ESSE: An Early Software Size Estimation Method Based on Auto-extracted Requirements Features

Cheng Zhang[1], Shensi Tong[1], Wenkai Mo[1], Yang Zhou[1], Yong Xia[2], Beijun Shen[1]
[1]School of Software, Shanghai Jiao Tong University, Shanghai 200240, China.
[2]IBM Client Innovation Center China, Shanghai 200433, China.
bjshen@sjtu.edu.cn

ABSTRACT

Software size estimation is a crucial step in project management. According to the Standish Chaos Report, 65% of software projects are over budget or deadline; therefore, a good size estimation method is very important. However, existing estimation methods are complicated and human-effort consuming. In many industrial projects, project technical leads (PTLs) do not use these methods but just give a rough estimation based on their experience. To decrease human effort, we propose an early software size estimation (ESSE) method, which can extract semantic features from natural language requirements automatically, and build size estimation models for project. Firstly, ESSE makes a two-level semantic analysis of requirements specification documents by information extraction and activation spreading. Then, complexity-related features are extracted from the results of semantic analysis. Finally, a size estimation model is trained to predict size of new projects by regression algorithms. Experiments in real industrial datasets show that our method is effective and can be applied to real industrial projects.

CCS Concepts

•**Computing methodologies** → **Information extraction;** *Feature selection;* •**General and reference** → *Estimation;* •**Software and its engineering** → Requirements analysis;

Keywords

Semantic Analysis, Requirements Analysis, Software Size Estimation.

1. INTRODUCTION

It is important to predict how much size will be required for a software project as early as possible. Underestimation is one of the main problems that impact the success of software projects. Meanwhile, if size of a software project can be estimated in early phase, project technical leads (PTLs) can make a better plan. Usually, traditional software development consists of four main stages: requirements elicitation, design, coding and testing. Most size estimation methods have a good estimation after design. If estimation could be finished directly after requirements elicitation, it can positively affect project management. Precise early size estimation is critical not only in traditional software development, but also in agile development, as it makes projects more manageable for the parts, where requirements are less volatile.

However, most existing automatic estimation methods [1][2] only make simple word-counting analysis of requirements specification documents (RSD) rather than semantic analysis. Besides, there are also many learning-based size estimation methods [3][4], but these methods focus on learning algorithms and ignore requirements analysis. It will make estimation more accurate if requirements semantic analysis and a proper learning algorithm are integrated together.

In this paper, we propose an early software size estimation method (ESSE) based on requirements semantic analysis and machine learning. Specifically, our method consists of three steps: (1) Entity extraction from requirement specification. (2) Complexity-related features extraction from extracted entities. (3) Construction and application of size estimation model. Finally, experiments using real data from industrial and commercial projects show that our method is directly applicable.

Contributions of this paper are: (1) We propose a unified early software size estimation method that integrates natural language processing and machine learning techniques and it achieves superior estimation results. (2) We propose a two-level requirements semantic analysis algorithm to extract complexity-related features from requirements. (3) With complexity-related features and size drivers, regression models for size estimation are then constructed from historical data to predict a new project size.

2. METHODOLOGY

Fig.1 illustrates the overview of early software size estimation method we proposed. Firstly, ESSE processes each requirement in a requirement specification document (RSD) using chunk-level semantic analysis technology, and extracts a five-tuple entity. Further, ESSE extracts local features and global features to make word-level semantic analysis. And then, using these features, size drivers and real sizes of historical project data, the size estimation model can be established by regression algorithms. Finally, ESSE can es-

Internetware '16, September 18 2016, Beijing, China
© 2016 ACM. ISBN 978-1-4503-4829-4/16/09…$15.00
DOI: http://dx.doi.org/10.1145/2993717.2993733

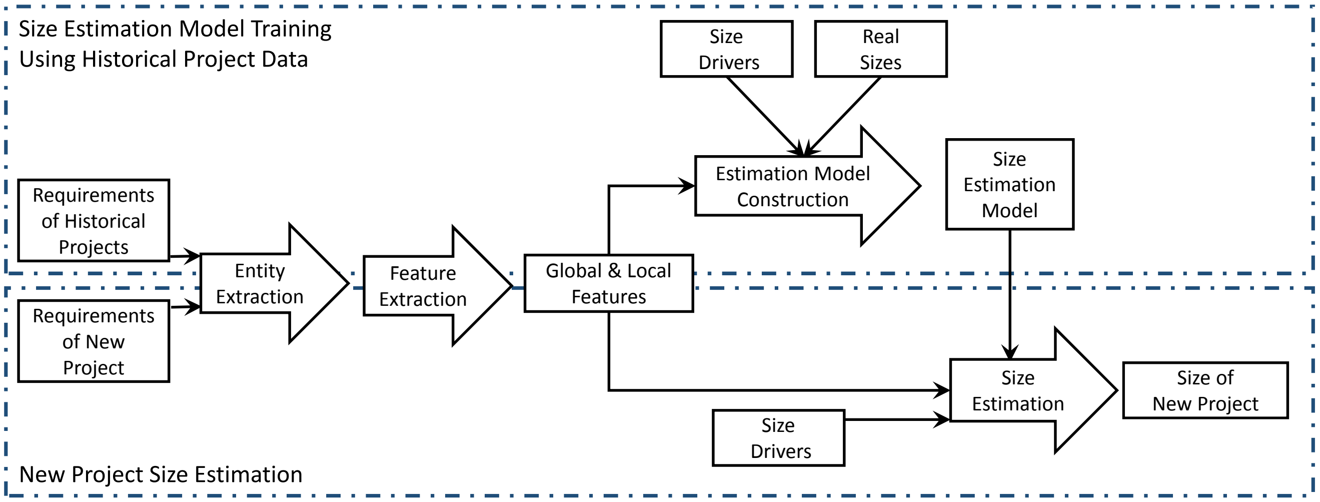
Size Estimation Model Training Using Historical Project Data
Size Drivers
Real Sizes
Estimation Model Construction
Size Estimation Model
Requirements of Historical Projects
Entity Extraction
Feature Extraction
Global & Local Features
Requirements of New Project
Size Estimation
Size of New Project
Size Drivers
New Project Size Estimation

Figure 1: The Overview of Early Software Size Estimation Method.

timate the size of a new project using this size estimation model.

2.1 Entity Extraction

The chunk-level requirements semantic analysis is to do semantic chunking to each requirement in a requirements specification document for size estimation features extraction. For each requirement in a RSD, a five-tuple:

$(User, Action, Object, ActionProperty, Condition)$

will be extracted. $User$ is the executor of the requirement, $Action$ is usually verbs in the requirement describing actions executed by $User$, $Object$ is the object of $Action$, $ActionProperty$ is the characteristic of $Action$, such as how to do the action, and $Condition$ is the precondition or post-condition of $Action$. For any five-tuple, $User$, $Action$ and $Objection$ are necessary, while $ActionProperty$ and $Condition$ are optional. These five entities are integrated into one model. We apply a sequential labeling algorithm to ESSE, called Conditional Random Fields (CRFs) [5], to train a model from historical data for extracting these entities from new data.

In the entity extraction step, input features are extracted from two levels, one is the word sequences of a requirement and the other is the corresponding Part-Of-Speech (POS) tagging sequences of the word sequences. We use the unigram word with POS features and bigram word with POS features in ESSE, which is called template in CRF++. A group of word level features we use are W_{-2}, W_{-1}, W_0, W_{+1}, W_{+2}, $W_{-1}W_0$ and W_0W_{+1}, where W stands for a word, index 0 indicates the current word in focus and indices $-n/+n$ indicate the n^{th} word to the left/right of the current word. Similarly, an example group of POS level features is POS_{-2}, POS_{-1}, POS_0, POS_{+1}, POS_{+2}, $POS_{-1}POS_0$ and POS_0POS_{+1}. Label for each word is $y_i \in \{B, I\} \times \{User, Action, Object, ActionProperty, Condition\}$, for $B-t$ are beginning words of tuple element t and $I-t$ are inner words of tuple element t.

2.2 Feature Extraction

After extraction of 5-tuple entity, complexity-related features can be extracted to train a size estimation model. For each requirement in a RSD, we extract two kinds of features: Global Features and Local Features.

Global Features. Global features are a kind of coarse-grained features extracted from the whole RSD, which reflect the complexity of the system involved in the document. All requirements in a RSD share the same global features. Totally five global features are extracted: (1) the number of requirements; (2) the number of unique nouns in the RSD; (3) the number of unique nouns in the $Object$ fields of all requirements; (4) the number of user categories in the $User$ fields of all requirements; and (5) the number of unique actions in the $Action$ fields of all requirements.

Local Features. Compared to global features, local features aim at measuring the complexity of a RSD. Totally, there are five kinds of local features: (1) the number of unique nouns in the requirement; (2) the number of unique nouns in the $Object$ field of the requirement; (3) whether the $ActionProperty$ in the five-tuple of the requirement is null; (4) whether the $Condition$ in the five-tuple of the requirement is null; and (5) a semantic vector of the $Action$ field. We consider verbs are more suitable to be vectorized than nouns. However, verbs in different projects are almost the same, we use Bag-Of-Words (BOW) [6] in ESSE to represent $Action$ of a requirement. But only extracting BOW to represent $Action$ of a requirement will lead to a sparse representation problem.

A word-level semantic analysis is proposed to solve these problems by introducing a new data structure - WordNet[1]. To utilize WordNet, we apply a propagation-based technique to solve the sparsity problem in ESSE.

Inputs of spreading activation are a similarity matrix S, where each element $s_{i,j}$ in S is the similarity between $action_i$ and $action_j$, and the initial state of actions, denoted by A. During the process of spreading activation, for each action in A, in one iteration, it propagates its weight to other actions with corresponding similarities. To apply this technique, the most important step is to construct the similarity matrix in which each element is the similarity between two actions. We propose Synonym Similarity Matrix (SSM) and Part-of

[1]https://wordnet.princeton.edu/

Similarity Matrix (PSM).

Because WordNet provides synonym relations between common words, synonym similarity matrix can be constructed from it. SSM can be constructed from WordNet synonym relations and to retain more meaning of actions, for each word, we assign a higher weighted value to its self-connected edge. Supposing there are $|S_w|$ synonyms of word w, we replace the self-connected weight of it (the $SSM_{w,w}$ element in SSM) from 1 to $|S_w|$. Different from synonym relations, part-of relations are asymmetric, and thus PSM is asymmetric. Next, we construct PSM for manually constructed part-of relations. If word u is part-of v, then in the matrix, $PSM_{u,v}$ is set as α and $PSM_{v,u}$ as β. In our experiment setting, we set $\alpha = 0.4$ and $\beta = 0.2$ by results of parameters tuning.

After obtaining these two similarity matrices, we can enhance the semantic meaning of actions vector to solve the sparsity problem. Algorithm 1 is the actions enhancement algorithm. The inputs are SSM, PSM, the original actions vector oav and the number of iteration t. The output is the enhanced actions vector eav. In each iteration, the eav is updated by multiple SSM firstly (line 3) and then is normalized (line 4). Secondly, the eav is updated by multiple PSM (line 5), followed by a normalization process. After t times propagation, vector eav will be returned.

Algorithm 1 Actions Enhancement

```
Input:
    The Synonym Similarity Matrix $SSM$;
    The Part-of Similarity Matrix $PSM$;
    The number of iteration $t$;
    The original actions vector $oav$;
Output:
    An enhanced actions vector $eav$
1: $eav = oav$
2: for $i = 1$ to $t$ do
3:     $eav \leftarrow eav * SSM$
4:     $eav \leftarrow normalize(eav)$
5:     $eav \leftarrow eav * PSM$
6:     $eav \leftarrow normalize(eav)$
7: end for
8: return $eav$;
```

2.3 Construction and application of estimation model

Learning from features and real sizes of historical project data, we construct the size estimation model using regression algorithms. There are many kinds of mature regression algorithms we can select, e.g. Support Vector Regression (SVR), Bayesian Regression (BR) and Decision Tree Regression (DT). Size drivers are also factors of estimation model. In our method, we borrow five drivers from COCOMO II [7]: (1) Precedentedness of Product; (2) Development Flexibility; (3) Architecture/Risk Resolution; (4) Team Cohesion; and (5) Process Maturity.

Finally, we can estimate the size of a new project by the size estimation model. ESSE will extract entities from the requirement specification of the new project, and further extract features from the five-tuple entities. Combining with five size drivers, we can obtain the estimated size of each statement in the requirement specification, using the estimation model. As for the whole estimated size of a new project, we simply sum up all sizes of statements in requirement specification.

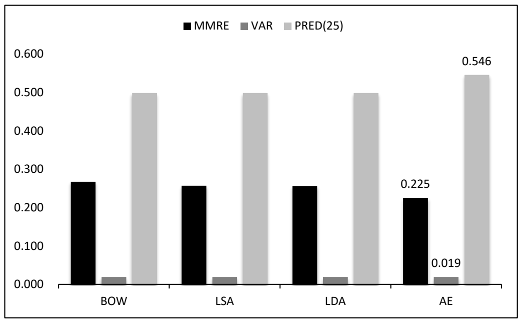

Figure 2: Results of Different Algorithms in Word-Level Semantic Analysis.

3. EXPERIMENTS

We totally collect 39 industrial projects in the commercial field from 5 companies, including IBM, whose programming languages are various, such as JAVA, C#, C++ etc. There are 21 projects and totally 508 requirements in the RSDs of these projects. For each requirement, we can obtain its source line of code (SLOC). Also, for each project, we ask its PTLs to fill a questionnaire to obtain its drivers. We use Magnitude of Relative Error(MRE, Eq.(1)) to calculate error between real values and estimated values. Mean Magnitude of Relative Error, Variance of Relative Error(VAR) and Percentage of estimation within 25% of actual values(PRED(25)) can be further calculated.

$$MRE_i = \frac{|Real_i - Estimated_i|}{Real_i} \qquad (1)$$

3.1 Performance of Entity and Feature Extraction

In the process of chunk-level semantic analysis, we apply CRFs-based algorithm to solve the problem and compare the result with other classifiers i.e. Naive Bayes (NB) and Support Vector Machine (SVM). Because of limited space, we do not demonstrate the experiment results here.

In the process of word-level semantic analysis, to solve problems in BOW vectors of action features in size estimation, we apply Action Enhancement to do the word-level semantic analysis. This experiment compares the performance of Action Enhancement and those of topic models, which are LSA [8], LDA [9] and so on. The regression algorithm we use to train these size estimation models in this experiment is SVR. The results are illustrated in Fig.2, which show that actions represented by original BOW obtain the worst results. To sum up, to do the word-level semantic analysis, our algorithm achieves the best results.

3.2 Performance of Size Estimation

This experiment is used to select a proper regression algorithm for training size estimation model. It estimates size of each requirement in a RSD. Fig.3 illustrates results of comparison of three different regression algorithms: Support Vector Regression (SVR), Bayesian Regression (BR) and Decision Tree Regression (DT). Also, we compare results with that of Hussain's method [2]. From the Fig.3,

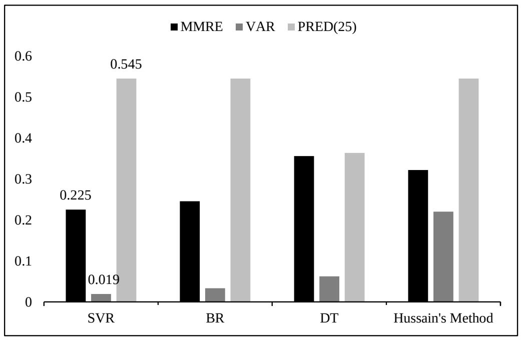

Figure 3: Results of Comparison of Different Regression Algorithms and Existing Methods in Size Estimation.

SVR obtains the best result, reaching 0.225 MMRE, 0.019 VAR and 0.545 PRED(25), followed by BR, just slightly worse than results of SVR, and DT performs the worst. As for Hussain *et al.*'s method, although it achieves the same PRED(25) as SVR, the VAR of it is too larger, and the MMRE of it is also higher than SVR. To sum up, we find the best regression algorithm for our estimation model is SVR.

Besides global and local features, there are also many other factors affecting the software. These factors, which are called size drivers, cannot be obtained from RSDs automatically. All these drivers can be obtained by asking PTLs with questionnaires. In the questionnaire, each driver is in six levels (0-5), and the higher the level is, the better the driver is. We conduct an experiment to study how three kinds of features: local features, global features and drivers contribute to the size estimation model. The results show these three kinds of features have positive contribution to model more or less. Because of limited space, we do not demonstrate the experiment figure here.

4. RELATED WORK

Various models can help measure software size manually. One simple measurement is Source Lines of Code (SLOC) [10]. Compared to SLOC, functional size [11] is a more objective and widely used size measurement. Despite the precision and objectivity of manual size measurements, the complexity makes them cost a lot of human effort. Nevertheless, some findings give ideas to make approximate estimation available. Bowden *et al.* found that the number of objects in requirements can be used to estimate software size at an early stage [12]. Ayyildiz *et al.* extracted conceptual class from Use Case and found its linear correlation with size [13]. In addition, many extended approaches of functional size estimation have also been proposed to make it automatic. Ceke *et al.* proposed a method to combine CFP and conceptual model to estimate size in web application development [1]. Hussain found ten linguistic features most highly correlated with CFP, such as the frequency of Noun Phrases, Parentheses, and Active Verbs [4]. On the basis of these features, historical data and machine learning, the software size can be automatically classified into four categories: Small, Medium, Large and Complex. However, all these extended methods try to make functional size estimation automatic, but they all just give a rough estimation.

5. CONCLUSION

In this paper, we propose an early software size estimation (ESSE) method based on requirements semantic analysis and machine learning. Firstly, ESSE makes a two-level semantic analysis of requirements specification documents. Then, features are extracted from semantic analysis results. Finally, a size estimation model is trained to predict size of new projects.

6. ACKNOWLEDGMENTS

This research is supported by 973 Program in China (Grant No. 2015CB352203) and National Natural Science Foundation of China (Grant No. 61472242).

7. REFERENCES

[1] Denis Čeke and Boris Milašinović. Early effort estimation in web application development. *Journal of Systems and Software*, 103:219–237, 2015.

[2] Ishrar Hussain, Leila Kosseim, and Olga Ormandjieva. Approximation of cosmic functional size to support early effort estimation in agile. *Data & Knowledge Engineering*, 85:2–14, 2013.

[3] Leandro L Minku and Xin Yao. Software effort estimation as a multiobjective learning problem. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 22(4):35, 2013.

[4] HM Hussain. *Linguistic Approaches for Early Measurement of Functional Size from Software Requirements*. PhD thesis, Concordia University, 2014.

[5] John Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. 2001.

[6] Hanna M. Wallach. Topic modeling: beyond bag-of-words. In *International Conference on Machine Learning*, pages 977–984, 2006.

[7] Robert E Park. Software size measurement: A framework for counting source statements. Technical report, DTIC Document, 1992.

[8] Thomas K Landauer, Peter W Foltz, and Darrell Laham. An introduction to latent semantic analysis. *Discourse processes*, 25(2-3):259–284, 1998.

[9] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *the Journal of machine Learning research*, 3:993–1022, 2003.

[10] Ifpug: Fsm method: Iso/iec 20926:2009, software and systems engineering - software measurement - ifpug functional size measurement method.

[11] Roberto Meli and Luca Santillo. Function point estimation methods: A comparative overview. *Accident Analysis and Prevention*, 11(1):1–5, 1999.

[12] P Bowden, M Hargreaves, and Caroline S Langensiepen. Estimation support by lexical analysis of requirements documents. *Journal of Systems and Software*, 51(2):87–98, 2000.

[13] Tülin Erçelebi Ayyıldız and Altan Koçyiğit. An early software effort estimation method based on use cases and conceptual classes. *Journal of Software*, 9(8):2169–2173, 2014.