An approach to software development effort estimation using machine learning

Vlad-Sebastian Ionescu
Faculty of Mathematics and Computer Science
Babeş-Bolyai University
1, M. Kogălniceanu Street, 400084, Cluj-Napoca, Romania

***Abstract*—We introduce a machine learning approach for real life software development effort estimation. Our method uses state of the art developments such as distributed word embeddings in order to create a system that can estimate effort given only basic project management metrics and, most importantly, textual descriptions of tasks. We use an artificial neural network for automating the effort estimation task. We evaluate our method on genuine software project data from a software company, obtaining results that surpass some of the related literature and a system that promises much easier integration into any software management tool that stores textual descriptions of tasks.**

I. Introduction

Software development effort estimation (SDEE) is an ever-present part of software development process. It is usually done by developers, who, upon seeing what is asked of them (from fixing a bug to implementing a new feature), provide a time estimate (in anything from hours to months). It is important for these estimates to be as accurate as possible, in order to have a correct picture of how the software is progressing so that management can make the right decisions about the project.

There are various strategies that developers can use in order to improve their estimates. Planning poker is a group method in which an entire team must reach consensus about the estimates [1]. This has the advantage of favoring a deeper analysis by each developer, since consensus must be reached. This usually leads to better accuracy. However, it also has the disadvantage that more time has to be spent not actually producing anything.

In this paper, we introduce an automated machine learning-based SDEE method based on the text of tasks. This is similar to methods such as COCOMO [2], [3] and the Putnam model [4] in that our estimates are automatically provided by a program. The difference is that, unlike COCOMO, Putnam and other similar methods, our approach learns how to make predictions from historical project data. Moreover, our approach relies primarily on the textual descriptions of tasks, which are almost always available, unlike various metrics that other frameworks require. This makes it more robust and easier to incorporate in a project than other alternatives. Our approach is novel, with only one other similar approach in the literature.

The remainder of the paper is structured as follows. The motivation behind the SDEE problem is given in Section **??**. A brief literature review on SDEE is presented in Section II, followed by the difficulty and state of the art in Section III. Some machine learning background and an overview of the machine learning elements we use is given in Section IV. Section V describes our data sets and provides experimental details, followed by Section VI where we describe our results and compare them to some existing ones. Our ideas for future work and possible improvements are given in the final Section VII.

II. Software development effort estimation

This section presents an overview of *Planning Poker*, an Agile SDEE methodology, and existing automated approaches for SDEE.

In order to obtain an automation framework that closely resembles the real life effort estimation process, we consider a text processing machine learning approach to the SDEE problem. Our hope is that, since most projects have some way of tracking solved and new tasks by their textual descriptions and their required time, we can use machine learning to make use of this data and build models which can predict the required time for new tasks but learning from the old ones.

A *task* is the smallest unit usually estimated in Agile methodologies. It always has a description of what is required from the developers responsible with completing it [1]. Developers usually have to provide time estimates to their project managers based on this textual information. At first, this text is the only input provided to developers. Of course, programmers can also tap into their domain and project knowledge, which is something that machine learning algorithms cannot currently make very good use of. However, it stands to reason that there might still be important relations that can be learning from the text, the time and perhaps the assignee of each task.

If available, automated SDEE has three important usages.

1) Verifying and correcting developer estimates. This is useful in situations when establishing trust can be difficult.
2) It allows for better task allocation to developers. We can build a predictive model for each team member and use them to assign tasks to whoever is more likely to finish them first.
3) It saves the time that developers would use to provide estimates or attend planning poker meetings.

It is clear that an automated solution for SDEE, which machine learning has the potential for providing, can have a lot of benefits, for developers, clients and management alike.

A. Planning Poker

Planning Poker is a consensus-based method for predicting programming effort. Its goal is to force developers to reach consensus in their estimates without improperly influencing each other.

Planning Poker usually consists of a deck of cards with Fibonacci numbers on them, starting from 0 up until 89. These represent effort, measured in any agreed upon unit, for example hours [1].

Cards are laid face down and revealed all at once. This process continues until a consensus is reached, with discussions taking place between rounds. The discussions force developers to think about their choices so that they are able to explain them.

Although quite accurate in practice, it has the disadvantage of taking a lot of time and involving a lot of people.

B. Algorithmic approaches to software development effort estimation

Computational approaches to SDEE usually consist of formulas on various code metrics and on domain knowledge. These are also called *parametric models*.

C. COCOMO

COCOMO [3] starts by dividing projects into three types:

1) *Organic*: projects with small teams, good experience, and lax requirements;
2) *Semi-detached*: Projects with medium-sized teams, mixed experience levels, and mixed requirement types ;
3) *Embedded*: A combination of the previous two.

COCOMO provides three formulas:

$$\begin{aligned} E &= a_b \times KLOC^{b_b} \\ D &= c_b \times E^{d_b} \\ P &= \frac{E}{D} \end{aligned} \tag{1}$$

where:

- E represents the *effort applied*, expressed in man-months;
- D is the *development time*, expressed in months;
- P is the number of people required;
- $KLOC$ is the estimated number of delivered thousands of lines of code for the entire project.

The other coefficients are constants that depend on the project type, and are given in Table I.

The basic COCOMO method can be improved to consider other factors as well, such as hardware data.

A crucial disadvantage of COCOMO is its hardcoded constants for various elements. This makes it lack robustness and adaptability. Therefore, it is unlikely to be either easy to use in practice or accurate for anything but very specific projects. Furthermore, it still requires quite a lot of human input.

Project type	a_b	b_b	c_b	d_b
Organic	2.4	1.05	2.5	0.38
Semi-detached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

TABLE I
CONSTANTS USED IN THE COCOMO FORMULAS FOR EACH PROJECT TYPE.

D. Putnam model

The Putnam model [4] uses Formula (2), where:

- S is the project size, usually measured in Effective Source Lines of Code [5].
- B is a scaling factor dependent on S.
- Pr is the process productivity, defined as the ability of the development team to stay within a given defect rate. This is distinct from the more conventional definition of the size divided by effort.
- E is the effort, expressed as the total person-years allocated to the project.
- T is the total number of years allocated to the project.

$$\frac{B^{\frac{1}{3}} \times S}{Pr} = E^{\frac{1}{3}} \times T^{\frac{4}{3}} \tag{2}$$

To provide an effort estimate for a given task, the equation in Formula (2) is solved for E, as shown in Formula (3).

$$E = \left(\frac{S}{Pr \times T^{\frac{4}{3}}} \right)^3 \times B \tag{3}$$

It is known that the method is sensitive to the S and Pr parameters, which must be estimated by human factors. The process productivity can be calibrated according to Formula (4) [4].

$$Pr = \frac{S}{\left(\frac{E}{B}\right)^{\frac{1}{3}} \times T^{\frac{4}{3}}} \tag{4}$$

The Putnam model has the advantage of easier calibration. Nevertheless, it still requires human estimations, which can contribute to its lack of accuracy in practice.

Many similar approaches exist, such as SEER-SEM [6]. However, their approaches are not very distinct from COCOMO and the Putnam model: they all usually rely on fixed equations and subjective measurements, which all add noise and thus decrease accuracy.

III. DIFFICULTY AND STATE OF THE ART

An algorithmic solution for the SDEE problem is difficult for many reasons, such as:

- Each project is different from many points of view.
- Each team is also different in many ways.
- Machine learning algorithms currently cannot make use of things that would definitely be useful for this problem, such as a programmer's knowledge. They can only learn relations between various easily quantifiable features of the task, the programmer and the project.

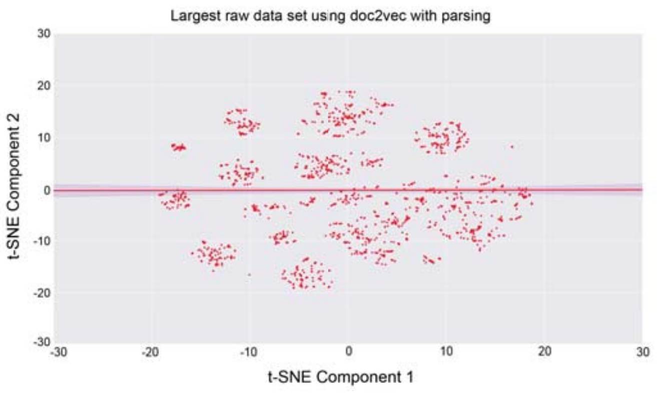

Fig. 1. Visualizations of the doc2vec transformer reduced to two dimensions on a data set from [7].

- There is very little public data available to study and train on.

We have introduced in [7] a similar approach on different data and using a different approach involving Support Vector Regression. Figure 1 presents a t-SNE [8] plot on a part of this data, on which doc2vec was applied, reduced to two dimensions.

We can see that it is a hard data set, with no obvious patterns. Our current data is similar in complexity and nature. This is true in general about the problem: it is hard to find obvious patterns.

The existing literature on SDEE is very diverse, with approaches ranging from classical COCOMO-like models applied to various projects to machine learning methods similar to what we propose and from results ranging from very good to unusable in practice. Moreover, the nature of such research makes it very difficult to compare methods and results, since usually neither the data sets used nor any software that implements the proposals are public. This is understandable considering that usually private client data is used in experiments.

Nevertheless, we consider it worthwhile to present existing approaches and make quick comparisons, in order to give as much context as possible to our proposal.

A lot of the literature on this subject, especially earlier research, deals with parametric models, such as COCOMO, SEER, SLIM and others. One study by Basha and Ponnurangamthe regarding these methods [9] found MMRE (Mean Magnitude of Relative Error) error values between 0.373% and 771.87%. This large interval leads to the conclusion that there is no silver bullet framework, and a lot depends on the available data.

Open source projects are also sometimes considered. Toka applies COCOMO II, SEER-SEM, Slim and TruePlanning on a group of them in [10], obtaining MMRE values between 34% and 74%.

Another study by Tharwon [11] surveys experimental research that uses the Function Point Analysis (FPA), Use Case Point Analysis (UCPA) and COCOMO models. The found error rates are once again highly variable between methods and projects.

Du et al. step into machine learning territory with a neuro-fuzzy enhanced SEER-SEM approach [12], obtaining significant improvements over SEER-SEM of up to 15 percentage points.

Han et al. apply in [13] more machine learning algorithms, such as linear regression and Gaussian processes, obtaining usable results with most of them.

Bayesian networks, regression trees, backward elimination, stepwise selection have also been applied, with good results, by van Koten and Gray in [14].

Wen et al. show in [15] that, even when using machine learning, MMRE values fluctuate a lot between different projects and learning algorithms. This suggests that simply switching to an ML approach does not solve the SDEE problem: we still have to find something tailored for each particular project needs.

According to the literature, approaches that use machine learning obtain better results than those using classic parametric models. However, the same major issue remains: the need for various project metrics that take time and effort to collect.

To the best of our knowledge, the only one approach researching the possibility of working directly with textual data is the work of Sapre [16]. A bag of words method with keywords extracted from Agile story cards is used, with promising results. Our goal is to improve these results especially, since this is the best work to compare ourselves against.

IV. Our approach

Our approach consists of the following steps, which make up our machine learning pipeline:

1) **Vocabulary and output preprocessing**. We sort the words in our training set by the standard deviation of the known completion times of the tasks in the training set in which each word appears. For each task, we only keep a percentage of the words in it (as long as the number of resulting words is above a given value) based on this statistic. We also offer the possibility, based on a hyperparameter setting, of duplicating some of these words.
   The resulting tasks are concatenated with the available, one-hot encoded, project metrics. The textual part and the numeric metrics part are kept track of independently, which allows us to apply further processing on each part independently, while still treating them as features of instances. This is an original statistical preprocessing method that we have experimentally determined to lead to improved results.
   We also take the logarithm of our targets (our targets being the known completion times of the tasks), since they are exponentially distributed in our data set.
2) **TF-IDF or doc2vec**. After our custom vocabulary preprocessing, we feed the textual part of our tasks to a TF-

IDF or doc2vec [17], [18] (experiments are performed for each) model that outputs numerical data.

TF-IDF is a weighting algorithm for terms from a text corpus. It represents the multiplication between how many times a term t appears in a document d (the *term frequncy*) and the inverse of the documents that contain the term (the *inverse document frequency* This is shown in Formula (5), where n_d is the total number of documents and nd_t is the number of documents that contain the term t. There are multiple ways of scaling the inverse document frequency.

$$\begin{aligned} idf(t) &= 1 + log\frac{1+n_d}{1+nd_t} \\ tf\text{-}idf(d,t) &= tf(d,t) \times idf(t) \end{aligned} \tag{5}$$

The resulting $tf\text{-}idf$ values are usually normalized using the L2 norm, described in Formula (6) [19].

$$v_{L2} = \frac{v}{\sqrt(v_1^2 + v_2^2 + ... + v_n^2)} \tag{6}$$

Models such as word2vec [17] and doc2vec [18] are, in theory, more powerful than classical bag of words models. For example, in TF-IDF, it is hard to draw conclusions about the relationship of words: ”Germany” and ”Spain” are likely to be as related as ”Carburator” and ”Bird”. In the case of word2vec, the model would learn, given a large enough corpus, that ”Germany” and ”Spain” are both countries, and we would be able to extract this relationship from the computed vector representations.

This is achieved in word2vec [17] by training a model to predict a word given a *context* (or, as an alternative, training a model to predict a context given a word), which is a set of words around it. This leads to words with similar meaning being positioned closer together in the vector space, because it is likely that words that share a taxonomy will appear in similar contexts.

A famous example of the ability to learn semantics is given in Formula (7), which expresses the fact that subtracting the vector for “man” from the vector for “king” and adding the vector for “woman” will lead to a vector that is very similar to the vector for “queen”.

$$\begin{aligned} v(\text{“}king\text{”}) &\quad - \\ v(\text{“}man\text{”}) &\quad + \\ v(\text{“}woman\text{”}) &\quad = \\ v(\text{“}queen\text{”}) & \end{aligned} \tag{7}$$

As shown in [18], there is an extension of this idea to paragraphs as well, allowing us to obtain vectors for entire documents and to infer new vectors for unseen paragraph.

Due to the important property of retaining semantics better, the algorithm has the potential to better aid our regressor learner than classical TF-IDF.

3) **Min-max normalization**. The fully numerical output so far is normalized according to the min-max method. This has the effect of converting every feature value of an instance to a value in the interval $[0, 1]$. Formula (8) presents this process.

$$X_{normalized} = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{8}$$

4) **Artificial neural network**. The resulting features are given to a neural network that learns to perform a regression task in order to predict MMRE. MMRE is defined as in Formula (9). Note that this is sometimes multiplied by 100 in order to obtain a percentage.

$$MMRE = \frac{1}{n}\sum_{i=1}^{n}\frac{|Actual_i - Estimated_i|}{Actual_i} \tag{9}$$

Artificial neural networks can be seen as a generalization of linear regression. They are made of *neurons* arranged in successive *layers*, starting from the input layer to the output layer, through a number of computational layers. Each input layer neuron takes as input one feature and outputs it to all the neurons in the first computational layer. Each neuron in a computational layer performs a weighted sum of its input, applies an *activation function*, and outputs the results to all the neurons in the next layer [20].

Various activation functions exist, such as the Logistic Sigmoid (Formula(10)) and the Hyperbolic Tangent (Formula (11)).

$$f(x) = \frac{1}{1 + exp(-x)} \tag{10}$$

$$f(x) = tanh(x) \tag{11}$$

The network is then trained using backpropagation with gradient descent or other optimizers [20].

5) **Random search**. A random search [21] is performed in order to optimize the hyperparameters for each of the above elements, over a fixed number of iterations. We perform this search on 67% of our data set, and report results on evaluating the resulting optimized pipeline by using it to make predictions on the rest of the data.

V. Data sets and experiments

Our data set contains real world data from a local software company. It consists of 7826 instances obtained from a project management software. Each instance describes a task and contains features such as the task title, description, type, reporter, team, developer responsible, severity and others.

There are 5 possible task types, based on whether they are for fixing bugs, implementing features and other such development classifications. We run experiments on the entire data set as well as on each task type separately.

For each task, we know the time it took to complete it. This is what we will learn in our supervised learning task. We also know the human estimates, which we use to compute the human MMRE value for each data set.

Table II presents a summary of our data sets. We only use 2000 iterations on the full data set because the execution time is too high otherwise for so many instances and with a cross validation for each hyperparameter sample.

Task type	Instances	Search iterations	Human MMRE
All	7826	2000	0.466
1	1679	5000	0.445
2	1191	5000	0.554
3	4366	5000	0.452
4	353	5000	0.422
5	237	5000	0.474

TABLE II
SUMMARY OF DATA SETS.

Table III describes what hyperparameters we consider in our random search and the possible values we sample them from. All of our experiments are performed with the help of the scikit-learn machine learning library [19]. Gensim is used for the doc2vec implementation [22].

Pipeline element	Considered hyperparameters
Vocabulary preprocessing	Minimum number of words to keep, method of word duplication, parameters associated with the method of duplication.
TF-IDF or doc2vec	All hyperparameters pertaining to these models: whether to use IDF, smoothing, how many features to keep, ngram types and their range, doc2vec type, learning rates, number of learning steps and others. Most of the parameters available in [22] and some others that we introduced for our purposes.
Artificial neural networks	Discrete configurations of number of layers (maximum 2), number of nodes in each layer, learning rates, activation functions, solving algorithms. Most of the parameters available in [19].

TABLE III
HYPERPARAMETER SEARCH DESCRIPTIONS.

We have gradually tuned our hyperparameter search over multiple runs, eliminated variants that never showed up in the top solutions.

VI. RESULTS AND COMPARISON TO RELATED WORK

As mentioned in the previous sections, our experiments are performed on each subset of the data, using both doc2vec and TF-IDF. Table IV shows our results on each subset together with the human estimates MMRE values. The best results are highlighted in green and the second best in yellow.

Figure 2 shows our results in graphical form.

From Table IV and Figure 2 we can see that our method obtains the best results on two subsets, both times using doc2vec. While most of the time we are unable to outperform the human estimates, our results are encouraging considering that we mostly rely only on text data.

Moreover, our method performs the best on the subsets with fewer instances, which represent more specialized type of

Task type	Instances	Method		Human MMRE
		doc2-vec	TF-IDF	
All	7826	0.889	0.814	0.466
1	1679	0.424	0.673	0.445
2	1191	0.779	0 .747	0.554
3	4366	0.818	0.834	0.452
4	353	0.538	1.310	0.422
5	237	0.408	0.409	0.474

TABLE IV
RESULTS AND COMPARISON TO HUMAN ESTIMATES - TABLE FORM.

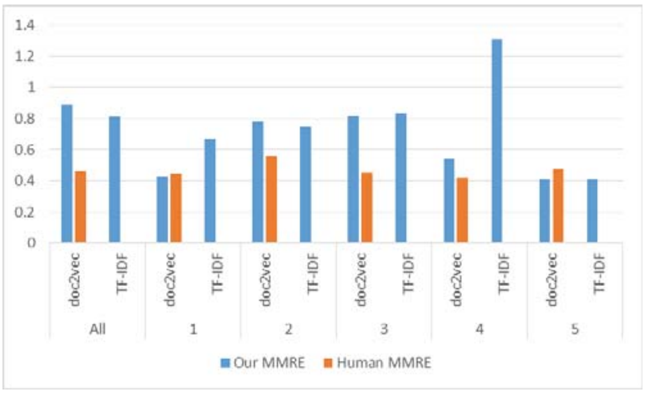

Fig. 2. Results and comparison to human estimates - chart form.

tasks. This suggests that more specialized tasks are easier to learn by machine learning algorithms. Our hypothesis is that such tasks have a clearer structure deducible from their task descriptions without requiring much knowledge of the entire code base.

More general tasks, on the other hand, might span a wider project area, and thus require intimate knowledge of multiple areas of the project in order to make proper estimates. Since our method does not incorporate any global project knowledge in its learning, and since our available instances are quite few for such general tasks, learning might be hindered.

Table V presents a comparison of our best result on the entire data set (0.814) with related work from the literature. We can see that our approach leads to better results than those presented in three other studies, which usually analyze more than one project. Compared to five studies, our results are better than some of the presented results and worse than others, while our results are worse only compared to four studies.

VII. CONCLUSION

We have shown that machine learning algorithms have the potential to provide good results for the SDEE problem. Although yet unable to surpass human estimations, our method outperforms a significant number of other approaches. Furthermore, its novelty of only requiring textual data and easily available metrics makes it easy to apply to any project in very little time.

More research needs to be done on the text processing algorithms in order to improve results. We also plan on researching how to incorporate existing project knowledge into

Related work	Description	MMRE
[11]	Function Point Analysis survey.	13.8% – 1624.31%, 90.38 average.
[16]	Machine learning applied on raw text data from Agile story cards.	92.32%
[13]	Multiple machine learning models.	87.5% – 95.1%
[9]	COCOMO, SEER, COSEEKMO, REVIC, SASET, Cost Model, SLIM, FP, Estimac and Cosmic frameworks on flight software and business applications.	0.373% – 771.87%
[11]	Use Case Point Analysis survey.	27.30% – 88.01%, 39.11% average
[14]	Bayesian networks, Regression trees, Backward elimination and Stepwise selection on metrics. Done on two software projects.	97.2% on one of the projects and 0.392% on the other.
[15]	Machine learning for SDEE survey.	13.55% – 143%
[23]	Linear regression and radial basis function networks on metrics.	66% – 90% for linear regression, 6% and 90% for RBF networks.
[10]	COCOMO and TruePlanning on open source projects.	30% - 74%
[12]	Neuro-fuzzy enhancement for SEER-SEM.	29.01% – 69.05%
[23]	Planning poker, Use Case Point Analysis and human estimates.	48% for planning poker, 2% – 11% for UCPA, 28% – 38% for human estimates.
[24]	COCOMO with project metrics.	10% – 46%

TABLE V
COMPARISON TO RELATED WORK. THE RELATED WORKS WITH HIGHER AVERAGE MMRE VALUES THAN OUR BEST RESULT ON THE FULL DATA SET ARE MARKED IN GREEN. THE WORKS THAT WE PROVIDE A MMRE INTERVAL FOR AND FOR WHICH WE DO BETTER THAN THE UPPER BOUND ON THE FULL SET ARE MARKED IN YELLOW. WORKS THAT CLEARLY OBTAIN BETTER MMRE VALUES ON THEIR DATA SETS THAN WE DO ON OUR FULL SET ARE LEFT IN WHITE.

our predictions, since this is an important contributing factor that our current approach ignores.

Another important factor is developer experience. A developer approaches a task with a lot of knowledge about the project and the skills required to work on that project, however, this knowledge is not trivial to properly quantify and feed to a machine learning algorithm. More research is required in this area as well.

Compared to other automated methods, our machine learning approach is therefore easier to use and provides better results than other software. It does not yet outperform human estimates, so the accuracy versus time spent estimating is a tradeoff that still requires consideration.

ACKNOWLEDGMENT

This work was supported by a grant of the Romanian National Authority for Scientific Research and Innovation, CNCS–UEFISCDI, project number PN-II-RU-TE-2014-4-0082.

REFERENCES

[1] M. Cohn, *Agile Estimating and Planning*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2005.
[2] B. W. Boehm, *Software Engineering Economics*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 99–150.
[3] B. W. Boehm, Clark, Horowitz, Brown, Reifer, Chulani, R. Madachy, and B. Steece, *Software Cost Estimation with Cocomo II with Cdrom*, 1st ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2000.
[4] L. H. Putnam and W. Myers, *Five Core Metrics: Intelligence Behind Successful Software Management*. New York, NY, USA: Dorset House Publishing Co., Inc., 2003.
[5] V. Nguyen, S. Deeds-rubin, T. Tan, and B. Boehm, “A sloc counting standard,” in *COCOMO II Forum 2007*, 2007.
[6] D. D. Galorath and M. W. Evans, *Software Sizing, Estimation, and Risk Management*. Boston, MA, USA: Auerbach Publications, 2006.
[7] C. for review, “Censored for review,” *Censored for review*, vol. Under review, 2017.
[8] L. van der Maaten and G. Hinton, “Visualizing high-dimensional data using t-sne,” *Journal of Machine Learning Research*, vol. 9: 2579–2605, Nov 2008.
[9] M. S. S. Basha and D. Ponnurangam, “Analysis of empirical software effort estimation models,” *CoRR*, vol. abs/1004.1239, 2010. [Online]. Available: http://arxiv.org/abs/1004.1239
[10] D. Toka, “Accuracy of contemporary parametric software estimation models: A comparative analysis,” in *Proceeding of the 39th Euromicro Conference Series on Software Engineering and Advanced Applications*, Santander, Spain, 2013, pp. 313–316.
[11] A. Tharwon, “A literature survey on the accuracy of software effort estimation models,” in *Proceedings of the International MultiConference of Engineers and Computer Scientists 2016*, vol. II, 2016.
[12] W. L. Du, D. Ho, and L. F. Capretz, “A neuro-fuzzy model with SEER-SEM for software effort estimation,” *CoRR*, vol. abs/1508.00032, 2015. [Online]. Available: http://arxiv.org/abs/1508.00032
[13] W. Han, L. Jiang, T. Lu, and X. Zhang, “Comparison of machine learning algorithms for software project time prediction,” *International Journal of Multimedia and Ubiquitous Engineering*, vol. 10, no. 9, pp. 1–8, 2015. [Online]. Available: http://dx.doi.org/10.14257/ijmue.2015.10.9.01
[14] C. van Koten and A. R. Gray, “An application of bayesian network for predicting object-oriented software maintainability,” *Inf. Softw. Technol.*, vol. 48, no. 1, pp. 59–67, Jan. 2006. [Online]. Available: http://dx.doi.org/10.1016/j.infsof.2005.03.002
[15] J. Wen, S. Li, Z. Lin, Y. Hu, and C. Huang, “Systematic literature review of machine learning based software development effort estimation models,” *Inf. Softw. Technol.*, vol. 54, no. 1, pp. 41–59, Jan. 2012. [Online]. Available: http://dx.doi.org/10.1016/j.infsof.2011.09.002
[16] A. V. Sapre, “Feasibility of automated estimation of software development effort in agile environments,” Master’s thesis, The Ohio State University, 2012.
[17] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, “Distributed representations of words and phrases and their compositionality,” *CoRR*, vol. abs/1310.4546, 2013. [Online]. Available: http://arxiv.org/abs/1310.4546
[18] Q. V. Le and T. Mikolov, “Distributed representations of sentences and documents,” *CoRR*, vol. abs/1405.4053, 2014. [Online]. Available: http://arxiv.org/abs/1405.4053
[19] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
[20] T. M. Mitchell, *Machine learning*. McGraw-Hill, Inc. New York, USA, 1997.

[21] J. Bergstra and Y. Bengio, “Random search for hyper-parameter optimization,” *J. Mach. Learn. Res.*, vol. 13, pp. 281–305, Feb. 2012. [Online]. Available: http://dl.acm.org/citation.cfm?id=2188385.2188395

[22] R. Řehůřek and P. Sojka, “Software Framework for Topic Modelling with Large Corpora,” in *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. Valletta, Malta: ELRA, May 2010, pp. 45–50, http://is.muni.cz/publication/884893/en.

[23] M. Usman, E. Mendes, F. Weidt, and R. Britto, “Effort estimation in agile software development: A systematic literature review,” in *Proceedings of the 10th International Conference on Predictive Models in Software Engineering*, ser. PROMISE ’14. New York, NY, USA: ACM, 2014, pp. 82–91. [Online]. Available: http://doi.acm.org/10.1145/2639490.2639503

[24] D. B. Jovan Popović, “A comparative evaluation of effort estimation methods in the software life cycle,” *Computer Science and Information Systems*, no. 21, pp. 455–484, 2012. [Online]. Available: http://eudml.org/doc/253105