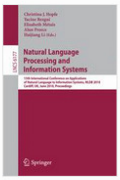

International Conference on Application of Natural Language to Information Systems

NLDB 2010: Natural Language Processing and Information Systems pp 80-91 | Cite as

Towards Approximating COSMIC Functional Size from User Requirements in Agile Development Processes Using Text Mining

Authors | Authors and affiliations

Ishrar Hussain, Leila Kosseim, Olga Ormandjieva

Conference paper

Part of the Lecture Notes in Computer Science book series (LNCS, volume 6177)

Abstract

Towards Approximating COSMIC Functional Size from User Requirements in Agile Development Processes Using Text Mining

Ishrar Hussain, Leila Kosseim and Olga Ormandjieva

Department of Computer Science and Software Engineering,
Concordia University, Montreal, Quebec, Canada
{h_hussa, kosseim, ormandj}@cse.concordia.ca

Abstract. Measurement of software size from user requirements is crucial for the estimation of the developmental time and effort. COSMIC, an ISO/IEC international standard for functional size measurement, provides an objective method of measuring the functional size of the software from user requirements. COSMIC requires the user requirements to be written at a level of granularity, where interactions between the internal and the external environments to the system are visible to the human measurer, in a form similar to use case descriptions. On the other hand, requirements during an agile software development iteration are written in a less formal way than use case descriptions — often in the form of user stories, for example, keeping with the goal of delivering a planned release as quickly as possible. Therefore, size measurement in agile processes uses methods (e.g. story-points, smart estimation) that strictly depend on the subjective judgment of the experts, and avoid using objective measurement methods like COSMIC. In this paper, we presented an innovative concept showing that using a supervised text mining approach, COSMIC functional size can be automatically approximated from informally written textual requirements, demonstrating its applicability in popular agile software development processes, such as Scrum.

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. It focuses on the rapid production of functioning software components along with providing the flexibility to adapt to emerging business realities [19]. In practice, agile processes have been extended to offer more techniques, e.g. describing the requirements with user stories [21]. Instead of a manager estimating developmental time and effort and assigning tasks based on conjecture, team members in agile approach use effort and degree of difficulty in terms of points, or size, to estimate of their own work, often with biased judgment [5]. Hence, an objective measurement of software size is crucial in planning and management of agile projects.

We know that effort is a function of size [22], 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.

The above has led the industry to formulate several methods for functional size measurement (FSM) of software. Allan Albrecht first proposed the idea in his work on function point analysis (FPA) in 1979 [2], where he named the unit of functional size as function point (FP). His idea of effort estimation was then validated by many studies, like [3,17], and, thus, measuring the functional size of the software became an integral part of effort estimation. There have been many standards developed by different organizations 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 [14], Mark II [15], NESMA [16] and COSMIC [13].

There have been many studies in recent years [6,7,26], where researchers attempted to automate the process of different functional size measurement methods, but none of them, to our knowledge, addressed the problem where they would take 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 formalization of the requirements. This is in response to the high industrial demands of performing size estimation during agile development processes, where formalization 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 [25] and the Easy and Quick (E&Q) measurement approach, that was originated in the IFPUG standard [14]. The applicability of this approach in COSMIC was manually demonstrated by [24].

2. Background

2.1 COSMIC

For the purposes 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 [13]. We chose this method in particular, because it conforms to all ISO requirements [12] for FSM, focuses on the "user view" of functional requirements, and is applicable throughout the Agile development life cycle. Its potential of being applied accurately in the requirements specification phase compared to the other FSM methods is demonstrated by the study of [8]. Also, COSMIC does not rely on subjective decisions by the functional size measurer during the measurement process [13]. Thus, its measurements, taken from well-specified requirements, tend to be same among multiple measures. This is particularly important for validating the performance of the automatic size measurements that would be yielded by our solution.

In COSMIC, the 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.

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 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 sums up sizes of all the functional processes to compute the total size of the system to be measured.

2.2 Size Measurement in Agile Development Processes

Agile development processes are driven by the motto of delivering releases as quickly as possible [19]. 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 [1] or user-stories [21], 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 [21]. Size measurement methods in agile development processes include story-points [5] and smart estimation [1], and depend on the subjective judgment of human experts, and, therefore, are prone to biases and errors [5].

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, the IT industry recognized the traditional processes to cause many problems including delays and is now increasingly moving towards agile development processes, such as Scrum [29], an agile approach that does not impose documentation templates or formalisms on requirements. Lack of formalism in requirements restricts FSM methods, like COSMIC, to be applied for measuring the functional size of an iteration in an agile development process. For example, from the discussion in section 2.1, 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 formalizing the requirements with conceptual model (e.g. a domain model).

Our work presents an alternative solution, which does not require the use of domain models; 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* [7], where the authors developed a comprehensive system called, µcROSE, which accepts state charts as inputs to measure the functional size of real-time systems only. We find their work to be largely dependant 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. They presented a very brief validation of their work by an expert, testing their system against only one case study, where it performed poorly in detecting the data groups, resulting in some erroneous measurement outputs.

Condori-Fernández *et al* [6] performed another study, where they 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.

4. Methodology

Most of the related work performed in this field were directed towards performing a precise measurement of COSMIC functional size. On the other hand, our goal is to develop an automated tool that would do quicker estimation of COSMIC size without requiring the formalization of the requirements. Our methodology requires the historical data of an organization to be stored for the purpose of generating a dataset for training/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

Our first step is to build our historical dataset by manually measuring the COSMIC size of the functional processes in units of CFP (COSMIC Function Point). 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 sums up CFP sizes for each of the functional processes to obtain the re-

spective CFP count of the whole system. Table 1 illustrates the CFP counting process with a hypothetical example of a system consisting of two functional processes.

Table 1. A hypothetical example of precise CFP calculation

Functional processes	User requirements	Types of Data-movements expressed by the statement:	Number of Data-groups involved in a 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**

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 textual requirements associated to a functional process.

4.2 Class Annotation of Functional Processes

Once we have gathered the historical dataset, 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 from our historical dataset, to produce four different classes of functional processes, based on their sizes in CFP. Table 2 shows the defined ranges of these classes.

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, ∞)

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 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; so that, if the new unseen sample is an outlier compared to the historical dataset, it would still get classified into a class.

After defining the class boundaries automatically, we then calculate the mean, the minimum and the maximum for each of the classes, to designate the range of the approximate size for each of the classes.

4.3 Text Mining

Our next step consists of extracting linguistic features from the textual requirements belonging to each of the functional processes from our training 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 above (i.e. *Small*, *Medium*, *Large* or *Complex*). It will then simply approximate the size of the functional processes by outputting its size as the calculated mean value of the class it belongs to, along with the minimum and the maximum seen CFP value for that class to indicate possible variation in the approximation; and, thus, provide the quickest possible approximation of the COSMIC functional size from textual requirements that are not formalized and can be written in any quality.

5. Preliminary Study

As a proof of concept, we performed a small preliminary study with four different case studies: two industrial projects from SAP Labs, Canada, and two university projects. They are all completed projects and from different domains. Table 3 summarizes the case studies.

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	1,955 words	3
C3	University	Course Registration System	Business	3,072 words	14
C4	University	IEEE Montreal Website	Web (Public)	5,611 words	32
Total number of functional processes extracted =					**61**

We manually pre-processed these requirements to extract sets of requirements sentences each of which belongs to a distinct functional process. This mimics the available set of user requirements before an iteration starts in an agile development process. 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 skilled to perform COSMIC FSM from requirements documents, to measure the CFP of these 61 functional

processes, similarly to what is shown in Table 1. The CFP values and the textual requirements of the 61 functional processes built our historical dataset.

5.1 The Annotated Corpus

As mentioned in Section 3.2, we performed a box-plot analysis on the CFP values of our historical dataset that gave us — median = 6 CFP; lower-quartile = 5 CFP; and, upper-quartile = 8 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 dataset are: *Small: [0,5)*; *Medium: [5,6)*; *Large: [6,8)*; and, *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*.

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 [18] (equipped with Brill's POS tagger [4] 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 the correlation between their values and the CFP values. The top ten highly correlated features are listed in Table 4.

Table 4. Ten linguistic features highly correlated with CFP

Features (*Frequency of..*)	Correlation with CFP
Noun Phrases	0.4640
Parentheses	0.4408
Active Verbs	0.4036
Tokens in Parentheses	0.4001
Conjunctions	0.3990
Pronouns	0.3697
Verb Phrases	0.3605
Words	0.3595
Sentences	0.3586
Uniques (*hapax legomena*)	0.3317

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

Noun Phrases. 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, the chances are that its data-movements involve more data-groups and its size is larger.

Parentheses & Number of tokens inside parentheses. When complex functional processes are often described in textual requirements, parentheses are used to provide brief explanations in a limited scope. Thus, a higher number of parentheses/Number of tokens inside parentheses can sometimes indicate a complex functional process.

Active Verbs & Verb Phrases. Verbs in active form define actions and 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.

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

Words, Sentences and Uniques. They all account for lengthy description of the requirements in functional processes; and, as mentioned above, long description of requirements of a functional process may indicate its complexity.

Next, we looked at possible keyword features than can be extracted.

5.3. Keyword Features

A large body of work (e.g. [10,27]) has shown that using keywords grouped into particular parts-of-speech categories helps to obtain good results with Text Mining. For this study, we have, therefore, considered list of keywords, each list belonging to a given parts-of-speech category. We chose three part-of-speech groups for these keywords to be selected. They are: Noun-keywords (coded as: *NN_keyword*), Verb-keywords (coded as: *Verb_keyword*), and Adjective-keywords (coded as: *JJ_keyword*).

We generate finite lists of these keywords based on two different probabilistic measures, as described in [10], that takes 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 keep the list significantly smaller. For example, the three lists that were automatically generated by this process from our training set during a fold of 10-fold-crossvalidation is shown in Table 5.

Table 5. 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	
...	...	

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.

5.4. Feature Extraction and Classification

To classify the sets of textual requirements belonging to different functional processes, we developed a Java-based feature extraction program that uses the Stanford Parser [18] to extract the values of all the syntactic and keyword features mentioned above, and uses Weka [28] to train and test the C4.5 decision tree learning algorithm [23]. 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 [9], and a logistic classifier [20]. 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

The results of the classification were very 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 6 shows a summary of the results.

Table 6. Summary of the results

	Scheme	Correctly Classified Sentences	Incorrectly Classified Sentences	Kappa	Comment
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	Tree is of desirable characteristics, not sparse, and also not flat. None of the branches are wrongly directed.
	Cross-validation (10 Folds)	41 (67.21%)	20 (32.79%)	0.5485	

The resultant decision tree after training on the complete dataset also came out well-formed. The tree is shown in Figure 1.

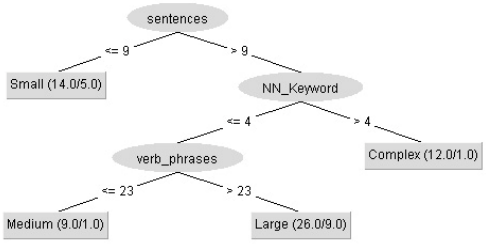

Fig. 1. The resultant C4.5 decision tree after training with the complete dataset.

Detailed results of using 10-fold-crossvalidation with the final confusion matrix is shown in Table 7.

Table 7. Confusion matrix when using 10-fold-crossvalidation

	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

The precision, recall and f-measure results for each of the classes in 10-fold-cross-validation are shown in Table 8.

Table 8. Precision, Recall and F-Measure, when using 10-fold-crossvalidation

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

Although the kappa results of Table 6 shows stable and moderate results in terms of performance with the 10-fold-crossvalidation, the results of the confusion matrix in Table 7 and the results in Table 8 show that the classifier is struggling to attain a good recall with the fuzzy class *Medium*. The reason for this is that classifying an instance into the size classes that fall in the middle (e.g. functional processes that are not small and not large, but of medium size) is hard when we do not have a larger number of instances that can allow the learning algorithm to find the threshold values for the other features and thus, utilize them in making fine-grained distinction.

We can also demonstrate by showing that if we had less number of classes, i.e. two or three size classes, the available number of instances would have been enough for a more realistic classification task. To show that, we developed both 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) using the same principles and the same sets of features described earlier in this paper. 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.

7. Conclusions and Future Work

In this paper, we have shown that classification of textual requirements in terms of their functional size is plausible. Since our work uses a supervised text mining approach, where we needed 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. Yet, the results that we were able to gather by crossvalidating on such small number of samples show a promising behavior of the classifier in terms of its performance. We have been able to identify automatically a set of highly discriminating features that can effectively help together with a classifier.

It should be mentioned that we have not yet tested our approach as to be used with requirements written in variable level of quality. Therefore, we believe that this approach would be organization-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 [11], 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 the anonymous reviewers for their valuable comments on an earlier version of the paper.

References

1. Accelerated Delivery Platform. (2009). Smart use cases. Retrieved on February 14, 2009 from http://www.smartusecase.com/(X(1)S(hp3vxp242ym1mg45faqtegbg))/Default.aspx?Page=SmartUseCase
2. Albrecht, A. J. (1979). Measuring Application Development Productivity. Proceedings of IBM Application Development Symp. (pp. 83-92). Monterey, Calif.: Press I.B.M.
3. Albrecht, A. J., & Gaffney, J. E. (1983). Software function, source lines of code, and development effort prediction: A software science validation. IEEE Transactions on Software Engineering, 9, 639-648.
4. Brill, E. (1992). A Simple Rule-Based Part of Speech Tagger. Proceedings of the third conference on Applied natural language processing (pp. 152-155). Trento, Italy: Association for Computational Linguistics.
5. Cohn, M. (2005). Agile Estimating and Planning. Upper Saddle River, NJ: Prentice Hall.
6. Condori-Fernández, N., Abrahão, S., & Pastor, O. (2007). On the estimation of the functional size of software from requirements specifications. Journal of Computer Science and Technology, 22 (3), 358-370.
7. Diab, H., Koukane, F., Frappier, M., & St-Denis, R. (2005). µcROSE: Automated Measurement of COSMIC-FFP for Rational Rose Real Time. Information and Software Technology, 47 (3), 151-166.
8. Gencel, C., Demirors, O., & Yuceer, E. (2005). Utilizing Functional Size Measurement Methods for Real Time Software System. 11th IEEE International Software Metrics Symposium (METRICS 2005). IEEE Press.

9. George H. John & Pat Langley. (1995). Estimating Continuous Distributions in Bayesian Classifiers. Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence, San Mateo, 338-345.
10. Hussain, I., Kosseim, L., & Ormandjieva, O. (2008). Using Linguistic Knowledge to Classify Non-functional Requirements in SRS documents. LNCS: Natural Language and Information Systems (Vol. 5039/2008), pp. 287-298. Germany: Springer-Verlag.
11. Hussain, I., Ormandjieva, O., & Kosseim, L. (2009). Mining and Clustering Textual Requirements to Measure Functional Size of Software with COSMIC. Proceedings of the International Conference on Software Engineering Research and Practice (SERP 2009), pp. 599-605, CSREA Press.
12. ISO/IEC 14143-1. (1998). Functional Size Measurement - Definition of Concepts. International Organization for Standardization.
13. ISO/IEC 19761. (2003). COSMIC Full Function Points Measurement Manual v.2.2. International Organization for Standardization.
14. ISO/IEC 20926. (2003). Software Engineering -- IFPUG 4.1 Unadjusted functional size measurement method -- Counting Practices Manual. International Organization for Standardization.
15. ISO/IEC 20968. (2002). Software Engineering - Mk II Function Point Analysis - Counting Practices Manual. International Organization for Standardization.
16. ISO/IEC 24570. (2005). Software Engineering -- NESMA functional size measurement method version 2.1 -- Definitions and counting guidelines for the application of Function Points Analysis. International Organization for Standardization.
17. Kitchenham, B. A., & Taylor, N. R. (1984). Software cost models. ICL Technical Journal, 4, 73-102.
18. Klein, D., & Manning, C. D. (2003). Accurate unlexicalized parsing. Proceedings of the 41st Meeting of the Association for Computational Linguistics (pp. 423-430). Association for Computational Linguistics.
19. Larman, C. (2003). Agile & Iterative Development: a Manager's Guide. Boston, MA: Pearson Education.
20. le Cessie, S. & van Houwelingen, J.C. (1992). Ridge Estimators in Logistic Regression. Applied Statistics. 41(1):191-201.
21. Martin, R. C. (2003). Agile Software Development: Principles, Patterns and Practices. Upper Saddle River, NJ: Prentice Hall.
22. Pfleeger, S. L., Wu, F., & Lewis, R. (2005). Software Cost Estimation and Sizing Methods. Issues and Guidelines. RAND Corporation.
23. Quinlan, J. R. (1993). C4.5: Programs for machine learning. San Mateo, CA: Morgan Kaufmann.
24. Santillo, L., Conte, M., & Meli, R. (2005). E&Q: An Early & Quick Approach to Functional Size. IEEE International Symposium on Software Metrics (p. 41). Los Alamitos, CA, USA: IEEE Computer Society.
25. Shepperd, M., & Cartwright, M. (2001). Predicting with sparse data. IEEE Transactions on Software Engineering, 27, 987-998.
26. Sneed, H. M. (2001). Extraction of function points from source-code. Proceedings of New Approaches in Software Measurement, 10th International Workshop, IWSM (pp. 135-146). Berlin, Germany: Springer-Verlag.
27. Wiebe, J., Wilson, T., Bruce, R., Bell, M., & Martin, M. (2004). Learning Subjective Language. Computational Linguistics, v.30 n.3 (September 2004), Cambridge, MA, USA: MIT Press, 277–308.
28. Witten, I. H., & Frank, E. (2005). Data mining: Practical machine learning tools and techniques (2nd ed.). San Francisco, CA: Morgan Kaufman.