fax your responses and any corrections to:** **E-mail: corrections.esch@elsevier.spitech.com** **Fax: +1 619 699 6721**

Dear Author,

Please check your proof carefully and mark all corrections at the appropriate place in the proof (e.g., by using on-screen annotation in the PDF file) or compile them in a separate list. Note: if you opt to annotate the file with software other than Adobe Reader then please also highlight the appropriate place in the PDF file. To ensure fast publication of your paper please return your corrections within 48 hours.

For correction or revision of any artwork, please consult http://www.elsevier.com/artworkinstructions.

Any queries or remarks that have arisen during the processing of your manuscript are listed below and highlighted by flags in the proof. Click on the 'Q' link to go to the location in the proof.

Location in article	Query / Remark: click on the Q link to go Please insert your reply or correction at the corresponding line in the proof
Q1	Please confirm that given names and surnames have been identified correctly.
Q2, Q6, Q7	Please provide publisher location.
Q3, Q4	Please check author name, and correct if necessary.
Q5	Please provide publisher location

Please check this box if you have no corrections to make to the PDF file. ☐

Thank you for your assistance.

Data & Knowledge Engineering xxx (2012) xxx–xxx

Contents lists available at SciVerse ScienceDirect

Data & Knowledge Engineering

journal homepage: www.elsevier.com/locate/datak

Approximation of COSMIC functional size to support early effort estimation in Agile

Ishrar Hussain*, Leila Kosseim, Olga Ormandjieva

Department of Computer Science and Software Engineering, Concordia University, 1400 de Maisonneuve Blvd. West, Montreal, Quebec, Canada H3G 1M8

ARTICLE INFO

Article history:
Received 22 December 2010
Received in revised form 4 July 2011
Accepted 25 June 2012
Available online xxxx

Keywords:
Software requirements
Functional size measurement
Text mining
Natural language processing
Agile development processes

ABSTRACT

The demands in the software industry of estimating development effort in the early phases of development are met by measuring software size from user requirements. A large number of companies have adapted themselves with Agile processes, which, although, promise rapid software development, pose a huge burden on the development teams for continual decision making and expert judgement, when estimating the size of the software components to be developed at each iteration. COSMIC, on the other hand, is an ISO/IEC international standard that presents an objective method of measuring the functional size of the software from user requirements. However, its measurement process is not compatible with Agile processes, as COSMIC requires user requirements to be formalised and decomposed at a level of granularity where external interactions with the system are visible to the human measurer. This time-consuming task is avoided by agile processes, leaving it with the only option of quick subjective judgement by human measurers for size measurement that often tends to be erroneous. In this article, we address these issues by presenting an approach to approximate COSMIC functional size from informally written textual requirements demonstrating its applicability in popular agile processes. We also discuss the results of a preliminary experiment studying the feasibility of automating our approach using supervised text mining.

1. Introduction

The agile development process breaks down the software development lifecycle into a number of consecutive iterations that increases communication and collaboration among stakeholders. This type of process focuses on the rapid production of functioning software components along with providing the flexibility to adapt to emerging business realities [1]. In practice, agile processes have been extended to offer more techniques, e.g. describing the requirements with user stories [2]. Instead of a manager estimating developmental time and effort and assigning tasks based on conjecture, team members in agile processes use effort and degree of difficulty in terms of points to estimate the size of their own work, often with biased judgment [3]. Hence, an objective measurement of software size is crucial in the planning and management of agile projects.

We know that effort is a function of size [4], and a precise estimation of software size right from the start of a project life cycle gives the project manager confidence about future courses of action, since many of the decisions made during development depend on the initial estimations. Better estimation of size and effort allows managers to determine the comparative cost of a project, improve process monitoring, and negotiate contracts from a position of knowledge.

* Corresponding author.
E-mail addresses: h_hussa@cse.concordia.ca (I. Hussain), kosseim@cse.concordia.ca (L. Kosseim), ormandj@cse.concordia.ca (O. Ormandjieva).
URLs: http://www.linkedin.com/in/ishrar (I. Hussain), http://users.encs.concordia.ca/~kosseim (L. Kosseim), http://users.encs.concordia.ca/~ormandj (O. Ormandjieva).

0169-023X/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.datak.2012.06.005

The above has led the industry to formulate several methods for functional size measurement (FSM) of software. In 1979, Allan Albrecht first proposed FSM in his work on function point analysis (FPA) [5], where he named the unit of functional size as "Function Point (FP)". His idea of effort estimation was then validated by many studies, like [6,7], and, thus, measuring the functional size of the software became an integral part of effort estimation. Over the years, many standards have been developed by different organisations on FSM methods, following the concepts presented in Albrecht's FPA method. Four of these standards have been accepted as ISO standards: they are IFPUG [8], Mark II [9], NESMA [10] and COSMIC [11].

In recent years, many studies (e.g. [12–14]) have attempted to automate the process of different functional size measurement methods, but, to our knowledge, none has addressed this problem by taking the textual requirements as input to start the automatic measurement process. In addition, all these work depended on extracting manually the conceptual modeling artifacts first from the textual requirements, so that a precise functional size measurement can be performed. On the other hand, the work documented in this paper aims to develop a tool that would automatically perform a quicker approximation of COSMIC size without requiring the formalisation of the requirements. This is in response to the high industrial demands of performing size estimation during agile development processes, where formalisation of requirements are regarded as costly manipulation, and, thus, ignored during size estimation. Our methodology extends the idea presented in the Estimation by Analogy approach [15] and the Easy and Quick (E&Q) measurement approach, that was originated in the IFPUG standard [16]. The applicability of this approach in COSMIC was manually demonstrated by [17].

2. Background

2.1. COSMIC

For the purpose of this research, we have chosen to use the COSMIC FSM method developed by the Common Software Measurement International Consortium (COSMIC) and now adopted as an international standard [11]. We chose this method in particular, because it conforms to all ISO requirements [18] for FSM, focuses on the "user view" of functional requirements, and is applicable throughout the agile development life cycle. Its potential for being applied accurately in the requirements specification phase compared to the other FSM methods is demonstrated by the study of [19]. Also, COSMIC does not rely on subjective decisions by the functional size measurer during the measurement process [11]. Thus, its measurements, taken from well-specified requirements, tend to be same among multiple measurers. This is particularly important for validating the performance of our automatic size measurements.

In COSMIC, size is measured in terms of the number of *Data-movements*, which accounts for the movement of one or more data-attributes belonging to a single *Data-group*. A data-group is an aggregated set of data-attributes. A *Functional Process*, in COSMIC, is an independently executable set of data-movements that is triggered by one or more *triggering events*. A triggering event is initiated by an *actor* (a functional user or an external component) that occurs outside the boundary of the software to be measured. Thus, a functional process holds the similar scope of a use case scenario, starting with the triggering event of a user-request and ending with the completion of the scenario. Fig. 1 illustrates the generic flow of data-groups from a functional perspective, presented in the COSMIC standard [11].

As shown in Fig. 1, the data-movements can be of four types: Entry, Exit, Read and Write. An Entry moves a data-group from a user across the boundary into the functional process, while an Exit moves a data group from a functional process across the

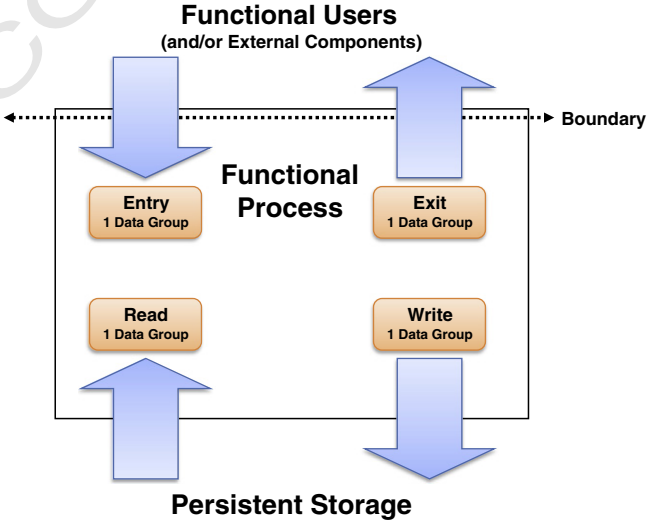


Fig. 1. Generic flow of data-groups in COSMIC [11].

Please cite this article as: I. Hussain, et al., Approximation of COSMIC functional size to support early effort estimation in Agile, Data Knowl. Eng. (2012), doi:10.1016/j.datak.2012.06.005

boundary to the user requiring it. A Write moves a data group lying inside the functional process to persistent storage, and a Read moves a data group from persistent storage to the functional process.

COSMIC counts each of these data-movements as one *CFP* (COSMIC Function Point) of functional size, and measures the size of each of the functional processes separately. It then adds the sizes of all the functional processes to compute the total size of the system to be measured.

COSMIC offers an objective method of measuring functional size. It is built to be applied in the traditional processes of software development, where documentation of requirements using formalisms and templates is required. However, over the years, the IT industry has recognised the traditional processes to cause many problems including delays and is now increasingly moving towards agile development processes [20], such as Scrum [2], an agile approach that does not impose documentation templates or formalisms on requirements.

2.2. Size measurement in agile development processes

Agile development processes are driven by the motto of delivering releases as quickly as possible [1]. Planning an iteration in an agile project involves estimating the size of the required features as the first step. Fig. 2 shows the steps of iteration planning in agile.

The size of every agile iteration is subjectively estimated by means of user requirements that are written less formally than use case descriptions. These textual requirements, which are mostly available in the form of smart use cases [21] or user-stories [2], although, do not provide detailed description of the scenarios like those found in use cases, they must hold "enough details" to perform the size estimation [2]. Size measurement methods in agile development processes include story-points [3] and smart estimation [21], and depend on the subjective judgment of human experts, and, therefore, are prone to biases and errors [3].

In an agile development process, the lack of formalism in requirements restricts FSM methods, like COSMIC, to be applied for measuring the functional size of an iteration. For example, from the discussion in Section 1, it can be understood that the number of data-groups, which is necessary to be known to carry out COSMIC FSM, cannot be identified by the measurer from a set of requirements statements alone unless he/she is supplied with a complete list of available data-groups that requires formalising the requirements with conceptual model (e.g. a domain model).

Our work presents an alternative solution to estimate the COSMIC functional size in agile that does not require the use of formalism in requirements; instead, it proposes an objective way of approximating the COSMIC functional size of a functional process (i.e. a use case) that is described by an informally written set of textual requirements, in forms likely to be used in agile size estimation.

3. Related work

One of the leading work done in the area of automating COSMIC FSM is by Diab et al. [13], where the authors developed a comprehensive system called, *μ*cROSE, which accepts state charts as inputs to measure the functional size of real-time systems only. We find their work to be largely dependent on a set of hard-coded rules for mapping different objects of interest to different COSMIC components, and also require C++code segments to be attached with the state transitions and supplied as input too, so that data-movements can be identified. In [13], the authors present a brief validation of their work by an expert, testing their system against one case study only, where, according to the authors, it resulted in some erroneous measurement outputs.

Another related work is that of Condori-Fernández et al. [12], who presented step by step guidelines to first derive manually the UML modeling artifacts, e.g. the use case models and system sequence diagrams from the requirements, and then, apply their set of rules for measuring the COSMIC functional size of the system from the UML models. Their approach was validated on 33 different observations, showing reproducible results with 95% confidence. A similar approach is presented by Marín et al. [22] that uses an automated tool, called *OOmCFP*. However, it also depends on conceptual requirements models to be manually prepared, so that COSMIC functional size can be automatically measured.

Most of the related work in this field has attempted to perform a precise measurement of COSMIC functional size that rely on tedious manual processing to extract conceptual modeling artifacts and require formalisation of the requirements, and, therefore, are not applicable to agile development processes. On the other hand, the work of [17] presents a fully manual approach of quick approximation of COSMIC size from textual requirements without extracting COSMIC modeling artifacts. It first classifies past projects into fuzzy size classes (e.g. Small, Medium, Large, Very Large,…), finds the common traits within the concepts used in software belonging to the same size class, and, finally, allows a human measurer to discover similar traits in the new software

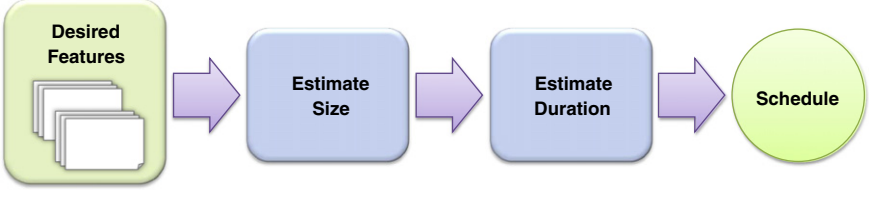


Fig. 2. Steps of iteration planning in agile (as presented in [3]).

Please cite this article as: I. Hussain, et al., Approximation of COSMIC functional size to support early effort estimation in Agile, Data Knowl. Eng. (2012), doi:10.1016/j.datak.2012.06.005

component, so that the measurer can estimate its COSMIC size by drawing analogy with past projects. We find a good potential of this work to be applied in the environment of agile process that demand quicker size estimation.

The goal of our work described in this article is to develop a fully automated tool that would do quicker estimation of COSMIC size using textual requirements written in unrestricted natural language as input, making it favorable for agile processes. We extend the idea of [17] by finding common traits, or 'features', among software projects of the same size classes, but looking for linguistic features within the textual requirements, and use supervised text mining methods to automate the process.

4. Methodology

Our methodology requires the historical data of an organisation to be stored for the purpose of generating a dataset for training and testing our application. The historical dataset needs to contain sets of textual user requirements written in any quality, where each set corresponds to a unique functional process, along with their respective functional size in COSMIC to be recorded by human measurers. We present our detailed methodology in the following sections.

4.1. CFP measurement

In the cases where a historical database is not available or is not in the form required by our approach, our first step would then be to build the historical database by manually measuring the size of the functional processes in units of *CFP* (COSMIC Function Point) and storing these measurements in the database. The available textual description of the user requirements corresponding to each functional process is used for this purpose. Here, for each requirements statement belonging to a functional process, the human measurer first identifies how many different types of data-movements are expressed by the statement, and then, how many data-groups participate in each of the types of data-movements present in the statement. Following COSMIC, the sum of number of data-groups for each type of data-movements indicates the total *CFP* size of one requirements statement. The measurer repeats this step for the rest of the requirements statements within the functional process and summing up their sizes results in the *CFP* count for the whole functional process. The measurer then again adds the *CFP* sizes for each of the functional processes to obtain the respective *CFP* count of the whole system. Table ftab: CFP Calculation illustrates the *CFP* counting process with a hypothetical example of a system consisting of two functional processes.

Our approach requires these measurement data to be saved in the historical database for the past completed projects. For this work, we will need the *CFP* count for each of the functional processes that have been measured, along with the set of textual requirements associated to each on them. Fig. 3 illustrates the steps of building a historical database, when it is not already available.

4.2. Class annotation of functional processes

Once we have prepared the historical database, we need to define classes of functional processes, based on their sizes in *CFP*, to be used later in the automatic classification task. To do this, we performed a box-plot analysis on the *CFP* size values stored in our historical database, to produce four different classes of functional processes, based on their sizes in *CFP*. Table 2 shows the defined ranges of these classes.

Here, the lower quartile would cut off the lowest 25% of all the recorded *CFP* size data from the historical database. The median would divide the dataset by 50%, and the upper quartile cuts off the highest 25% of the dataset.

These four sets of ranges allow us to annotate the textual requirements belonging to each of the functional processes automatically into four fuzzy size classes. In our class ranges, we keep the minimum and the maximum values as 0 and ∞, respectively, instead of the sample minimum or the sample maximum, like in an actual box-plot analysis. Thus, if the new unseen sample is an outlier compared to samples stored in the database, it would still get classified, either as *Small* or as *Complex*.

After annotating the textual requirements automatically into the four classes, we then calculate the median, the minimum and the maximum for each of these classes, to designate the range of the approximate size for each class. Fig. 4 illustrates the process of automatic class annotation described in this section.

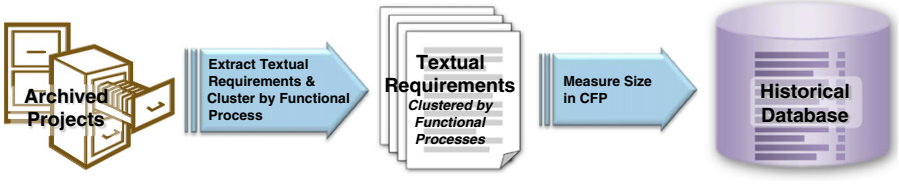


Fig. 3. Building a historical database.

Please cite this article as: I. Hussain, et al., Approximation of COSMIC functional size to support early effort estimation in Agile, Data Knowl. Eng. (2012), doi:10.1016/j.datak.2012.06.005

Table 1
A hypothetical example of precise CFP calculation.

Functional processes	User requirements	Type of *data-movement* expressed by the statement	Number of *data-groups* involved in the *data-movement*	Size in *CFP*
FPr#1	*1.1 User requests to view the detailed information of one item.*	Entry	2	2
		Read	1	1
		Size of statement 1.1 =		3
	1.2 System displays detailed item information.	Exit	1	1
		Size of statement 1.2 =		1
	Total size of FPr#1 =			3 + 1 = 4
FPr#2	*2.1 When user requests to add the item to the shopping cart, system adds it and displays the cart.*	Entry	2	2
		Write	1	1
		Exit	1	1
		Size of statement 2.1 =		4
	Total size of FPr#2 =			4
Total size of the whole system =				4 + 4 = 8

Table 2
Ranges of CFP values to define the classes.

Classes	Ranges
Small	[0, Lower Quartile)
Medium	[Lower Quartile, Median)
Large	[Median, Upper Quartile)
Complex	[Upper Quartile, ~~ℵ~~∞)

4.3. Text mining

Our next step consists of randomly selecting a subset of the annotated textual requirements as our training dataset and extracting linguistic features from the dataset, to train a text classification algorithm that can automatically classify a new set of textual requirements belonging to a functional process into one of the classes defined earlier (i.e. *Small*, *Medium*, *Large* or *Complex*). The classifier will then simply provide the approximate size of each functional process by outputting the median *CFP* value of the class it belongs to, along with the minimum and the maximum *CFP* value seen for that class to indicate possible variation in the approximation. This will provide the quickest possible approximation of the COSMIC functional size from textual requirements that are not formalised and can be written in any quality. Fig. 5 shows the steps of this process.

5. Preliminary study

As a proof of concept, we performed a preliminary experiment with four different case studies: two industrial projects from SAP Labs, Canada, and two university projects. They are all completed projects and are from different domains. Their requirements documents vary in size (from about 2000 words to 11,000 words) and contain from 3 to 32 distinct functional processes, along with detailed descriptions of the problem domains. Table 3 shows some characteristics of these case studies.

We manually pre-processed these requirements to extract sets of requirements sentences, each belonging to a distinct functional process. This mimics the sets of user requirements available before an iteration starts in an agile development process. Thus, from all four requirements documents, we were able to extract 61 sets of textual requirements, each belonging to a distinct functional process.

We used five human measurers, all graduate students in Software Engineering, thoroughly trained for applying the COSMIC standard, to measure the *CFP* of these 61 functional processes, in the same way to what is shown in Table 1. The textual requirements of the 61 functional processes, each tagged with its corresponding *CFP* value, built our historical dataset. The frequency distribution of these *CFP* values in our historical database is shown in Fig. 6. The figure shows that most functional

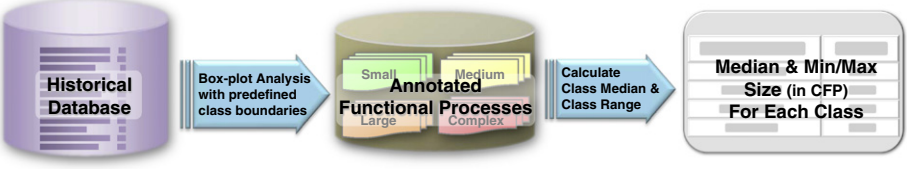


Fig. 4. Class annotation by box-plot analysis.

Please cite this article as: I. Hussain, et al., Approximation of COSMIC functional size to support early effort estimation in Agile, Data Knowl. Eng. (2012), doi:10.1016/j.datak.2012.06.005

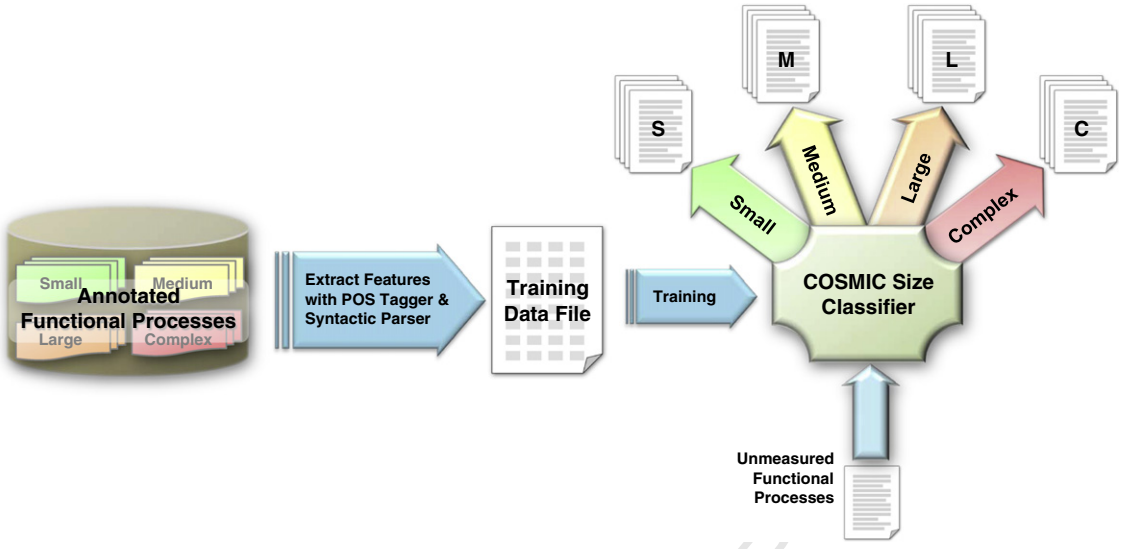


Fig. 5. Text mining for fast approximation of COSMIC functional size.

processes (17 of them) were of size **6** *CFP*. The box-plot on top of the histogram points out the lower quartile, the upper quartile, the sample minimum and the sample maximum, and also indicates that the median size is **6** *CFP* in our historical database.

5.1. The annotated corpus

As mentioned in Section 2, in order to define the ranges of our four size classes, we performed a box-plot analysis on the *CFP* values of our historical database. The resulting boundary points are:

Median 6 *CFP*
Lower Quartile 5 *CFP*
Upper Quartile 8 *CFP*
Sample Minimum 2 *CFP*
Sample Maximum 19 *CFP*

Therefore, according to the ranges defined in Table 2 in Section 3.2, the actual *CFP* ranges for the four size classes for our historical database are:

Small: [0,**5**)
Medium: [**5**,**6**)
Large: [**6**,**8**)
Complex: [**8**,∝)

We then followed these ranges to automatically annotate the sets of textual requirements belonging to the 61 functional processes into the four size classes — where 9 (15%) functional processes were annotated as *Small*, 15 (25%)were *Medium*, 21 (34%) were *Large*, and 16 (26%) were annotated as *Complex*.

Table 3
Summary of the case studies.

ID	Source	Title	Type of application	Size of requirements document	Functional processes extracted
C1	Industry (SAP)	(undisclosed)	Web (Internal)	11,371 words	12
C2	Industry (SAP)	(undisclosed)	Business	1955 words	3
C3	University	Course Registration System	Business	3072 words	14
C4	University	IEEE Montreal Website	Web (Public)	5611 words	32
Total number of functional processes extracted =					61

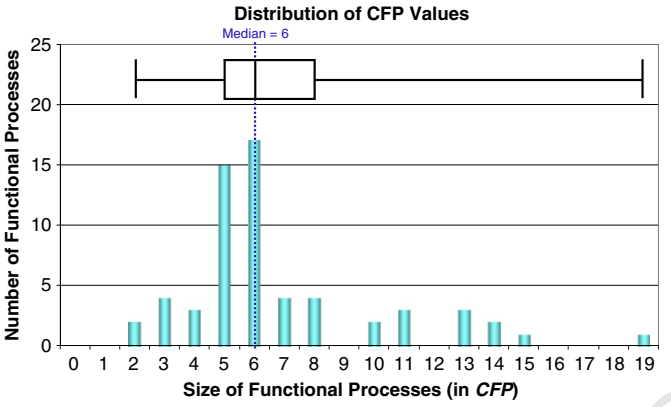


Fig. 6. Distribution (with a box plot) of CFP values in our historical database.

We then collected from our historical database the class data, i.e. the mean, the minimum and the maximum sizes for each of these classes, so that the size of a newly classified functional process belonging to any of these four classes can be approximated by its class data. The resultant class data are shown in Table 4.

It should be noted that due to the small number of functional processes that we currently have collected in our historical database, Table 4 does not show much variation of size among the classes, especially between the classes tMedium and *Large*. This drastically reduces the error margin of our approximation and, therefore, the approximate size, when correctly calculated by the median size of these classes, would will be more precise and introduce much less error. For example, when a functional process will be correctly classified as *Medium* by our text miner, our system will indicate, according to the class data, shown in Table 4, that its approximate (i.e. the median) size is **5** *CFP*, which will in fact be the precise size value of the functional process instead of an approximation. This is because only the functional processes of size **5** *CFP* are set to the *Medium* class by our box-plot analysis. As *CFP* values are always integer numbers, it allows zero margin of error in our approximation of the size of a functional process that belongs to the *Medium* class. Similarly, the error margin of the *Small* and the *Large* classes are also very small. This will also make the task of discriminating between close classes harder than discriminating between widely-varying classes.

5.2. Syntactic features

To perform the classification task, we considered a large pool of linguistic features that can be extracted by a syntactic parser. In this regards, we used the Stanford Parser [23] (equipped with Brill's POS tagger [24] and a morphological stemmer) to morphologically stem the words and extract many linguistic features, e.g. the frequency of words appearing in different parts-of-speech categories. As we have the actual *CFP* values in our historical dataset, we sorted the linguistic features based on their correlation with the *CFP* values. The ten highest correlated features are listed in Table 5.

The correlation shows the ten syntactic features that influence COSMIC functional size the most. The intuitive reasons for them are explained below.

5.2.1. Frequency of noun phrases (#1)

No matter how poorly a requirement is described, the involvement of a data-group in a particular data-movement is typically indicated by the presence of a noun phase. Therefore, if a functional process contains more noun phrases, chances are that its data-movements involve more data-groups and its size is larger.

5.2.2. Frequency of parentheses (#2) and number of tokens inside parentheses (#4)

When complex functional processes are described as textual requirements, parentheses are often used to provide brief explanations in a limited scope, or to include references to additional information that are, otherwise, not included in the description. Thus, a higher number of parentheses or number of tokens inside parentheses can sometimes indicate a complex functional process.

Table 4
Data to be associated with a functional process to approximate its size.

Class	Median size	Minimum size	Maximum size	Approximation error
Small	3 *CFP*	2 *CFP*	4 *CFP*	[−1,1] *CFP*
Medium	5 *CFP*	5 *CFP*	5 *CFP*	0 *CFP*
Large	6 *CFP*	6 *CFP*	7 *CFP*	[0,1] *CFP*
Complex	11 *CFP*	8 *CFP*	19 *CFP*	[-3,8] *CFP*

Please cite this article as: I. Hussain, et al., Approximation of COSMIC functional size to support early effort estimation in Agile, Data Knowl. Eng. (2012), doi:10.1016/j.datak.2012.06.005

Table 5
Ten linguistic features most highly correlated with CFP.

ID	Features (*Frequency of…*)	Correlation with *CFP*
1	Noun phrases	0.4640
2	Parentheses	0.4408
3	Active Verbs	0.4036
4	Tokens in parentheses	0.4001
5	Conjunctions	0.3990
6	Pronouns	0.3697
7	Verb phrases	0.3605
8	Words	0.3595
9	Sentences	0.3586
10	Uniques (hapax legomena)	0.3317

5.2.3. Frequency of active verbs (#3) and verb phrases (#7)

Verbs in active form are frequently used to describe actions and, hence, are often used in larger numbers in textual requirements to explain data-movements, as data-movements result from actions carried out by the user or the system or an external system.

5.2.4. Frequency of pronouns (#6)

A longer description in textual requirements for a functional process often indicates its complexity, and requires the use of more pronouns and other referring expressions within the functional process to maintain cohesion.

5.2.5. Number of words (#8), conjunctions (#5), sentences (#9) and uniques (#10)

In general, lengthy descriptions of the requirements (hence, a higher frequency of words, sentences and unique words) often indicate a more complex functional process.

In addition to the above syntactic features, we also looked at possible keywords that can be used in our classification task.

5.3. Keyword features

Studies (e.g. [25,26]) have shown that using keywords grouped into particular part-of-speech categories can help to obtain good results in various text mining problems, especially for learning the domain-specific terminology. In our case, textual requirements tend to use certain keywords frequently to describe functionality within particular problem domains. We have, therefore, considered lists of keywords as additional features for our work.

Here, each keyword list belongs to a given part-of-speech category to isolate some senses to the keywords. For example, this process would differentiate between the word “open” as a verb (that designates the action to open) from the word “open” as an adjective (that indicates the state of something that is open). For this work, we chose three open-class part-of-speech groups for these keywords to be selected. They are: Noun-keywords (coded as: *NN_keyword*), Verb-keywords (coded as: *VB_keyword*), and Adjective-keywords (coded as: *JJ_keyword*).

We generate finite lists of these keywords based on two different probabilistic measures, as described in ref. [25], that take into account how many more times the keywords occur in one class of the training set than the other class. A cutoff threshold is then used to reduce the list to keep only the top most discriminating words. For example, the three lists that were automatically generated by this process from our training set during a single fold of 10-fold-cross-validation are shown in Table 6.

These three lists constituted three additional features for our classification task. Thus, when we extract the features, we counted one of the keyword feature, for example, as how many times words from its keyword-list appears in the set of requirements of a functional process, and appearing in the same part-of-speech class.

Table 6
Some of the keywords of POS group: noun, verb and adjective.

NN_keyword	VB_keyword	JJ_keyword
user	ensure	supplied
category	get	current
quota	choose	previous
content	start	available
default	restart	
chart	fill	
…	…	

Please cite this article as: I. Hussain, et al., Approximation of COSMIC functional size to support early effort estimation in Agile, Data Knowl. Eng. (2012), doi:10.1016/j.datak.2012.06.005

Table 7
Interpretation of the values of Kappa (κ) [33].

Kappa (κ) value	Strength of agreement beyond chance
<0.00	Poor
0.01–0.20	Slight
0.21–0.40	Fair
0.41–0.60	Moderate
0.61–0.80	Substantial
0.81–1.00	Almost perfect

5.4. Feature extraction and classification

To classify the sets of textual requirements belonging to different functional processes, we developed a Java-based text classifier program that uses the Stanford Parser [23] that extracts the values of all the syntactic and keyword features mentioned above. It takes 4.68 seconds on average (running on a dual-core CPU with 64-bit JVM) to extract all the selected features from a functional process that contains about 5 sentences on average. The classifier then uses the publicly available Weka package [27] to train and test the C4.5 decision tree learning algorithm [28]. We used the implementation of the C4.5 (revision 8) that comes with Weka (as J48), setting its parameter for the minimum number of instances allowed in a leaf to 6 to counter possible chances of over-fitting. The results are discussed in the next section. We also trained/tested with a Naïve Bayes classifier [29], and a logistic classifier [30]. The C4.5 decision tree-based classifier performed the best in comparison to the other classifiers with more consistent results during 10-fold-cross-validation.

6. Results and analysis

In this article, we evaluated performance mostly in terms of the degree of agreement, measured by the Kappa statistic [31], between the actual classes and the classes predicted by our classifier for all the test instances. The Kappa index, denoted by κ, refers to the following ratio:

$$\kappa = \frac{P(A) - P(E)}{1 - P(E)} \quad (1)$$

Here, $P(A)$ is the proportion of total times that the predicted classes are observed to agree with the actual classes, and $P(E)$ is the proportion of the total times that the predicted classes are expected to agree with the actual classes. The interpretation of different values of the κ index varies with applications in different fields of study [32]. One most commonly used interpretation put forth by Landis and Koch [33] is shown in Table 7.

The results attained by our classifier were moderate when using the whole dataset for training and testing. Since the dataset was not very large, we could not use a separate dataset for testing, and we could only use cross-validation, which can be very harsh on the performance, when the number of instances is very low. Yet, the classifier results did not drop significantly. Table 8 shows a summary of the results.

The resultant decision tree after training on the complete dataset is shown in Fig. 7. As the figure shows, the tree came out well-formed and of desirable characteristics — not sparse, and also not flat. Also, none of its branches are wrongly directed.

Although the kappa results of Table 8 shows stable and moderate results in terms of performance with the 10-fold-cross-validation, the confusion matrix of Table 9 shows that the classifier struggled to classify functional processes of size *Medium* into the correct class; classifying only 47% of them (7 out of 15) correctly. We can also see that the mistakes the classifier made with the *Medium* sized functional processes are mostly because it confused them as *Large* (shown in darker shade, in Table 8, it classified another 7 out of the 15 *Medium* functional processes incorrectly into the size class *Large*). The reason for this can be understood by the fact discussed in Section 1, where, in Table 4, we see that our box-plot analysis automatically chose zero approximation error for the class *Medium*. It, therefore, became the hardest class to classify among the other classes, carrying very minute differences from its adjacent class *Large*, which also has a smaller margin of approximation error. Thus, when our classifier correctly classifies a functional process as xtitMedium, it does not really approximate its size; rather it accurately identifies its precise size value, which is **5** *CFP*. Again, when the classifier mistakenly classified a *Medium* functional process as *Large*, the error

Table 8
Summary of the results.

	Scheme	Correctly classified instances	Incorrectly classified instances	Kappa
Corpus size = 61 (sets of textual requirements, each set representing a functional process)	Training + Testing on same set	45 (73.77%)	16 (26.23%)	0.6414
	Cross-validation (10 Folds)	41 (67.21%)	20 (32.79%)	0.5485

Please cite this article as: I. Hussain, et al., Approximation of COSMIC functional size to support early effort estimation in Agile, Data Knowl. Eng. (2012), doi:10.1016/j.datak.2012.06.005

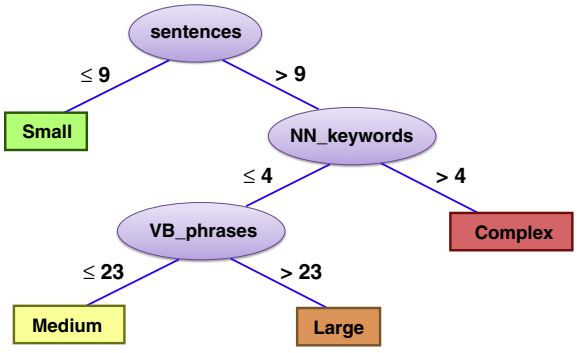


Fig. 7. The resultant C4.5 decision tree after training with the complete dataset.

Table 9
Confusion matrix when using 10-fold-cross-validation.

	Classified as			
	Small	Medium	Large	Complex
Small	7	0	1	1
Medium	1	7	7	0
Large	2	1	16	2
Complex	2	0	3	11

in size approximation that it made is of only **1** *CFP*. If we had a larger number of instances, there would most likely have been a wider variation of size values in our historical database. We believe that this would make the classification task easier for our classifier allowing the learning algorithm to find the threshold values for the other unused linguistic features and, thus, utilise them in making fine-grained distinction and render better results.

By analysing Table 9, we also find that the classifier had difficulty in identifying the functional processes of size *Small*. Although it classified 7 out of 9 *Small* functional processes correctly as *Small*, it misclassified some *Medium*, *Large*, and even *Complex* functional processes as *Small* (see the 1st column of Table 9). Here, again, we believe that the small size of our dataset (e.g. we had only 9 instances of size *Small*) may be the cause. It should be noted that these results were extracted during cross-validation of 10 random folds, which can significantly reduce the number of training instances for a particular class during a single fold in a skewed corpus. In our case, for example, during one fold, the number of training instances for the *Small* class went minimum of only 2 instances, which were inadequate for the learning algorithm to discover the thresholds of most of the discriminating linguistic features that we selected for this work.

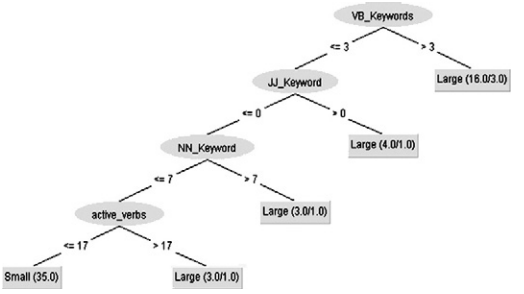
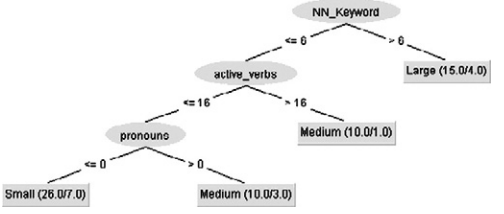
The phenomena discussed above are also reflected in the precision and recall results shown in Table 10. Moreover, Table 10 also shows that a good performance on average attained by the classifier with such a small dataset. It should be mentioned that, in our previous work [25], we showed the applicability of using similar a approach for requirements classification, where we had a significantly large dataset (765 instances) to classify into only two classes (Functional and Non-functional requirements) and the classifier attained a much higher level of accuracy (0.98 for precision, and 1.0 for recall, during 10-fold-cross-validation).

Thus, although we believe that the results presented in this article would improve with the introduction of more instances in our dataset, the absence of a large dataset does not allow us to scientifically prove this claim. However, we can demonstrate what happens if we increase the number of instances per class in our current dataset by reducing the total number of our target classes. In this article, we explained our methodology for building a four-class classifier (classifying functional processes into four distinct size classes: *Small*, *Medium*, *Large* and *Complex*). This drastically reduces the number of our training instances by dividing them into four different sets. So, if we had less number of classes, i.e. two or three size classes, instead of four, the available number of instances per class in our current dataset would have been higher to perform a more realistic classification task.

Table 10
Precision, Recall and F-Measure, when using 10-fold-cross-validation.

Size class	Precision	Recall	*F*-Measure
Small	0.583	0.778	0.667
Medium	0.875	0.467	0.609
Large	0.593	0.762	0.667
Complex	0.786	0.688	0.733
Mean	0.709	0.673	0.669

t11.1 **Table 11**
10-fold cross validation results of using 2-class and 3-class classifiers.

Classifier	Kappa	Size Classes	Precision	Recall	Tree
2-class Classifier	0.606	Small	0.895	0.829	
		Large	0.696	0.8	
3-class Classifier	0.575	Small	0.677	0.875	
		Medium	0.794	0.721	
		Large	0.9	0.73	

To show what would happen if we had increased the number of training instances per class, we developed a two-class size classifier (classifying functional processes into *Small* and *Large* classes), and a three-class size classifier (classifying functional processes into *Small*, *Medium* and *Large* classes). Thus, the number of instances per class in our dataset increased, compared to how we originally had it for our four-class classifier. We used the same dataset, the same methodology and the same sets of features described in this article while building these classifiers. The results were significantly better, attaining mean f-measures of 0.802 and 0.746 for the 2-class and the 3-class classifiers respectively during 10-fold-cross-validation. The summary of the results of performing 10-fold-cross-validation using both the classifiers is shown in Table 11.

The results in Table 11 shows that a similar classification technique when applied on the same dataset, which now contains more training instances per class than what we had for our four-class classifier, improves the results significantly. This allows us to conclude that the results of our original four-class classifier would also improve, in case we had more training instances per class.

7. Conclusions and Future Work

In this article, we have shown that classification of textual requirements in terms of their functional size is plausible using linguistic features. Since our work uses a supervised text mining approach, where we need experts to build our historical database by manually measuring the COSMIC functional size from textual requirements, we could not train and test our system with a large number of samples (only 61 in total). Yet, the results that we were able to gather by cross-validating on such small number of samples show a promising behavior of the classifier in terms of its performance. Using our methodology, we have also been able to identify automatically a set of highly discriminating features that can effectively help together with a classifier in approximating the size of functional processes.

It should be mentioned that we have not yet tested this approach as to be used with requirements written in variable level of quality. Therefore, we believe that this approach would be organisation-specific, where textual requirements saved in the historical dataset should all be written in the same format or writing style having similar quality. This would allow our classifier to pick the best set of features and set the best thresholds that would classify new requirements written in similar style and quality.

We are currently in the process of building larger datasets for training and testing our system. Our future work includes implementing a full-fledged prototype to demonstrate its use and a complete integration to the READ-COSMIC project [34], which is our umbrella project on software development effort estimation from textual requirements. We are also working on predicting the impact of non-functional requirements on the functional size for better precision in software effort estimation.

Acknowledgements

The authors would like to thank SAP Labs, Canada for providing the requirements documents used in the experiments presented in this article, and the anonymous reviewers for their valuable comments on its earlier version.

Please cite this article as: I. Hussain, et al., Approximation of COSMIC functional size to support early effort estimation in Agile, Data Knowl. Eng. (2012), doi:10.1016/j.datak.2012.06.005

References

[1] C. Larman, Agile & Iterative Development: A Manager's Guide, Pearson Education, Boston, MA, 2003.
[2] R.C. Martin, Agile Software Development: Principles, Patterns and Practices, Prentice Hall, Upper Saddle River, NJ, 2003.
[3] M. Cohn, Agile Estimating and Planning, Prentice Hall, Upper Saddle River, NJ, 2005.
[4] S.L. Pfleeger, F. Wu, R. Lewis, Software Cost Estimation and Sizing Methods, Issues and Guidelines, In: Technical Report, RAND Corporation, 2005.
[5] A.J. Albrecht, Measuring application development productivity, In: IBM Application Development Symposium, Press IBM, 1979, pp. 83–92.
[6] A.J. Albrecht, J.E. Gaffney Jr., Software function, source lines of code, and development effort prediction: a software science validation, IEEE Transactions on Software Engineering 9 (1983) 639–648.
[7] B.A. Kitchenham, N.R. Taylor, Software Cost Models, ICL Technical Journal 4 (1984) 73–102.
[8] ISO, Software engineering — IFPUG 4.1 unadjusted functional size measurement method — counting practices manual, 2003.
[9] ISO, Software engineering — Mk II function point analysis — counting practices manual. International Organization for Standardization, 2002.
[10] ISO, Software engineering — NESMA functional size measurement method version 2.1 — definitions and counting guidelines for the application of function points analysis, 2005.
[11] ISO, COSMIC Full Function Points Measurement Manual v.2.2, International Organization for Standardization, 2003.
[12] N. Condori-Fernández, S. Abrahão, O. Pastor, On the estimation of the functional size of software from requirements specifications, Journal of Computer Science and Technology 22 (2007) 358–370.
[13] H. Diab, F. Koukane, M. Frappier, R. St-Denis, μcROSE: automated Measurement of COSMIC-FFP for rational rose real time, Information and Software Technology 43 (2005) 151–166.
[14] H.M. Sneed, Extraction of Function Points from Source-Code, In: Proceedings of New Approaches in Software Measurement, 10th International Workshop, IWSM, Springer-Verlag, Berlin, Germany, 2001, pp. 135–146.
[15] M. Shepperd, M. Cartwright, Predicting with sparse data, IEEE Transactions on Software Engineering 27 (2001) 987–998.
[16] R. Meli, Early and Extended Function Point: a new method for Function Points estimation, In: Proceedings of the IFPUG Fall Conference, IFPUG, Scottsdale, Arizona, 1997.
[17] L. Santillo, M. Conte, R. Meli, Early & Quick Function Point: Sizing More with Less, In: 11th IEEE International Software Metrics Symposium (METRICS'05), IEEE Press, Como, Italy, 2005, p. 41.
[18] ISO, Functional Size Measurement—Definition of Concepts. International Organization for Standardization, 1998.
[19] C. Gencel, O. Demirors, E. Yuceer, A Case Study on Using Functional Size Measurement Methods for Real Time Systems, In: Proceedings of the 15th. International Workshop on Software Measurement (IWSM), Shaker-Verlag, Montreal, Canada, 2005, pp. 159–178.
[20] J.L. Cooke, Agile Productivity Unleashed: Proven Approaches for Achieving Real Productivity Gains in Any Organization, It Governance Ltd., 2010.
[21] S. Hoogendoorn, Smart use cases, http://www.smartusecase.com/(X(1)S(hp3vxp242ym1mg45faqtegbg))/Default.aspx?Page=SmartUseCase, 2009, (Retrieved on August 29, 2010).
[22] B. Marín, O. Pastor, G. Giachetti, Automating the Measurement of Functional Size of Conceptual Models in an MDA Environment, In: Proceedings of the 9th international conference on Product-Focused Software Process Improvement, Springer-Verlag, Berlin, Heidelberg, 2008, pp. 215–229.
[23] D. Klein, C.D. Manning, Accurate Unlexicalized Parsing, In: ACL '03: Proceedings of the 41st Annual Meeting on Association for Computational Linguistics, Association for Computational Linguistics, Morristown, NJ, USA, 2003, pp. 423–430.
[24] E. Brill, A Simple Rule-Based Part of Speech Tagger, In: Proceedings of the third conference on Applied Natural Language Processing (ANLP), Association for Computational Linguistics, 1992, pp. 152–155.
[25] I. Hussain, L. Kosseim, O. Ormandjieva, Using Linguistic Knowledge to Classify Non-functional Requirements in SRS Documents, In: NLDB '08: Proceedings of the 13th International Conference on Natural Language and Information Systems, Springer-Verlag, Berlin, Heidelberg, 2008, pp. 287–298.
[26] J. Wiebe, T. Wilson, R. Bruce, M. Bell, M. Martin, Learning subjective language, Computational Linguistics 30 (2004) 277–308.
[27] I.H. Witten, E. Frank, Data Mining: Practical Machine Learning Tools and Techniques, 2nd edition Morgan Kaufmann, San Francisco, CA, 2005.
[28] J.R. Quinlan, C4.5: Programs for Machine Learning (Morgan Kaufmann Series in Machine Learning), 1 edition Morgan Kaufmann, 1993.
[29] G.H. John, P. Langley, Estimating Continuous Distributions in Bayesian Classifiers, In: Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence, Morgan Kaufmann, Montreal, Quebec, 1995, pp. 338–345.
[30] S. le Cessie, J.C. van Houwelingen, Ridge Estimators in Logistic Regression, Applied Statistics 41 (1992) 191–201.
[31] J. Cohen, A coefficient of agreement for nominal scales, Educational and Psychological Measurement 20 (1960) 37–46.
[32] J. Carletta, Assessing agreement on classification tasks: the kappa statistic, Computational Linguistics 22 (1996) 249–254.
[33] J. Landis, G. Koch, The measurement of observer agreement for categorical data, Biometrics 33 (1977) 159–174.
[34] I. Hussain, O. Ormandjieva, L. Kosseim, Mining and Clustering Textual Requirements to Measure Functional Size of Software with COSMIC, In: Proceedings of the International Conference on Software Engineering Research and Practice (SERP 2009), CSREA Press, 2009, pp. 599–605.

Ishrar Hussain received his Master' degree in Computer Science in 2007 from Concordia University in Montreal, Canada. He then joined the Ph.D. in Computer Science program at the same university, and is now a Ph.D. Candidate expecting to graduate in 2013. His thesis is jointly supervised by Dr. Leila Kosseim and Dr. Olga Ormandjieva, and is about applying natural language processing to improve functional size measurement from software requirements.
Ishrar has been continuing his research in the fields of Natural Language Processing, Machine Learning and Requirements Engineering for the last seven years. He is proficient in different platforms for linguistic analysis, e.g. GATE, UIMA, Stanford NLP etc., and has developed several standalone applications. Ishrar is also the author of seven research papers.

Dr. Leila Kosseim is an Associate Professor in the Department of Computer Science and Software Engineering at Concordia University in Montreal, Canada. She holds a PhD. in Computer Science (1995) from the Université de Montréal in 1995.After her Ph.D., Leila received an NSERC industrial post-doctorate fellowship, then joined the RALI laboratory at the Université de Montréal as a researcher before joining Concordia University as a faculty member in 2001.
Leila's research interests are in Natural Language Processing (NLP), specifically: Question Answering, Natural Language Generation, Text Summarization and Word Sense Disambiguation. She also has a strong interest for machine learning, data mining, information retrieval and logic programming. Leila is the author of over 50 papers in NLP.

Please cite this article as: I. Hussain, et al., Approximation of COSMIC functional size to support early effort estimation in Agile, Data Knowl. Eng. (2012), doi:10.1016/j.datak.2012.06.005

Dr. Olga Ormandjieva is an Associate Professor in the Computer Science and Software Engineering Department at Concordia University, Montreal, Canada and a member of the Ordre des Ingénieurs du Québec (OIQ). She holds a Ph.D. in Computer Science (2002) and Master's Degree in Computer Science and Mathematics (1987).
Olga joined the Department in 2002 as an Assistant Professor. She has conducted extensive research in the area of Measurement in Software Engineering and its extension to the development of formal methods for modelling and monitoring functional and non-functional requirements with category theory. She has over 80 journal and conference papers to her name, as well as two book chapters. Olga has supervised over 20 graduate students and is held in very high esteem by all those she teaches. Currently she is teaching software engineering undergraduate and graduate courses at Concordia University.

Please cite this article as: I. Hussain, et al., Approximation of COSMIC functional size to support early effort estimation in Agile, Data Knowl. Eng. (2012), doi:10.1016/j.datak.2012.06.005