University of Wollongong

Research Online

Faculty of Engineering and Information Sciences - Papers: Part B

Faculty of Engineering and Information Sciences

2019

A deep learning model for estimating story points

Morakot Choetkiertikul
University of Wollongong, mc650@uowmail.edu.au

Hoa K. Dam
University of Wollongong, hoa@uow.edu.au

Truyen Tran
Deakin University

Trang Pham
Deakin University

Aditya K. Ghose
University of Wollongong, aditya@uow.edu.au

See next page for additional authors

Follow this and additional works at: https://ro.uow.edu.au/eispapers1

 Part of the Engineering Commons, and the Science and Technology Studies Commons

Research Online is the open access institutional repository for the University of Wollongong. For further information contact the UOW Library: research-pubs@uow.edu.au

A deep learning model for estimating story points

Abstract

Although there has been substantial research in software analytics for effort estimation in traditional software projects, little work has been done for estimation in agile projects, especially estimating the effort required for completing user stories or issues. Story points are the most common unit of measure used for estimating the effort involved in completing a user story or resolving an issue. In this paper, we propose a prediction model for estimating story points based on a novel combination of two powerful deep learning architectures: long short-term memory and recurrent highway network. Our prediction system is end-to-end trainable from raw input data to prediction outcomes without any manual feature engineering. We offer a comprehensive dataset for story points-based estimation that contains 23,313 issues from 16 open source projects. An empirical evaluation demonstrates that our approach consistently outperforms three common baselines (Random Guessing, Mean, and Median methods) and six alternatives (e.g. using Doc2Vec and Random Forests) in Mean Absolute Error, Median Absolute Error, and the Standardized Accuracy.

Disciplines

Engineering | Science and Technology Studies

Publication Details

Choetkiertikul, M., Dam, H. Khanh., Tran, T., Pham, T., Ghose, A. & Menzies, T. (2019). A deep learning model for estimating story points. IEEE Transactions on Software Engineering, 45 (7), 637-656.

Authors

Morakot Choetkiertikul, Hoa K. Dam, Truyen Tran, Trang Pham, Aditya K. Ghose, and Tim Menzies

This journal article is available at Research Online: https://ro.uow.edu.au/eispapers1/2980

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2018.2792473, IEEE Transactions on Software Engineering

A deep learning model for estimating story points

Morakot Choetkiertikul, Hoa Khanh Dam, Truyen Tran, Trang Pham, Aditya Ghose, and Tim Menzies

Abstract—Although there has been substantial research in software analytics for effort estimation in traditional software projects, little work has been done for estimation in agile projects, especially estimating the effort required for completing user stories or issues. Story points are the most common unit of measure used for estimating the effort involved in completing a user story or resolving an issue. In this paper, we propose a prediction model for estimating story points based on a novel combination of two powerful deep learning architectures: long short-term memory and recurrent highway network. Our prediction system is *end-to-end* trainable from raw input data to prediction outcomes without any manual feature engineering. We offer a comprehensive dataset for story points-based estimation that contains 23,313 issues from 16 open source projects. An empirical evaluation demonstrates that our approach consistently outperforms three common baselines (Random Guessing, Mean, and Median methods) and six alternatives (e.g. using Doc2Vec and Random Forests) in Mean Absolute Error, Median Absolute Error, and the Standardized Accuracy.

Index Terms—software analytics, effort estimation, story point estimation, deep learning.

1 Introduction

Effort estimation is an important part of software project management, particularly for planning and monitoring a software project. Cost and schedule overruns have been a common risk in software projects. Mckinsey and the University of Oxford has conducted a study on 5,400 large scale IT projects, and found that on average large software projects run 66% over budget and 33% overtime [1]. A different study on 1,471 software projects [2] revealed similar findings: one in six software projects has a budget overrun of 200% and a schedule overrun of almost 70%. Activities involving effort estimation forms a critical part in planning and managing a software project to ensure it complete in time and within budget [3–5]. Effort estimates may be used by different stakeholders as input for developing project plans, scheduling iteration or release plans, budgeting, and costing [6]. Hence, incorrect estimates may have adverse impact on the project outcomes [3, 7–9]. Research in software effort estimation dates back several decades and they can generally be divided into model-based methods, expert-based methods, and hybrid methods which combine model-based and expert-based methods [10]. Model-based approaches leverages data from old projects to make predictions about new projects. Expert-based methods rely on human expertise to make such judgements. Most of the existing work (e.g. [11–22]) focus on the effort required for completing a whole project (as opposed to user stories or issues). These approaches estimate the effort required for developing a complete software system, relying on a set of features manually designed for characterizing a software project.

In modern agile development settings, software is developed through repeated cycles (iterative) and in smaller parts at a time (incremental), allowing for adaptation to changing requirements at any point during a project's life. A project has a number of *iterations* (e.g. *sprints* in Scrum [23]). An iteration is usually a short (usually 2–4 weeks) period in which the development team designs, implements, tests and delivers a distinct product increment, e.g. a working milestone version or a working release. Each iteration requires the completion of a number of user stories, which are a common way for agile teams to express user requirements. This is a shift from a model where all functionalities are delivered together (in a single delivery) to a model involving a series of incremental deliveries.

There is thus a need to focus on estimating the effort of completing a single user story at a time rather than the entire project. In fact, it has now become a common practice for agile teams to go through each user story and estimate the effort required for completing it. *Story points* are commonly used as a unit of effort measure for a user story [24]. Currently, most agile teams heavily rely on experts' subjective assessment (e.g. planning poker, analogy, and expert judgment) to arrive at an estimate. This may lead to inaccuracy and more importantly inconsistencies between estimates [25].

To facilitate research in effort estimation for agile development, we have developed a new dataset for story point effort estimation. This dataset contains 23,313 user stories or issues with ground truth story points. Note that ground-truth story points refer to the actual story points that were assigned to an issue by the team. We collected these issues from 16 large open source projects in 9 repositories namely Apache, Appcelerator, DuraSpace, Atlassian, Moodle, Lsstcorp, Mulesoft, Spring, and Talendforge. To the best of our

- *M. Choetkiertikul, H. Dam, and A. Ghose are with the School of Computing and Information Technology, Faculty of Engineering and Information Sciences, University of Wollongong, NSW, Australia, 2522. E-mail: {mc650, hoa, aditya}@uow.edu.au*
- *T. Tran and T. Pham are with the School of Information Technology, Deakin University, Victoria, Australia, 3216. E-mail: {truyen.tran, phtra}@deakin.edu.au*
- *T. Menzies is with North Carolina State University, USA. E-mail: tim.menzies@gmail.com*

0098-5589 (c) 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2018.2792473, IEEE Transactions on Software Engineering

knowledge, this is the largest dataset (in terms of number of data points) for story point estimation where the focus is at the issue/user story level rather than at the project level as in traditional effort estimation datasets.

We also propose a prediction model which supports a team by recommending a story-point estimate for a given user story. Our model learns from the team's previous story point estimates to predict the size of new issues. This prediction system will be used in conjunction with (instead of a replacement for) existing estimation techniques practiced by the team. It can be used in an completely automated manner, i.e. the team will use the story points given by the prediction system. Alternatively, it could be used as a decision support system and takes part in the estimation process. This is similar to the notions of combination-based effort estimation in which estimates come from different sources, e.g. a combination of expert and formal model-based estimates [10]. The key novelty of our approach resides in the combination of two powerful *deep learning* architectures: long short-term memory (LSTM) and recurrent highway network (RHN). LSTM allows us to model the long-term context in the textual description of an issue, while RHN provides us with a deep representation of that model. We named this approach as Deep learning model for Story point Estimation (Deep-SE).

Our Deep-SE model is a fully *end-to-end* system where raw data signals (i.e. words) are passed from input nodes up to the final output node for estimating story points, and the prediction errors are propagated from the output node all the way back to the word layer. Deep-SE automatically learns semantic features which represent the meaning of user stories or issue reports, thus liberating the users from manually designing and extracting features. Feature engineering usually relies on domain experts who use their specific knowledge of the data to create features for machine learners to work. For example, the performance of most of existing defect prediction models heavily relies on the careful designing of good features (e.g. size of code, number of dependencies, cyclomatic complexity, and code churn metrics) which can discriminate between defective and non-defective code [26]. Coming up with good features is difficult, time-consuming, and requires domain-specific knowledge, and hence poses a major challenge. In many situations, manually designed features normally do not generalize well: features that work well in a certain software project may not perform well in other projects [27]. Bag-of-Words (BoW) is a traditional technique to "engineer" features representing textual data like issue description. However, the BoW approach has two major weaknesses: it ignores the semantics of words, e.g. fails to recognize the semantic relations between "she" and "he", and it ignore the sequential nature of text. In our approach, features are automatically learned, thus obviating the need for designing them manually.

Although our Deep-SE is a deep model of multiple layers, it is recurrent and thus model parameters are shared across layers. Hence, the number of parameters does not grow with the depth and consequently avoid overfitting. We also employ a number of common techniques such as dropout and early stopping combat overfitting. Our approach consistently outperforms three common baseline estimators: Random Guessing, Mean, and Median methods and six alternatives (e.g. using Doc2Vec and Random Forests) in Mean Absolute Error, Median Absolute Error, and the Standardized Accuracy. These claims have also been tested using a non-parametric Wilcoxon test and Vargha and Delaney's statistic to demonstrate the statistical significance and the effect size.

The remainder of this paper is organized as follows. Section 2 provides a background of the story point estimation and deep neural networks. We then present the Deep-SE model and explain how it can be trained in Section 3 and Section 4 respectively. Section 5 reports on the experimental evaluation of our approach. Related work is discussed in Section 6 before we conclude and outline future work in Section 7.

2 Background

2.1 Story point estimation

When a team estimates with story points, it assigns a point value (i.e. story points) to each user story. A story point estimate reflects the *relative* amount of effort involved in resolving or completing the user story: a user story that is assigned two story points should take twice as much effort as a user story assigned one story point. Many projects have now adopted this story point estimation approach [25]. Projects that use issue tracking systems (e.g. JIRA [28]) record their user stories as *issues*. Figure 1 shows an example of issue `XD-2970` in the Spring XD project [29] which is recorded in JIRA. An issue typically has a title (e.g. "Standardize XD logging to align with Spring Boot") and description. Projects that use JIRA Agile also record story points. For example, the issue in Figure 1 has 8 story points.

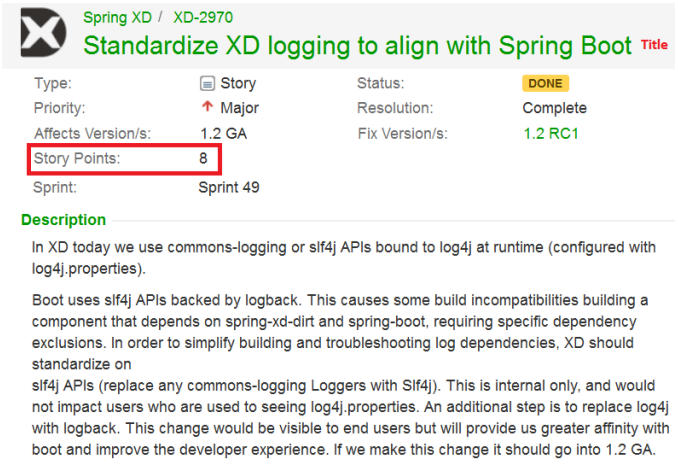


Fig. 1. An example of an issue with estimated story points

Story points are usually estimated by the whole team within a project. For example, the widely-used Planning Poker [30] method suggests that each team member provides an estimate and a consensus estimate is reached after a few rounds of discussion and (re-)estimation. This practice is different from traditional approaches (e.g. function points) in several aspects. Both story points and function points reflect an effort for resolving an issue. However, function points can be determined by an external estimator based on

0098-5589 (c) 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2018.2792473, IEEE Transactions on Software Engineering

a standard set of rules (e.g. counting inputs, outputs, and inquiries) that can be applied consistently by any trained practitioner. On the other hand, story points are developed by a specific team based on the team's cumulative knowledge and biases, and thus may not be useful outside the team (e.g. in comparing performance across teams). Since story points represent the effort required for completing a user story, an estimate should cover different factors which can affect the effort. These factors include how much work needed to be done, the complexity of the work, and any uncertainty involving in the work [24].

In agile development, user stories or issues are commonly viewed as the first-class entity of a project since they describe what has to be built in the software project, forming the basis for design, implementation and testing. Story point sizes are used for measuring a team's progress rate, prioritizing user stories, planning and scheduling for future iterations and releases, and even costing and allocating resources. Story points are also the basis for other effort-related estimation. For example, in our recent work [31], they are used for predicting delivery capability for an ongoing iteration. Specifically, we predict the amount of work delivered at the end of an iteration, relative to the amount of work which the team has originally committed to. The amount of work done in an iteration is then quantified in terms of story points from the issues completed within that iteration. To enable such a prediction, we have taken into account both the information of an iteration and user stories or issues involving in the iteration. Interaction between user stories and between user stories and resources are captured through extracting information related to the dependencies between user stories and the assignment of user stories to developers.

Velocity is the sum of the story-point estimates of the issues that the team resolved during an iteration. For example, if the team resolves four stories each estimated at three story points, their velocity is twelve. Velocity is used for planning and predicting when a software (or a release) should be completed. For example, if the team estimates the next release to include 100 story points and the team's current velocity is 20 points per 2-week iteration, then it would take 5 iterations (or 10 weeks) to complete the project. Hence, it is important that the team is *consistent* in their story point estimates to avoid reducing the predictability in planning and managing their project. A machine learner can help the team maintain this consistency, especially in coping with increasingly large numbers of issues. It does so by learning insight from past issues and estimations to make future estimations.

2.2 Long Short Term Memory

Long Short-Term Memory (LSTM) [32, 33] is a special variant of recurrent neural networks [34]. While a feedforward neural network maps an input vector into an output vector, an LSTM network uses a loop in a network that allows information to persist and it can map a sequence into a sequence (see Figure 2). Let $w_1, ..., w_n$ be the input sequence (e.g. words in a sentence) and $y_1, ..., y_n$ be the sequence of corresponding labels (e.g. the next words). At time step t, an LSTM unit reads the input w_t, the previous hidden state h_{t-1}, and the previous memory c_{t-1} in order to compute the hidden state h_t. The hidden state is used to produce an output at each step t. For example, the output of predicting the next word k in a sentence would be a vector of probabilities across our vocabulary, i.e. $softmax(V_k h_t)$ where V_k is a row in the output parameter matrix W_{out}.

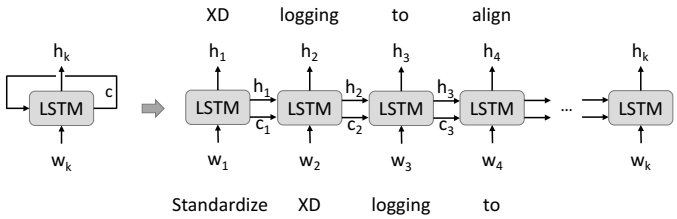


Fig. 2. An LSTM network

The most important element of LSTM is a short-term *memory cell* – a vector that stores accumulated information over time. The information stored in the memory is refreshed at each time step through partially forgetting old, irrelevant information and accepting fresh new input. An LSTM unit uses the forget gate $\boldsymbol{f}_t$ to control how much information from the memory of previous context (i.e. c_{t-1}) should be removed from the memory cell. The forget gate looks at the the previous output state h_{t-1} and the current word w_t, and outputs a number between 0 and 1. A value of 1 indicates that all the past memory is preserved, while a value of 0 means "completely forget everything". The next step is updating the memory with new information obtained from the current word w_t. The input gate $\boldsymbol{i}_t$ is used to control which new information will be stored in the memory. Information stored in the memory cell will be used to produce an output h_t. The output gate $\boldsymbol{o}_t$ looks at the current code token w_t and the previous hidden state h_{t-1}, and determines which parts of the memory should be output.

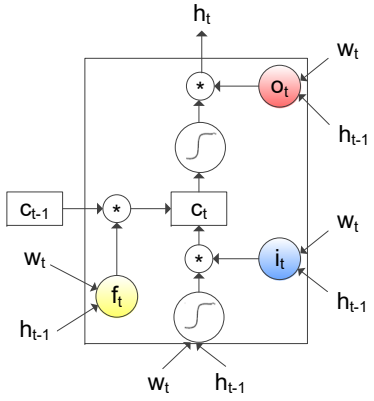


Fig. 3. The internal structure of an LSTM unit

This mechanism allows LSTM to effectively learn long-term dependencies in text. Consider trying to predict the last word in the following text extracted from the description of issue `XD-2970` in Figure 1: *"Boot uses slf4j APIs backed by logback. This causes some build incompatibilities .... An additional step is to replace log4j with __."*. Recent information suggests that the next word is probably the name of a logging library,

0098-5589 (c) 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2018.2792473, IEEE Transactions on Software Engineering

but if we want to narrow down to a specific library, we need to remember that "logback" and "log4j" are logging libraries from the earlier text. There could be a big gap between relevant information and the point where it is needed, but LSTM is capable to learn to connect the information. In fact, LSTM has demonstrated ground-breaking results in many applications such as language models [35], speech recognition [36] and video analysis [37].

The reading of the new input, writing of the output, and the forgetting (i.e. all those gates) are all *learnable*. As an recurrent network, LSTM network shares the same parameters across all steps since the same task is performed at each step, just with different inputs. Thus, comparing to traditional feedforward networks, using an LSTM network significantly reduces the total number of parameters which we need to learn. An LSTM model is trained using many input sequences with known actual output sequences. Learning is done by minimizing the error between the actual output and the predicted output by adjusting the model parameters. Learning involves computing the gradient of $L(\boldsymbol{\theta})$ during the backpropagation phase, and parameters are updated using a stochastic gradient descent. It means that parameters are updated after seeing only a small random subset of sequences. We refer the readers to the seminal paper [32] for more details about LSTM.

3 Approach

Our overall research goal is to build a prediction system that takes as input the title and description of an issue and produces a story-point estimate for the issue. Title and description are required information for any issue tracking system. Hence, our prediction system is applicable to a wide range of issue tracking systems, and can be used at any time, even when an issue is created.

We combine the title and description of an issue report into a single text document where the title is followed by the description. Our approach computes vector representations for these documents. These representations are then used as features to predict the story points of each issue. It is important to note that these features are *automatically* learned from raw text, hence removing us from manually engineering the features.

Figure 4 shows the Deep learning model for Story point Estimation (Deep-SE) that we have designed for the story point prediction system. It is composed of four components arranged sequentially: (i) word embedding, (ii) document representation using Long Short-Term Memory (LSTM) [32], (iii) deep representation using Recurrent Highway Net (RHWN) [38]; and (iv) differentiable regression. Given a document which consists of a sequence of words $s = (w_1, w_2, ..., w_n)$, e.g. the word sequence *(Standardize, XD, logging, to, align, with, ...)* in the title and description of issue `XD-2970` in Figure 1.

We model a document's semantics based on the principle of compositionality: the meaning of a document is determined by the meanings of its constituents (e.g. words) and the rules used to combine them (e.g. one word followed by another). Hence, our approach models document representation in two stages. It first converts each word in a document into a fixed-length vector (i.e. word embedding).

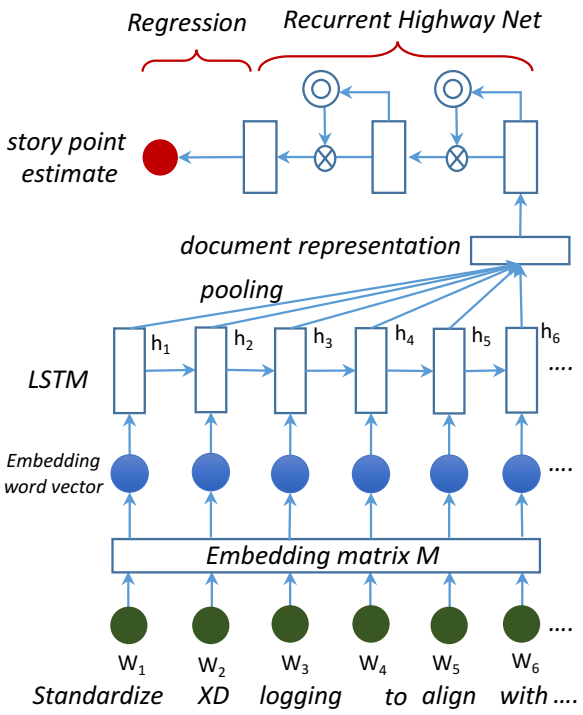


Fig. 4. Deep learning model for Story point Estimation (Deep-SE). The input layer (bottom) is a sequence of words (represented as filled circles). Words are first embedded into a continuous space, then fed into the LSTM layer. The LSTM outputs a sequence of state vectors, which are then pooled to form a document-level vector. This global vector is then fed into a Recurrent Highway Net for multiple transformations (See Eq. (1) for detail). Finally, a regressor predicts an outcome (story-point).

These word vectors then serve as an input sequence to the Long Short-Term Memory (LSTM) layer which computes a vector representation for the whole document.

After that, the document vector is fed into the Recurrent Highway Network (RHWN), which transforms the document vector multiple times, before outputting a final vector which represents the text. The vector serves as input for the *regressor* which predicts the output story-point. While many existing regressors can be employed, we are mainly interested in regressors that are *differentiable* with respect to the training signals and the input vector. In our implementation, we use the simple *linear regression* that outputs the story-point estimate.

Our entire system is trainable from *end-to-end*: (a) data signals are passed from the words in issue reports to the final output node; and (b) the prediction error is propagated from the output node all the way back to the word layer.

3.1 Word embedding

We represent each word as a low dimensional, continuous and real-valued vector, also known as *word embedding*. Here we maintain a look-up table, which is a word embedding matrix $\mathcal{M} \in \mathbb{R}^{d \times |V|}$ where d is the dimension of word vector and $|V|$ is vocabulary size. These word vectors are pre-trained from corpora of issue reports, which will be described in details in Section 4.1.

0098-5589 (c) 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2018.2792473, IEEE Transactions on Software Engineering

3.2 Document representation using LSTM

Since an issue document consists of a sequence of words, we model the document by accumulating information from the start to the end of the sequence. A powerful accumulator is a Recurrent Neural Network (RNN) [34], which can be seen as multiple copies of the same single-hidden-layer network, each passing information to a successor. Thus, recurrent networks allow information to be accumulated. While RNNs are theoretically powerful, they are difficult to train for long sequences [34], which are often seen in issue reports (e.g. see the description of issue `XD-2970` in Figure 1). Hence, our approach employs Long Short-Term Memory (LSTM), a special variant of RNN (see Section 2 for more details of how LSTM work).

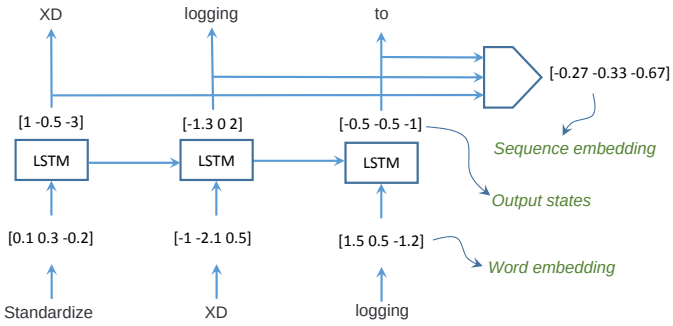


Fig. 5. An example of how a vector representation is obtained for issue reports

After the vector output state has been computed for every word in the input sequence, the next step is aggregating those vectors into a single vector representing the whole document. The aggregation operation is known as pooling. There are multiple ways to perform pooling, but the main requirement is that pooling must be length invariant. In other words, pooling is not sensitive to variable length of the document. For example, the simplest statistical pooling method is mean-pooling where we take the sum of the state vectors and divide it by the number of vectors. Other pooling methods are such as max pooling (e.g. choose the maximum value in each dimension), min pooling and sum pooling. From our experience in other settings, a simple but often effective pooling method is averaging, which we also employed here [39].

3.3 Deep representation using Recurrent Highway Network

Given that vector representation of an issue report has been extracted by the LSTM layer, we can use a differentiable regressor for immediate prediction. However, this may be sub-optimal since the network is rather shallow. Deep neural networks have become a popular method with many ground-breaking successes in vision [40], speech recognition [41] and NLP [42, 43]. Deep nets represent complex data more efficiently than shallow ones [44]. Deep models can be expressive while staying compact, as theoretically analysed by recent work [45–49]. This have been empirically validated in recent record-breaking results in vision, speech recognition and machine translation. However, learning standard feedforward networks with many hidden layers is notoriously difficult due to two main problems: (i) the number of parameters grows with the number of layers, leading to overfitting; and (ii) stacking many non-linear functions makes it difficult for the information and the gradients to pass through.

To address these problems, we designed a deep representation that performs multiple non-linear transformations using the idea from Highway Networks. Highway Nets are the latest idea that enables efficient learning through those many non-linear layers [50]. A Highway Net is a special type of feedforward neural networks with a modification to the transformation taking place at a hidden unit to let information from lower layers pass *linearly* through. Specifically, the hidden state at layer l is defined as:

$$\boldsymbol{h}_{l+1} = \boldsymbol{\alpha}_l * \boldsymbol{h}_l + (\mathbf{1} - \boldsymbol{\alpha}_l) * \sigma_l(\boldsymbol{h}_l) \quad (1)$$

where σ_l is a non-linear transform (e.g., a logistic or a tanh) and $\boldsymbol{\alpha}_l = \text{logit}(\boldsymbol{h}_l)$ is a linear logistic transform of $\boldsymbol{h}_l$. Here $\boldsymbol{\alpha}_l$ plays the role of a *highway gate* that lets information passing from layer l to layer $l+1$ without loss of information. For example, $\boldsymbol{\alpha}_l \to \mathbf{1}$ enables simple copying.

We need to learn a mapping from the raw words in an issue description to the story points. A deep feedforward neural network like Highway Net effectively breaks the mapping into a series of nested simple mappings, each described by a different layer of the network. The first layer provides a (rough) estimate, and subsequent layers iteratively refine that estimate. As the number of layers increase, further refinement can be achieved. Comparing to traditional feedforward networks, the special gating scheme in Highway Net is highly effective in letting the information and the gradients to pass through while stacking many non-linear functions. In fact, earlier work has demonstrated that Highway Net can have up to a thousand layers [50], while traditional deep neural nets cannot go beyond several layers [51].

We have also modified the standard Highway Network by *sharing* parameters between layers, i.e. all the hidden layers having the same hidden units. In other words, all the hidden layers to have the same hidden units. This is similar to the notion of a recurrent network, and thus we called it a Recurrent Highway Network. Our previous work [38] has demonstrated the effectiveness of this approach in pattern recognition. This key novelty allows us to create a very compact version of Recurrent Highway Network with only one set of parameters in $\boldsymbol{\alpha}_l$ and σ_l. This clearly produces a great advantage of avoiding overfitting. We note that the number of layers here refers to the number of hidden layers of a Recurrent Highway Network, not the number of LSTM layers. The number of LSTM layers is the same as the number of words in an issue's description.

3.4 Regression

At the top-layer of Deep-SE, we employ linear activation function in a feedforward neural network as the final regressor (see Figure 4) to produce a story-point estimate. This function can be defined as follows.

$$y = b_0 + \sum_{i=1}^{n} b_i x_i \quad (2)$$

0098-5589 (c) 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2018.2792473, IEEE Transactions on Software Engineering

where y is the output story point , x_i is an input signal from RHWN layer, b_i is trained coefficient (weight), and n is the size of embedding dimension.

4 Model Training

4.1 Pre-training

Pre-training is a way to come up with a good parameter initialization *without* using the labels (i.e. ground-truth story points). We pre-train the lower layers of Deep-SE (i.e. embedding and LSTM), which operate at the word level. Pre-training is effective when the labels are not abundant. During pre-training, we do *not* use the ground-truth story points, but instead leverage two sources of information: the strong predictiveness of natural language, and availability of free texts without labels (e.g. issue reports without story points). The first source comes from the property of languages that the next word can be predicted using previous words, thanks to grammars and common expressions. Thus, at each time step t, we can predict the next word w_{t+1} using the state $\boldsymbol{h}_t$, using the softmax function:

$$P\left(w_{t+1}=k \mid \boldsymbol{w}_{1:t}\right)=\frac{\exp\left(U_k \boldsymbol{h}_t\right)}{\sum_{k'} \exp\left(U_{k'} \boldsymbol{h}_t\right)} \tag{3}$$

where U_k is a free parameter. Essentially we are building a language model, i.e., $P(s)=P\left(\boldsymbol{w}_{1:n}\right)$, which can be factorized using the chain-rule as: $P\left(w_1\right) \prod_{t=2}^{n} P\left(w_{t+1} \mid \boldsymbol{w}_{1:t}\right)$.

We note that the probability of the first word $P(w_1)$ in a sequence is the number of sequences in the corpus which has that word w_1 starting first. At step t, h_t is computed by feeding h_{t-1} and w_t to the LSTM unit (see Figure 2). Since w_t is a word embedding vector, Eq. (3) indirectly refers to the embedding matrix .

The language model can be learned by optimizing the log-loss $-\log P(s)$. However, the main bottleneck is computational: Equation (3) costs $|V|$ time to evaluate where $|V|$ is the vocabulary size, which can be hundreds of thousands for a big corpus. For that reason, we implemented an approximate but very fast alternative based on Noise-Contrastive Estimation [52], which reduces the time to $M \ll |V|$, where M can be as small as 100. We also run the pre-training multiple times against a validation set to choose the best model. We use *perplexity*, a common intrinsic evaluation metric based on the log-loss, as a criterion for choosing the best model and early stopping. A smaller perplexity implies a better language model. The word embedding matrix $\mathcal{M} \in \mathbb{R}^{d \times |V|}$ (which is first randomly initialized) and the initialization for LSTM parameters are learned through this pre-training process.

4.2 Training Deep-SE

We have implemented the Deep-SE model in Python using Theano [53]. To simplify our model, we set the size of the memory cell in an LSTM unit and the size of a recurrent layer in RHWN to be the same as the embedding size. We tuned some important hyper-parameters (e.g. embedding size and the number of hidden layers) by conducting experiments with different values, while for some other hyper-parameters, we used the default values. This will be discussed in more details in the evaluation section.

Recall that the entire network can be reduced to a parameterized function defined, which maps sequences of raw words (in issue reports) to story points. Let θ be the set of all parameters in the model. We define a loss function $L(\theta)$ that measures the quality of a particular set of parameters based on the difference between the predicted story points and the ground truth story points in the training data. A setting of the parameters θ that produces a prediction for an issue in the training data consistent with its ground truth story points would have a very low loss L. Hence, learning is achieved through the optimization process of finding the set of parameters θ that minimizes the loss function.

Since every component in the model is differentiable, we use the popular stochastic gradient descent to perform optimization: through backpropagation, the model parameters θ are updated in the opposite direction of the gradient of the loss function $L(\theta)$. In this search, a learning rate η is used to control how large of a step we take to reach a (local) minimum. We use RMSprop, an adaptive stochastic gradient method (unpublished note by Geoffrey Hinton), which is known to work best for recurrent models. We tuned RMSprop by partitioning the data into mutually exclusive training, validation, and test sets and running the training multiple times. Specifically, the training set is used to learn a useful model. After each training epoch, the learned model was evaluated on the validation set and its performance was used to assess against hyperparameters (e.g. learning rate in gradient searches). Note that the validation set was *not* used to learn any of the model's parameters. The best performing model in the validation set was chosen to be evaluated on the test set. We also employed the early stopping strategy (see Section 5.4), i.e. monitoring the model's performance during the validation phase and stopping when the performance got worse. If the log-loss does not improve for ten consecutive runs, we than terminate the training.

To prevent overfitting in our neural network, we have implemented an effective solution called *dropout* in our model [54], where the elements of input and output states are randomly set to zeros during training. During testing, parameter averaging is used. In effect, dropout implicitly trains many models in parallel, and all of them share the same parameter set. The final model parameters represent the average of the parameters across these models. Typically, the dropout rate is set at 0.5.

An important step prior to optimization is parameter initialization. Typically the parameters are initialized randomly, but our experience shows that a good initialization (through pre-training of embedding and LSTM layers) helps learning converge faster to good solutions.

5 Evaluation

The empirical evaluation we carried out aimed to answer the following research questions:

- **RQ1. Sanity Check**: *Is the proposed approach suitable for estimating story points?*
  This sanity check requires us to compare our Deep-SE prediction model with the three common baseline benchmarks used in the context of effort estimation:

0098-5589 (c) 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2018.2792473, IEEE Transactions on Software Engineering

Random Guessing, Mean Effort, and Median Effort. Random guessing is a naive benchmark used to assess if an estimation model is useful [55]. Random guessing performs random sampling (with equal probability) over the set of issues with known story points, chooses randomly one issue from the sample, and uses the story point value of that issue as the estimate of the target issue. Random guessing does not use any information associated with the target issue. Thus any useful estimation model should outperform random guessing. Mean and Median Effort estimations are commonly used as baseline benchmarks for effort estimation [19]. They use the mean or median story points of the past issues to estimate the story points of the target issue. Note that the samples used for all the naive baselines (i.e. Random Guessing, Mean Effort, and Median Effort) were from the training set.

- **RQ2. Benefits of deep representation**: *Does the use of Recurrent Highway Nets provide more accurate story point estimates than using a traditional regression technique?*
  To answer this question, we replaced the Recurrent Highway Net component with a regressor for immediate prediction. Here, we compare our approach against four common regressors: Random Forests (RF), Support Vector Machine (SVM), Automatically Transformed Linear Model (ATLM), and Linear Regression (LR). We choose RF over other baselines since ensemble methods like RF, which combine the estimates from multiple estimators, are an effective method for effort estimation [20]. RF achieves a significant improvement over the decision tree approach by generating many classification and regression trees, each of which is built on a random resampling of the data, with a random subset of variables at each node split. Tree predictions are then aggregated through averaging. We used the issues in the validation set to fine-tune parameters (i.e. the number of tress, the maximum depth of the tree, and The minimum number of samples). For SVM, it has been widely use in software analytics (e.g. defect prediction) and document classification (e.g. sentiment analysis) [56]. SVM is known as Support Vector Regression (SVR) for regression problems. We also used the issues in the validation set to find the kernel type (e.g. linear, polynomial) for testing. We used the Automatically Transformed Linear Model (ATLM) [57] recently proposed as the baseline model for software effort estimation. Although ATLM is simple and requires no parameter tuning, it performs well over a range of various project types in the traditional effort estimation [57]. Since LR is the top layer of our approach, we also used LR as the immediate regressor after LSTM layers to assess whether RHWN improves the predictive performance. We then compare the performance of these alternatives, namely LSTM+RF, LSTM+SVM, LSTM+ATLM, and LSTM+LR against our Deep-SE model.
- **RQ3. Benefits of LSTM document representation**: *Does the use of LSTM for modeling issue reports provide more accurate results than the traditional Doc2Vec and Bag-of-Words (BoW) approach?*
  The most popular text representation is Bag-of-Words (BoW) [58], where a text is represented as a vector of word counts. For example, the title and description of issue `XD-2970` in Figure 1 would be converted into a sparse binary vector of vocabulary size, whose elements are mostly zeros, except for those at the positions designated to "standardize", "XD", "logging" and so on. However, BoW has two major weaknesses: they lose the sequence of the words and they also ignore semantics of the words. For example, "Python", "Java", and "logging " are equally distant, while semantically "Python" should be closer to "Java" than "logging". To address this issue, Doc2vec [59] (i.e. alternatively known as paragraph2vec) is an unsupervised algorithm that learns fixed-length feature representations from texts (e.g. title and description of issues). Each document is represented in a dense vector which is trained to predict next words in the document.
  Both BoW and Doc2vec representations however effectively destroys the sequential nature of text. This question aims to explore whether LSTM with its capability of modeling this sequential structure would improve the story point estimation. To answer this question, we feed three different feature vectors: one learned by LSTM and the other two derived from BoW technique and Doc2vec to the same Random Forrests regressor, and compare the predictive performance of the former (i.e. LSTM+RF) against that of the latter (i.e. BoW+RF and Doc2vec+RF).We used Gensim[1], a well-known implementation for Doc2vec in our experiments.
- **RQ4. Cross-project estimation**: *Is the proposed approach suitable for cross-project estimation?*
  Story point estimation in new projects is often difficult due to lack of training data. One common technique to address this issue is training a model using data from a (source) project and applying it to the new (target) project. Since our approach requires only the title and description of issues in the source and target projects, it is readily applicable to both within-project estimation and cross-project estimation. In practice, story point estimation is however known to be specific to teams and projects. Hence, this question aims to investigate whether our approach is suitable for cross-project estimation. We have implemented Analogy-based estimation called ABE0, which were proposed in previous work [60–63] for cross-project estimation, and used it as a benchmark. The ABE0 estimation bases on the distances between individual issues. Specifically, the story point of issues in the target project is the mean of story points of k-nearest issues from the source project. We used the Euclidean distance as a distance measure, Bag-of-Words of the title and the description as the features of an issue, and $k = 3$.
- **RQ5. Normalizing/adjusting story points**: *Does our*

1. https://radimrehurek.com/gensim/models/doc2vec.html

0098-5589 (c) 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2018.2792473, IEEE Transactions on Software Engineering

approach still perform well with normalized/adjusted story points?
We have ran our experiments again using the new labels (i.e. the normalized story points) for addressing the concern that whether our approach still performs well on those adjusted ground-truths. We adjusted the story points of each issue using a range of information, including the number of days from creation to resolved time, the development time, the number of comments, the number of users who commented on the issue, the number of times that an issue had their attributes changed, the number of users who changed the issue's attributes, the number of issue links, the number of affect versions, and the number of fix versions. These information reflect the actual effort and we thus refer to them as effort indicators. The values of these indicators were extracted after the issue was completed. The normalized story point ($SP_{normalized}$) is then computed as the following:

$$SP_{normalized} = (0.5)SP_{original} + (0.5)SP_{nearest}$$

where $SP_{orginal}$ is the original story point, and $SP_{nearest}$ is the mean of story points from 10 nearest issues based on their actual effort indicators. Note that we use K-Nearest Neighbour (KNN) to find the nearest issues and the Euclidean metric to measure the distance. We ran the experiment on the new labels (i.e $SP_{normalized}$) using our proposed approach against all other baseline benchmark methods.

- **RQ6. Compare against the existing approach:** *How does our approach perform against existing approaches in story point estimation?*
Recently, Porru et. al. [64] also proposed an estimation model for story points. Their approach uses the type of an issue, the component(s) assigned to it, and the TF-IDF derived from its summary and description as features representing the issue. They also performed univariate feature selection to choose a subset of features for building a classifier. By contrast, our approach automatically learns semantic features which represent the actual meaning of the issue's report, thus potentially providing more accurate estimates. To answer this research question, we ran Deep-SE on the dataset used in Porru et. al, re-implemented their approach, and performed a comparison on the results produced by the two approaches.

5.1 Story point datasets

To collect data for our dataset, we looked for issues that were estimated with story points. JIRA is one of the few widely-used issue tracking systems that support agile development (and thus story point estimation) with its JIRA Agile plugin. Hence, we selected a diverse collection of nine major open source repositories that use the JIRA issue tracking system: Apache, Appcelerator, DuraSpace, Atlassian, Moodle, Lsstcorp, MuleSoft, Spring, and Talendforge. We then used the Representational State Transfer (REST) API provided by JIRA to query and collected those issue reports. We collected all the issues which were assigned a story point measure from the nine open source repositories up until August 8, 2016. We then extracted the story point, title and description from the collected issue reports. Each repository contains a number of projects, and we chose to include in our dataset only projects that had more than 300 issues with story points. Issues that were assigned a story point of zero (e.g., a non-reproducible bug), as well as issues with a negative, or unrealistically large story point (e.g. greater than 100) were filtered out. Ultimately, about 2.66% of the collected issues were filtered out in this fashion. In total, our dataset has 23,313 issues with story points from 16 different projects: Apache Mesos (ME), Apache Usergrid (UG), Appcelerator Studio (AS), Aptana Studio (AP), Titanum SDK/CLI (TI), DuraCloud (DC), Bamboo (BB), Clover (CV), JIRA Software (JI), Moodle (MD), Data Management (DM), Mule (MU), Mule Studio (MS), Spring XD (XD), Talend Data Quality (TD), and Talend ESB (TE). Table 1 summarizes the descriptive statistics of all the projects in terms of the minimum, maximum, mean, median, mode, variance, and standard deviations of story points assigned used and the average length of the title and description of issues in each project. These sixteen projects bring diversity to our dataset in terms of both application domains and project's characteristics. Specifically, they are different in the following aspects: number of observation (from 352 to 4,667 issues), technical characteristics (different programming languages and different application domains), sizes (from 88 KLOC to 18 millions LOC), and team characteristics (different team structures and participants from different regions).

Since story points rate the relative effort of work between user stories, they are usually measured on a certain scale (e.g. 1, 2, 4, 8, etc.) to facilitate comparison (e.g. a user story is double the effort of the other) [25]. The story points used in planning poker typically follow a Fibonacci scale, i.e. 1, 2, 3, 5, 8, 13, 21, and so on [24]. Among the projects we studied, only seven of them (i.e. Usergrid, Talend ESB, Talend Data Quality, Mule Studio, Mule, Appcelerator Studio, and Aptana Studio followed the Fibonacci scale, while the other nine projects did not use any scale. When our prediction system gives an estimate, we did not round it to the nearest story point value on the Fibonacci scale. An alternative approach (for those project which follow a Fibonacci scale) is treating this as a classification problem: each value on the Fibonacci scale represents a class. The limitations of this approach is that the number of classes must be pre-determined and that it is not applicable to projects that do not follow this scale. We however note that the Fibonacci scale is only a guidance for estimating story points. In practice, teams may follow other common scales, define their own scales or may not follow any scale at all. Our approach does not rely on these specific scales, thus making it applicable to a wider range of projects. It predicts a scalar value (regression) rather than a class (classification).

5.2 Experimental setting

We performed experiments on the sixteen projects in our dataset – see Table 1 for their details. To mimic a real deployment scenario that prediction on a current issue is made by using knowledge from estimations of the past issues, the issues in each project were split into training set

0098-5589 (c) 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2018.2792473, IEEE Transactions on Software Engineering

TABLE 1
Descriptive statistics of our story point dataset

Repo.	Project	Abb.	# issues	min SP	max SP	mean SP	median SP	mode SP	var SP	std SP	mean TD length	LOC
Apache	Mesos	ME	1,680	1	40	3.09	3	3	5.87	2.42	181.12	247,542+
	Usergrid	UG	482	1	8	2.85	3	3	1.97	1.40	108.60	639,110+
Appcelerator	Appcelerator Studio	AS	2,919	1	40	5.64	5	5	11.07	3.33	124.61	2,941,856#
	Aptana Studio	AP	829	1	40	8.02	8	8	35.46	5.95	124.61	6,536,521+
	Titanium SDK/CLI	TI	2,251	1	34	6.32	5	5	25.97	5.10	205.90	882,986+
DuraSpace	DuraCloud	DC	666	1	16	2.13	1	1	4.12	2.03	70.91	88,978+
Atlassian	Bamboo	BB	521	1	20	2.42	2	1	4.60	2.14	133.28	6,230,465#
	Clover	CV	384	1	40	4.59	2	1	42.95	6.55	124.48	890,020#
	JIRA Software	JI	352	1	20	4.43	3	5	12.35	3.51	114.57	7,070,022#
Moodle	Moodle	MD	1,166	1	100	15.54	8	5	468.53	21.65	88.86	2,976,645+
Lsstcorp	Data Management	DM	4,667	1	100	9.57	4	1	275.71	16.61	69.41	125,651*
Mulesoft	Mule	MU	889	1	21	5.08	5	5	12.24	3.50	81.16	589,212+
	Mule Studio	MS	732	1	34	6.40	5	5	29.01	5.39	70.99	16,140,452#
Spring	Spring XD	XD	3,526	1	40	3.70	3	1	10.42	3.23	78.47	107,916+
Talendforge	Talend Data Quality	TD	1,381	1	40	5.92	5	8	26.96	5.19	104.86	1,753,463#
	Talend ESB	TE	868	1	13	2.16	2	1	2.24	1.50	128.97	18,571,052#
	Total		23,313									

SP: story points, TD length: the number of words in the title and description of an issue, LOC: line of code
(+: LOC obtained from www.openhub.net, *: LOC from GitHub, and #: LOC from the reverse engineering)

(60% of the issues), development/validation set (i.e. 20%), and test set (i.e. 20%) based on their creation time. The issues in the training set and the validation set were created before the issues in the test set, and the issues in the training set were also created before the issues in the validation set.

5.3 Performance measures

There are a range of measures used in evaluating the accuracy of an effort estimation model. Most of them are based on the Absolute Error, (i.e. $|ActualSP - EstimatedSP|$). where $AcutalSP$ is the real story points assigned to an issue and $EstimatedSP$ is the outcome given by an estimation model. Mean of Magnitude of Relative Error (MRE) or Mean Percentage Error and Prediction at level l [65], i.e. Pred(l), have also been used in effort estimation. However, a number of studies [66–69] have found that those measures bias towards underestimation and are not stable when comparing effort estimation models. Thus, the *Mean Absolute Error (MAE)*, *Median Absolute Error (MdAE)*, and the *Standardized Accuracy (SA)* have recently been recommended to compare the performance of effort estimation models [19, 70]. MAE is defined as:

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|ActualSP_i - EstimatedSP_i|$$

where N is the number of issues used for evaluating the performance (i.e. test set), $ActualSP_i$ is the actual story point, and $EstimatedSP_i$ is the estimated story point, for the issue i.

We also report the Median Absolute Error (MdAE) since it is more robust to large outliers. MdAE is defined as:

$$MdAE = Median\{|ActualSP_i - EstimatedSP_i|\}$$

where $1 \leq i \leq N$.

SA is based on MAE and it is defined as:

$$SA = \left(1 - \frac{MAE}{MAE_{rguess}}\right) \times 100$$

where MAE_{rguess} is the MAE of a large number (e.g. 1000 runs) of random guesses. SA measures the comparison against random guessing. Predictive performance can be improved by decreasing MAE or increasing SA.

We assess the story point estimates produced by the estimation models using MAE, MdAE and SA. To compare the performance of two estimation models, we tested the statistical significance of the absolute errors achieved with the two models using the Wilcoxon Signed Rank Test [71]. The Wilcoxon test is a safe test since it makes no assumptions about underlying data distributions. The null hypothesis here is: "the absolute errors provided by an estimation model are not different to those provided by another estimation model". We set the confidence limit at 0.05 and also applied Bonferroni correction [72] (0.05/K, where K is the number of statistical tests) when multiple testing were performed.

In addition, we also employed a non-parametric effect size measure, the correlated samples case of the Vargha and Delaney's $\hat{A}_{XY}$ statistic [73] to assess whether the effect size is interesting. The $\hat{A}_{XY}$ measure is chosen since it is agnostic to the underlying distribution of the data, and is suitable for assessing randomized algorithms in software engineering generally [74] and effort estimation in particular [19]. Specifically, given a performance measure (e.g. the Absolute Error from each estimation in our case), the $\hat{A}_{XY}$ measures the probability that estimation model X achieves better results (with respect to the performance measure) than estimation model Y. We note that this falls into the correlated samples case of the Vargha and Delaney [73] where the Absolute Error is derived by applying different estimation methods on the same data (i.e. same issues). We thus use the following formula to calculate the stochastic superiority value between two estimation methods:

$$\hat{A}_{XY} = \frac{[\#(X < Y) + (0.5 \times \#(X = Y))]}{n},$$

where $\#(X < Y)$ is the number of issues that the Absolute Error from X less than Y, $\#(X = Y)$ is the number of issues that the Absolute Error from X equal to Y, and n is the number of issues. We also compute the average of the stochastic superiority measures (A_{iu}) of our approach

0098-5589 (c) 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2018.2792473, IEEE Transactions on Software Engineering

against each of the others using the following formular:

$$A_{iu} = \frac{\sum_{k \neq i} A_{ik}}{l - 1},$$

where A_{ik} is the pairwise stochastic superiority values ($\hat{A}_{XY}$) for all (i, k) pairs of estimation methods, $k = 1, ..., l$, and l is a number of estimation methods, e.g. variable i refers to Deep-SE and $l = 4$ when comparing Deep-SE against Random, Mean and Median methods.

5.4 Hyper-parameter settings for training a Deep-SE model

We focused on tuning two important hyper-parameters: the number of word embedding dimensions and the number of hidden layers in the recurrent highway net component of our model. To do so, we fixed one parameter and varied the other to observe the MAE performance. We chose to test with four different embedding sizes: 10, 50, 100, and 200, and twelve variations of the number of hidden layers from 2 to 200. The embedding size is the number of dimensions of the vector which represents a word. This word embedding is a low dimensional vector representation of words in the vocabulary. This tuning was done using the validation set. Figure 6 shows the results from experimenting with Apache Mesos. As can be seen, the setting where the number of embeddings is 50 and the number of hidden layers is 10 gives the lowest MAE, and thus was chosen.

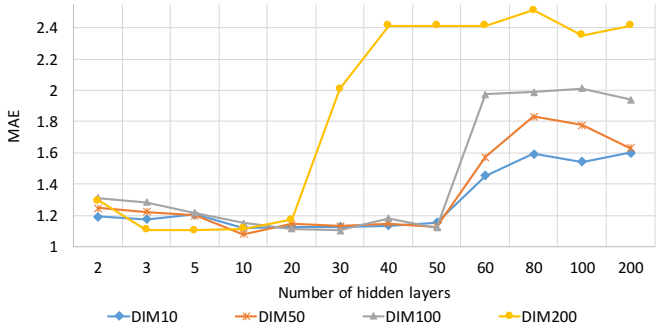


Fig. 6. Story point estimation performance with different parameter.

For both pre-training we trained with 100 runs and the batch size is 50. The initial learning rate in pre-training was set to 0.02, adaptation rate was 0.99, and smoothing factor was 10^{-7}. For the main Deep-SE model we used 1,000 epoches and the batch size wass set to 100. The initial learning rate in the main model was set to 0.01, adaptation rate was 0.9, and smoothing factor was 10^{-6}. Dropout rates for the RHWN and LSTM layers were set to 0.5 and 0.2 respectively. The maximum sequence length used by the LSTM is 100 words, which is the average length of issue description.

5.5 Pre-training

In most repositories, we used around 50,000 issues without story points (i.e. without labels) for pre-training, except the Mulesoft repository which has much smaller number of issues (only 8,036 issues) available for pre-training. Figure 7 show the top-500 frequent words used in Apache. They are divided into 9 clusters (using K-means clustering) based on their embedding which was learned through the pre-training process. We used t-distributed stochastic neighbor embedding (t-SNE) [75] to display high-dimensional vectors in two dimensions.

We show here some representative words from some clusters for a brief illustration. Words that are semantically related are grouped in the same cluster. For example, words related to networking like soap, configuration, tcp, and load are in one cluster. This indicates that to some extent, the learned vectors effectively capture the semantic relations between words, which is useful for the story-point estimation task we do later.

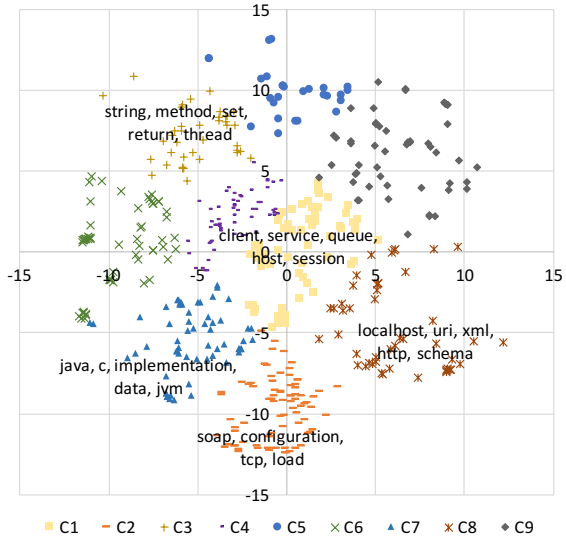


Fig. 7. Top-500 word clusters used in the Apache's issue reports

The pre-training step is known to effectively deal with limited labelled data [76–78]. Here, pre-training does not require story-point labels since it is trained by predicting the next words. Hence the number of data points equals to the number of words. Since for each project repository we used 50,000 issues for pre-training, we had approximately 5 million data points per repository for pre-training.

5.6 The correlation between the story points and the development time

Identifying the actual effort required for completing an issue is very challenging (especially in open source projects) since in most cases the actual effort was not tracked and recorded. We were however able to extract the development time which was the duration between when the issue's status was set to "in-progress" and when it was set to "resolved". Thus, we have explicitly excluded the waiting time for being assigned to a developer or being put on hold. The development time is the closest to the actual effort of completing the issue that we were able to extract from the data. We then performed two widely-used statistical tests (Spearman's rank and Pearson rank correlation) [79] for all the issues in our dataset. Table 2 shows the Spearman's rank and Pearson rank correlation coefficient and p-value for all projects. We have found that there is a significantly ($p < 0.05$) positive correlation between the story points and the development

0098-5589 (c) 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2018.2792473, IEEE Transactions on Software Engineering

TABLE 2
The coefficient and p-value of the Spearman's rank and Pearson rank correlation on the story points against the development time

Project	Spearman's rank coefficient	Spearman's rank p-value	Pearson correlation coefficient	Pearson correlation p-value
Appcelerator Studio	0.330	<0.001	0.311	<0.001
Aptana Studio	0.241	<0.001	0.325	<0.001
Bamboo	0.505	<0.001	0.476	<0.001
Clover	0.551	<0.001	0.418	<0.001
Data Management	0.753	<0.001	0.769	<0.001
DuraCloud	0.225	<0.001	0.393	<0.001
JIRA Software	0.512	<0.001	0.560	<0.001
Mesos	0.615	<0.001	0.766	<0.001
Moodle	0.791	<0.001	0.816	<0.001
Mule	0.711	<0.001	0.722	<0.001
Mule Studio	0.630	<0.001	0.565	<0.001
Spring XD	0.486	<0.001	0.614	<0.001
Talend Data Quality	0.390	<0.001	0.370	<0.001
Talend ESB	0.504	<0.001	0.524	<0.001
Titanium SDK/CLI	0.322	<0.001	0.305	<0.001
Usergrid	0.212	0.005	0.263	0.001

time across all 16 project we studied. In some projects (e.g. Moodle) there was a strong correlation with the coefficients was around 0.8. This positive correlation demonstrates that the higher story point, the longer development time, which suggests that a correlation between an issue's story points and its actual effort.

5.7 Results

We report here the results in answering research questions RQs 1–6.

RQ1: Sanity check

TABLE 3
Evaluation results of Deep-SE, the Mean and Median method (the best results are highlighted in bold). MAE and MdAE - the lower the better, SA - the higher the better.

Proj	Method	MAE	MdAE	SA	Proj	Method	MAE	MdAE	SA
ME	**Deep-SE**	**1.02**	**0.73**	**59.84**	JI	**Deep-SE**	**1.38**	**1.09**	**59.52**
	mean	1.64	1.78	35.61		mean	2.48	2.15	27.06
	median	1.73	2.00	32.01		median	2.93	2.00	13.88
UG	**Deep-SE**	**1.03**	**0.80**	**52.66**	MD	**Deep-SE**	**5.97**	**4.93**	**50.29**
	mean	1.48	1.23	32.13		mean	10.90	12.11	9.16
	median	1.60	1.00	26.29		median	7.18	6.00	40.16
AS	**Deep-SE**	**1.36**	**0.58**	**60.26**	DM	**Deep-SE**	**3.77**	**2.22**	**47.87**
	mean	2.08	1.52	39.02		mean	5.29	4.55	26.85
	median	1.84	1.00	46.17		median	4.82	3.00	33.38
AP	**Deep-SE**	**2.71**	**2.52**	**42.58**	MU	**Deep-SE**	**2.18**	**1.96**	**40.09**
	mean	3.15	3.46	33.30		mean	2.59	2.22	28.82
	median	3.71	4.00	21.54		median	2.69	2.00	26.07
TI	**Deep-SE**	**1.97**	**1.34**	**55.92**	MS	**Deep-SE**	**3.23**	**1.99**	**17.17**
	mean	3.05	1.97	31.59		mean	3.34	2.68	14.21
	median	2.47	2.00	44.65		median	3.30	2.00	15.42
DC	**Deep-SE**	**0.68**	**0.53**	**69.92**	XD	**Deep-SE**	**1.63**	**1.31**	**46.82**
	mean	1.30	1.14	42.88		mean	2.27	2.53	26.00
	median	0.73	1.00	68.08		median	2.07	2.00	32.55
BB	**Deep-SE**	**0.74**	**0.61**	**71.24**	TD	**Deep-SE**	**2.97**	**2.92**	**48.28**
	mean	1.75	1.31	32.11		mean	4.81	5.08	16.18
	median	1.32	1.00	48.72		median	3.87	4.00	32.43
CV	**Deep-SE**	**2.11**	**0.80**	**50.45**	TE	**Deep-SE**	**0.64**	**0.59**	**69.67**
	mean	3.49	3.06	17.84		mean	1.14	0.91	45.86
	median	2.84	2.00	33.33		median	1.16	1.00	44.44

Table 3 shows the results achieved from Deep-SE, and two baseline methods: Mean and Median method (See Appendix A.1 for the distribution of the Absolute Error). The analysis of MAE, MdAE, and SA suggests that the estimations obtained with our approach, Deep-SE, are better than those achieved by using Mean, Median, and Random estimates. Deep-SE consistently outperforms all these three baselines in all sixteen projects.

Our approach improved between 3.29% (in project MS) to 57.71% (in project BB) in terms of MAE, 11.71% (in MU) to 73.86% (in CV) in terms of MdAE, and 20.83% (in MS) to 449.02% (in MD) in terms of SA over the Mean method. The improvements of our approach over the Median method are between 2.12% (in MS) to 52.90% (in JI) in MAE, 0.50% (in MS) to 63.50% (in ME) in MdAE, and 2.70% (in DC) to 328.82% (in JI) in SA. Overall, the improvement achieved by Deep-SE over the Mean and Median method is 34.06% and 26.77% in terms of MAE, averaging across all projects.

We note that the results achieved by the estimation models vary between different projects. For example, our Deep-SE achieved 0.64 MAE in the Talend ESB project (TE), while it achieved 5.97 MAE in Moodle (MD) project. The distribution of story points may be the cause of this variation: the standard deviation of story points in TE is only 1.50, while that in MD is 21.65 (see Table 1).

TABLE 4
Comparison on the effort estimation benchmarks using Wilcoxon test and $\hat{A}_{XY}$ effect size (in brackets)

Deep-SE vs	Mean		Median		Random		A_{iu}
ME	<0.001	[0.77]	<0.001	[0.81]	<0.001	[0.90]	0.83
UG	<0.001	[0.79]	<0.001	[0.79]	<0.001	[0.81]	0.80
AS	<0.001	[0.78]	<0.001	[0.78]	<0.001	[0.91]	0.82
AP	0.040	[0.69]	<0.001	[0.79]	<0.001	[0.84]	0.77
TI	<0.001	[0.77]	<0.001	[0.72]	<0.001	[0.88]	0.79
DC	<0.001	[0.80]	0.415	[0.54]	<0.001	[0.81]	0.72
BB	<0.001	[0.78]	<0.001	[0.78]	<0.001	[0.85]	0.80
CV	<0.001	[0.75]	<0.001	[0.70]	<0.001	[0.91]	0.79
JI	<0.001	[0.76]	<0.001	[0.79]	<0.001	[0.79]	0.78
MD	<0.001	[0.81]	<0.001	[0.75]	<0.001	[0.80]	0.79
DM	<0.001	[0.69]	<0.001	[0.59]	<0.001	[0.75]	0.68
MU	0.003	[0.73]	<0.001	[0.73]	<0.001	[0.82]	0.76
MS	0.799	[0.56]	0.842	[0.56]	<0.001	[0.69]	0.60
XD	<0.001	[0.70]	<0.001	[0.70]	<0.001	[0.78]	0.73
TD	<0.001	[0.86]	<0.001	[0.85]	<0.001	[0.87]	0.86
TE	<0.001	[0.73]	<0.001	[0.73]	<0.001	[0.92]	0.79

Table 4 shows the results of the Wilcoxon test (together with the corresponding $\hat{A}_{XY}$ effect size) to measure the statistical significance and effect size (in brackets) of the improved accuracy achieved by Deep-SE over the baselines: Mean Effort, Median Effort, and Random Guessing. In 45/48 cases, our Deep-SE significantly outperforms the baselines after applying Bonferroni correction with effect sizes greater than 0.5. Moreover, the average of the stochastic superiority (A_{iu}) of our approach against the baselines is greater than 0.7 in the most cases. The highest A_{iu} achieving in the Talend Data Quality project (TD) is 0.86 which can be considered as large effect size ($\hat{A}_{XY} > 0.8$).

We note that the improvement brought by our approach over the baselines was not significant for project MS. One possible reason is that the size of the training and pre-training data for MS is small, and deep learning techniques tend to perform well with large training samples.

Our approach outperforms the baselines, thus passing the sanity check required by RQ1.

0098-5589 (c) 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2018.2792473, IEEE Transactions on Software Engineering

RQ2: Benefits of deep representation

TABLE 5
Evaluation results of Deep-SE, LSTM+RF, LSTM+SVM, LSTM+ATLM, and LSTM+LR (the best results are highlighted in bold). MAE and MdAE - the lower the better, SA - the higher the better.

Proj	Method	MAE	MdAE	SA	Proj	Method	MAE	MdAE	SA
ME	**Deep-SE**	**1.02**	**0.73**	**59.84**	JI	**Deep-SE**	**1.38**	**1.09**	**59.52**
	lstm+rf	1.08	0.90	57.57		lstm+rf	1.71	1.27	49.71
	lstm+svm	1.07	0.90	58.02		lstm+svm	2.04	1.89	40.05
	lstm+atlm	1.08	0.95	57.60		lstm+atlm	2.10	1.95	38.26
	lstm+lr	1.10	0.96	56.94		lstm+lr	2.10	1.95	38.26
UG	**Deep-SE**	**1.03**	**0.80**	**52.66**	MD	**Deep-SE**	**5.97**	**4.93**	**50.29**
	lstm+rf	1.07	0.85	50.70		lstm+rf	9.86	9.69	17.86
	lstm+svm	1.06	1.04	51.23		lstm+svm	6.70	5.44	44.19
	lstm+atlm	1.40	1.20	35.55		lstm+atlm	9.97	9.61	16.92
	lstm+lr	1.40	1.20	35.55		lstm+lr	9.97	9.61	16.92
AS	**Deep-SE**	**1.36**	**0.58**	**60.26**	DM	**Deep-SE**	**3.77**	**2.22**	**47.87**
	lstm+rf	1.62	1.40	52.38		lstm+rf	4.51	3.69	37.71
	lstm+svm	1.46	1.42	57.20		lstm+svm	4.20	2.87	41.93
	lstm+atlm	1.59	1.30	53.29		lstm+atlm	4.70	3.74	35.01
	lstm+lr	1.68	1.46	50.78		lstm+lr	5.30	3.66	26.68
AP	**Deep-SE**	**2.71**	**2.52**	**42.58**	MU	**Deep-SE**	**2.18**	**1.96**	**40.09**
	lstm+rf	2.96	2.80	37.34		lstm+rf	2.20	2.21	38.73
	lstm+svm	3.06	2.90	35.26		lstm+svm	2.28	2.89	37.44
	lstm+atlm	3.06	2.76	35.21		lstm+atlm	2.46	2.39	32.51
	lstm+lr	3.75	3.66	20.63		lstm+lr	2.46	2.39	32.51
TI	**Deep-SE**	**1.97**	**1.34**	**55.92**	MS	**Deep-SE**	**3.23**	**1.99**	**17.17**
	lstm+rf	2.32	1.97	48.02		lstm+rf	3.30	2.77	15.30
	lstm+svm	2.00	2.10	55.20		lstm+svm	3.31	3.09	15.10
	lstm+atlm	2.51	2.03	43.87		lstm+atlm	3.42	2.75	12.21
	lstm+lr	2.71	2.31	39.32		lstm+lr	3.42	2.75	12.21
DC	**Deep-SE**	**0.68**	**0.53**	**69.92**	XD	**Deep-SE**	**1.63**	**1.31**	**46.82**
	lstm+rf	0.69	0.62	69.52		lstm+rf	1.81	1.63	40.99
	lstm+svm	0.75	0.90	67.02		lstm+svm	1.80	1.77	41.33
	lstm+atlm	0.87	0.59	61.57		lstm+atlm	1.83	1.65	40.45
	lstm+lr	0.80	0.67	64.96		lstm+lr	1.85	1.72	39.63
BB	**Deep-SE**	**0.74**	**0.61**	**71.24**	TD	**Deep-SE**	**2.97**	**2.92**	**48.28**
	lstm+rf	1.01	1.00	60.95		lstm+rf	3.89	4.37	32.14
	lstm+svm	0.81	1.00	68.55		lstm+svm	3.49	3.37	39.13
	lstm+atlm	1.97	1.78	23.70		lstm+atlm	3.86	4.11	32.71
	lstm+lr	1.26	1.16	51.24		lstm+lr	3.79	3.67	33.88
CV	**Deep-SE**	**2.11**	**0.80**	**50.45**	TE	**Deep-SE**	**0.64**	**0.59**	**69.67**
	lstm+rf	3.08	2.77	27.58		lstm+rf	0.66	0.65	68.51
	lstm+svm	2.50	2.32	41.22		lstm+svm	0.70	0.90	66.61
	lstm+atlm	3.11	2.49	26.90		lstm+atlm	0.70	0.72	66.51
	lstm+lr	3.36	2.76	21.07		lstm+lr	0.77	0.71	63.20

Table 5 shows MAE, MdAE, and SA achieved from Deep-SE using Recurrent Highway Networks (RHWN) for deep representation of issue reports against using Random Forests, Support Vector Machine, Automatically Transformed Linear Model, and Linear Regression Model coupled with LSTM (i.e. LSTM+RF, LSTM+SVM, LSTM+ATLM, and LSTM+LR). The distribution of the Absolute Error is reported in Appendix A.2. When we use MAE, MdAE, and SA as evaluation criteria, Deep-SE is still the best approach, consistently outperforming LSTM+RF, LSTM+SVM, LSTM+ATLM, and LSTM+LR across all sixteen projects.

Using RHWN improved over RF between 0.91% (in MU) to 39.45% (in MD) in MAE, 5.88% (in UG) to 71.12% (in CV) in MdAE, and 0.58% (in DC) to 181.58% (in MD) in SA. The improvements of RHWN over SVM are between 1.50% (in TI) to 32.35% (in JI) in MAE, 9.38% (in MD) to 65.52% (in CV) in MdAE, and 1.30% (in TI) to 48.61% (in JI). In terms of using ATLM, RHWN improved over it between 5.56% (in MS) to 62.44% (in BB) in MAE, 8.70% (in AP) to 67.87% (in CV) in MdAE, and 3.89% (in ME) to 200.59% (in BB) in SA. Overall, RHWN improved , in terms of MAE, 9.63% over SVM, 13.96% over RF, 21.84% over ATLM, and 23.24% over LR, averaging across all projects.

In addition, the results for the Wilcoxon test to compare our approach (Deep-SE) against LSTM+RF, LSTM+SVM, LSTM+ATLM, and LSTM+LR is shown in Table 6. The improvement of our approach over LSTM+RF, LSTM+SVM, and LSTM+ATLM is still significant after applying p-value correction with the effect size greater than 0.5 in 59/64 cases. In most cases, when comparing the proposed model against LSTM+RF, LSTM+SVM, LSTM+ATLM, and LSTM+LR, the effect sizes are small (between 0.5 and 0.6). A major part of those improvement were brought by our use of the deep learning LSTM architecture to model the textual description of an issue. The use of highway recurrent networks (on top of LSTM) has also improved the predictive performance, but not as large effects as the LSTM itself (especially for those projects which have very small number of issues). However, our approach, Deep-SE, achieved A_{iu} greater than 0.6 in the most cases.

TABLE 6
Comparison between the Recurrent Highway Net against Random Forests, Support Vector Machine, Automatically Transformed Linear Model, and Linear Regression using Wilcoxon test and $\hat{A}_{12}$ effect size (in brackets)

Deep-SE vs	LSTM+RF	LSTM+SVM	LSTM+ATLM	LSTM+LR	A_{iu}
ME	<0.001 [0.57]	<0.001 [0.54]	<0.001 [0.59]	<0.001 [0.59]	0.57
UG	0.004 [0.59]	0.010 [0.55]	<0.001 [1.00]	<0.001 [0.73]	0.72
AS	<0.001 [0.69]	<0.001 [0.51]	<0.001 [0.71]	<0.001 [0.75]	0.67
AP	<0.001 [0.60]	<0.001 [0.52]	<0.001 [0.62]	<0.001 [0.64]	0.60
TI	<0.001 [0.65]	0.007 [0.51]	<0.001 [0.69]	<0.001 [0.71]	0.64
DC	0.406 [0.55]	0.015 [0.60]	<0.001 [0.97]	0.024 [0.58]	0.68
BB	<0.001 [0.73]	0.007 [0.60]	<0.001 [0.84]	<0.001 [0.75]	0.73
CV	<0.001 [0.70]	0.140 [0.63]	<0.001 [0.82]	0.001 [0.70]	0.71
JI	0.006 [0.71]	0.001 [0.67]	0.002 [0.89]	<0.001 [0.79]	0.77
MD	<0.001 [0.76]	<0.001 [0.57]	<0.001 [0.74]	<0.001 [0.69]	0.69
DM	<0.001 [0.62]	<0.001 [0.56]	<0.001 [0.61]	<0.001 [0.62]	0.60
MU	0.846 [0.53]	0.005 [0.62]	0.009 [0.67]	0.003 [0.64]	0.62
MS	0.502 [0.53]	0.054 [0.50]	<0.001 [0.82]	0.195 [0.56]	0.60
XD	<0.001 [0.63]	<0.001 [0.57]	<0.001 [0.65]	<0.001 [0.60]	0.61
TD	<0.001 [0.78]	<0.001 [0.68]	<0.001 [0.70]	<0.001 [0.70]	0.72
TE	0.020 [0.53]	0.002 [0.59]	<0.001 [0.66]	0.006 [0.65]	0.61

The proposed approach of using Recurrent Highway Networks is effective in building a deep representation of issue reports and consequently improving story point estimation.

RQ3: Benefits of LSTM document representation

To study the benefits of using LSTM in representing issue reports, we compared the improved accuracy achieved by Random Forest using the features derived from LSTM against that using the features derived from BoW and Doc2vec. For a fair comparison we used Random Forests as the regressor in all settings and the result is reported in Table 7 (see the distribution of the Absolute Error in Appendix A.3). LSTM performs better than BoW and Doc2vec with respect to the MAE, MdAE, and SA measures in twelve projects (e.g. ME, UG, and AS) from sixteen projects. LSTM improved 4.16% and 11.05% in MAE over Doc2vec and BoW, respectively, averaging across all projects.

Among those twelve projects, LSTM improved over BoW between 0.30% (in MS) to 28.13% (in DC) in terms of MAE, 1.06% (in AP) to 45.96% (in JI) in terms of MdAE, and 0.67% (in AP) to 47.77% (in TD) in terms of SA. It also improved over Doc2vec between 0.45% (in MU) to 18.57% (in JI) in terms of MAE, 0.71% (in AS) to 40.65% (in JI) in terms of MdAE, and 2.85% (in TE) to 31.29% (in TD) in terms of SA.

We acknowledge that BoW and Doc2vec perform better than LSTM in some cases. For example, in the Moodle project (MD), D2V+RF performed better than LSTM+RF

0098-5589 (c) 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2018.2792473, IEEE Transactions on Software Engineering

TABLE 7
Evaluation results of LSTM+RF, BoW+RF, and Doc2vec+RF (the best results are highlighted in bold). MAE and MdAE - the lower the better, SA - the higher the better.

Proj	Method	MAE	MdAE	SA	Proj	Method	MAE	MdAE	SA
ME	**lstm+rf**	**1.08**	**0.90**	**57.57**	JI	**lstm+rf**	**1.71**	**1.27**	**49.71**
	bow+rf	1.31	1.34	48.66		bow+rf	2.10	2.35	38.34
	d2v+rf	1.14	0.98	55.28		d2v+rf	2.10	2.14	38.29
UG	**lstm+rf**	**1.07**	**0.85**	**50.70**	MD	**lstm+rf**	9.86	**9.69**	17.86
	bow+rf	1.19	1.28	45.24		bow+rf	10.20	10.22	15.07
	d2v+rf	1.12	0.92	48.47		d2v+rf	**8.02**	9.87	**33.19**
AS	**lstm+rf**	**1.62**	**1.40**	**52.38**	DM	**lstm+rf**	**4.51**	**3.69**	**37.71**
	bow+rf	1.83	1.53	46.34		bow+rf	4.78	3.98	33.84
	d2v+rf	1.62	1.41	52.38		d2v+rf	4.71	3.99	34.87
AP	**lstm+rf**	**2.96**	**2.80**	**37.34**	MU	**lstm+rf**	**2.20**	**2.21**	38.73
	bow+rf	2.97	2.83	37.09		bow+rf	2.31	2.54	36.64
	d2v+rf	3.20	2.91	32.29		d2v+rf	2.21	2.69	**39.36**
TI	**lstm+rf**	**2.32**	**1.97**	**48.02**	MS	**lstm+rf**	**3.30**	2.77	15.30
	bow+rf	2.58	2.30	42.15		bow+rf	3.31	**2.57**	**15.58**
	d2v+rf	2.41	2.16	46.02		d2v+rf	3.40	2.93	12.79
DC	**lstm+rf**	**0.69**	**0.62**	**69.52**	XD	**lstm+rf**	**1.81**	**1.63**	**40.99**
	bow+rf	0.96	1.11	57.78		bow+rf	1.98	1.72	35.56
	d2v+rf	0.77	0.77	66.14		d2v+rf	1.88	1.73	38.72
BB	**lstm+rf**	**1.01**	**1.00**	**60.95**	TD	**lstm+rf**	**3.89**	**4.37**	**32.14**
	bow+rf	1.34	1.26	48.06		bow+rf	4.49	5.05	21.75
	d2v+rf	1.12	1.16	56.51		d2v+rf	4.33	4.80	24.48
CV	**lstm+rf**	3.08	**2.77**	27.58	TE	**lstm+rf**	**0.66**	**0.65**	**68.51**
	bow+rf	**2.98**	2.93	**29.91**		bow+rf	0.86	0.69	58.89
	d2v+rf	3.16	2.79	25.70		d2v+rf	0.70	0.89	66.61

TABLE 8
Comparison of Random Forest with LSTM, Random Forests with BoW, and Random Forests with Doc2vec using Wilcoxon test and $\hat{A}_{XY}$ effect size (in brackets)

LSTM vs	BoW	Doc2Vec	A_{iu}
ME	<0.001 [0.70]	0.142 [0.53]	0.62
UG	<0.001 [0.71]	0.135 [0.60]	0.66
AS	<0.001 [0.66]	<0.001 [0.51]	0.59
AP	0.093 [0.51]	0.144 [0.52]	0.52
TI	<0.001 [0.67]	<0.001 [0.55]	0.61
DC	<0.001 [0.73]	0.008 [0.59]	0.66
BB	<0.001 [0.77]	0.002 [0.66]	0.72
CV	0.109 [0.61]	0.581 [0.57]	0.59
JI	0.009 [0.67]	0.011 [0.62]	0.65
MD	0.022 [0.63]	0.301 [0.51]	0.57
DM	<0.001 [0.60]	<0.001 [0.55]	0.58
MU	0.006 [0.59]	0.011 [0.57]	0.58
MS	0.780 [0.54]	0.006 [0.57]	0.56
XD	<0.001 [0.60]	0.005 [0.55]	0.58
TD	<0.001 [0.73]	<0.001 [0.67]	0.70
TE	<0.001 [0.69]	0.005 [0.61]	0.65

in MAE and SA – it achieved 8.02 MAE and 33.19 SA. This could reflect that the combination between LSTM and RHWN significantly improves the accuracy of the estimations.

The improvement of LSTM over BoW and Doc2vec is significant after applying Bonferroni correction with effect size greater than 0.5 in 24/32 cases and A_{iu} being greater than 0.5 in all projects (see Table 8).

The proposed LSTM-based approach is effective in automatically learning semantic features representing issue description, which improves story-point estimation.

TABLE 9
Mean Absolute Error (MAE) on cross-project estimation and comparison of Deep-SE and ABE0 using Wilcoxon test and $\hat{A}_{XY}$ effect size (in brackets)

Source	Target	Deep-SE	ABE0	Deep-SE vs ABE0
		(i) within-repository		
ME	UG	1.07	1.23	<0.001 [0.78]
UG	ME	1.14	1.22	0.012 [0.52]
AS	AP	2.75	3.08	<0.001 [0.67]
AS	TI	1.99	2.56	<0.001 [0.70]
AP	AS	2.85	3.00	0.051 [0.55]
AP	TI	3.41	3.53	0.003 [0.56]
MU	MS	3.14	3.55	0.041 [0.55]
MS	MU	2.31	2.64	0.030 [0.56]
Avg		2.33	2.60	
		(ii) cross-repository		
AS	UG	1.57	2.04	0.004 [0.61]
AS	ME	2.08	2.14	0.022 [0.51]
MD	AP	5.37	6.95	<0.001 [0.58]
MD	TI	6.36	7.10	0.097 [0.54]
MD	AS	5.55	6.77	<0.001 [0.61]
DM	TI	2.67	3.94	<0.001 [0.64]
UG	MS	4.24	4.45	0.005 [0.54]
ME	MU	2.70	2.97	0.015 [0.53]
Avg		3.82	4.55	

RQ4: Cross-project estimation

We performed sixteen sets of cross-project estimation experiments to test two settings: (i) within-repository: both the source and target projects (e.g. Apache Mesos and Apache Usergrid) were from the same repository, and pre-training was done using only the source projects, not the target projects; and (ii) cross-repository: the source project (e.g. Appcelerator Studio) was in a different repository from the target project Apache Usergrid, and pre-training was done using only the source project.

Table 9 shows the performance of our Deep-SE model and ABE0 for cross-project estimation (see the distribution of the Absolute Error in Appendix A.4). We also used a benchmark of within-project estimation where older issues of the target project were used for training (see Table 3). In all cases, the proposed approach when used for cross-project estimation performed worse than when used for within-project estimation (e.g. on average 20.75% reduction in performance for within-repository and 97.92% for cross-repository). However, our approach outperformed the cross-project baseline (i.e. ABE0) in all cases – it achieved 2.33 and 3.82 MAE in within and cross repository, while ABE0 achieved 2.60 and 4.55 MAE. The improvement of our approach over ABE0 is still significant after applying p-value correction with the effect size greater than 0.5 in 14/16 cases.

These results confirm a universal understanding [25] in agile development that story point estimation is specific to teams and projects. Since story points are relatively measured, it is not uncommon that two different same-sized teams could give different estimates for the same user story. For example, team A may estimate 5 story points for user story UC_1 while team B gives 10 story points. However, it does not necessarily mean that team B would do more work for completing UC_1 than team A. It more likely means

0098-5589 (c) 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2018.2792473, IEEE Transactions on Software Engineering

that team'B baselines are twice bigger than team A's, i.e. for "baseline" user story which requires 5 times less the effort than UC_1 takes, team A would give it 1 story point while team B gives 2 story points. Hence, historical estimates are more valuable for within-project estimation, which is demonstrated by this result.

Given the specificity of story points to teams and projects, our proposed approach is more effective for within-project estimation.

RQ5: Adjusted/normalized story points

Table 10 shows the results of our Deep-SE and the other baseline methods in predicting the normalized story points. Deep-SE performs well across all projects. Deep-SE improved MAE between 2.13% to 93.40% over the Mean method, 9.45% to 93.27% over the Median method, 7.02% to 53.33% over LSTM+LR, 1.20% to 61.96% over LSTM+ATLM, 1.20% to 53.33% over LSTM+SVM, 4.00% to 30.00% over Doc2vec+RF, 2.04% to 36.36% over BoW+RF, and 0.86% to 25.80% over LSTM+RF. The best result is obtained in the Usergrid project (UG), it is 0.07 MAE, 0.01 MdAE, and 93.50 SA. We however note that the adjusted story points benefits all methods since it narrows the gap between minimum and maximum value and the distribution of the story points.

Our proposed approach still outperformed other techniques in estimating the new adjusted story points.

RQ6: Compare Deep-SE against the existing approach

We applied our approach, Deep-SE, and the Porru et. al.'s approach on their dataset consisted of eight projects. Table 11 shows the evaluation results in MAE and the comparison of Deep-SE and the Porru et. al.'s approach. The distribution of the Absolute Error is reported in Appendix A.5. Deep-SE outperforms the existing approach in all cases. Deep-SE improved between 18.18% (in TIMOB) to 56.48% (in DNN) in terms of MAE. In addition, the improvement of our approach over the Porru et. al.'s approach is still significant after applying p-value correction with the effect size greater than 0.5 in all cases. Especially, the large effect size ($\hat{A}_{XY} > 0.7$) of the improvement is obtained in the DNN project.

Our proposed approach outperformed the existing technique using TF-IDF in estimating the story points.

5.8 Training/testing time

Deep learning models are known for taking a long time for training. This is an important factor in considering adopting our approach, especially in an agile development setting. If training time takes longer than the duration of a sprint (e.g. one or two weeks), the prediction system would not be useful in practice. We have found that the training time of our model was very small, ranging from 13 minutes to 40 minutes with an average of 22 minutes across the 16 projects (see Table 12). Pre-training time took much longer time, but it was done only once across a repository and took just below 7 hours at the maximum. Once the model was

TABLE 10
Evaluation results on the adjusted story points (the best results are highlighted in bold). MAE and MdAE - the lower the better, SA - the higher the better.

Proj	Method	MAE	MdAE	SA	Proj	Method	MAE	MdAE	SA
ME	**Deep-SE**	**0.27**	**0.03**	**76.58**	JI	**Deep-SE**	**0.60**	**0.51**	**63.20**
	lstm+rf	0.34	0.15	70.43		lstm+rf	0.74	0.79	54.42
	bow+rf	0.36	0.16	68.82		bow+rf	0.66	0.53	58.99
	d2v+rf	0.35	0.15	69.87		d2v+rf	0.70	0.53	56.99
	lstm+svm	0.33	0.10	71.20		lstm+svm	0.94	0.89	41.97
	lstm+atlm	0.33	0.14	70.97		lstm+atlm	0.89	0.89	45.18
	lstm+lr	0.37	0.21	67.68		lstm+lr	0.89	0.89	45.18
	mean	1.12	1.07	3.06		mean	1.31	1.71	18.95
	median	1.05	1.00	8.87		median	1.60	2.00	1.29
UG	**Deep-SE**	**0.07**	0.01	**93.50**	MD	**Deep-SE**	**2.56**	**2.29**	**31.83**
	lstm+rf	0.08	**0.00**	92.59		lstm+rf	3.45	3.55	8.24
	bow+rf	0.11	0.01	90.31		bow+rf	3.32	3.27	11.54
	d2v+rf	0.10	0.01	91.22		d2v+rf	3.39	3.48	9.70
	lstm+svm	0.15	0.10	86.38		lstm+svm	3.12	3.07	16.94
	lstm+atlm	0.15	0.08	86.25		lstm+atlm	3.48	3.49	7.41
	lstm+lr	0.15	0.08	86.25		lstm+lr	3.57	3.28	4.98
	mean	1.04	0.98	4.79		mean	3.60	3.67	4.18
	median	1.06	1.00	2.64		median	2.95	3.00	21.48
AS	**Deep-SE**	**0.53**	**0.20**	**69.16**	DM	**Deep-SE**	**2.30**	**1.43**	**31.99**
	lstm+rf	0.56	0.45	67.49		lstm+rf	2.83	2.59	16.23
	bow+rf	0.56	0.49	67.39		bow+rf	2.83	2.63	16.33
	d2v+rf	0.56	0.46	67.37		d2v+rf	2.92	2.80	13.80
	lstm+svm	0.55	0.32	68.34		lstm+svm	2.45	1.78	27.56
	lstm+atlm	0.57	0.46	66.87		lstm+atlm	2.83	2.57	16.28
	lstm+lr	0.57	0.49	67.12		lstm+lr	2.83	2.57	16.28
	mean	1.18	0.79	31.89		mean	3.27	3.41	3.25
	median	1.35	1.00	21.54		median	2.61	2.00	22.94
AP	**Deep-SE**	**0.92**	**0.86**	**21.95**	MU	**Deep-SE**	**0.68**	0.59	**63.83**
	lstm+rf	0.99	0.87	16.23		lstm+rf	0.70	**0.55**	63.01
	bow+rf	1.00	0.87	15.33		bow+rf	0.70	0.57	62.79
	d2v+rf	0.99	0.86	15.94		d2v+rf	0.71	0.57	62.17
	lstm+svm	1.12	0.92	5.26		lstm+svm	0.70	0.62	62.62
	lstm+atlm	1.03	0.84	12.63		lstm+atlm	0.93	0.74	50.77
	lstm+lr	1.17	1.05	1.14		lstm+lr	0.79	0.61	58.00
	mean	1.15	0.64	2.49		mean	1.21	1.51	35.86
	median	0.94	1.00	20.29		median	1.64	2.00	12.80
TI	**Deep-SE**	**0.59**	**0.17**	**56.53**	MS	**Deep-SE**	**0.86**	**0.65**	**56.82**
	lstm+rf	0.72	0.56	46.22		lstm+rf	0.91	0.76	54.37
	bow+rf	0.73	0.58	46.10		bow+rf	0.89	0.93	55.48
	d2v+rf	0.72	0.56	46.17		d2v+rf	0.90	0.69	54.66
	lstm+svm	0.73	0.62	45.74		lstm+svm	0.94	0.78	52.91
	lstm+atlm	0.73	0.57	45.86		lstm+atlm	0.99	0.87	50.45
	lstm+lr	0.73	0.56	45.77		lstm+lr	0.99	0.87	50.45
	mean	1.32	1.56	1.57		mean	1.23	0.62	38.49
	median	0.86	1.00	36.04		median	1.44	1.00	27.83
DC	**Deep-SE**	**0.48**	**0.48**	**55.77**	XD	**Deep-SE**	**0.35**	**0.08**	**80.66**
	lstm+rf	0.49	0.49	55.02		lstm+rf	0.44	0.37	75.78
	bow+rf	0.49	**0.48**	54.76		bow+rf	0.45	0.38	75.33
	d2v+rf	0.50	0.50	53.59		d2v+rf	0.45	0.32	75.31
	lstm+svm	0.49	0.43	55.24		lstm+svm	0.38	0.20	79.16
	lstm+atlm	0.53	0.47	51.02		lstm+atlm	0.92	0.76	49.05
	lstm+lr	0.53	0.47	51.02		lstm+lr	0.45	0.40	75.33
	mean	1.07	1.49	1.29		mean	1.03	1.28	43.06
	median	0.58	1.00	46.76		median	0.75	1.00	58.74
BB	**Deep-SE**	**0.41**	**0.12**	**72.00**	TD	**Deep-SE**	**0.82**	**0.64**	**53.36**
	lstm+rf	0.43	0.38	70.37		lstm+rf	0.84	0.68	52.65
	bow+rf	0.45	0.40	69.33		bow+rf	0.88	0.65	50.30
	d2v+rf	0.49	0.45	66.34		d2v+rf	0.86	0.70	51.46
	lstm+svm	0.42	0.21	71.21		lstm+svm	0.83	0.62	53.24
	lstm+atlm	0.47	0.41	67.53		lstm+atlm	0.83	0.58	52.82
	lstm+lr	0.47	0.41	67.53		lstm+lr	0.90	0.74	48.88
	mean	1.15	0.76	20.92		mean	1.29	1.42	27.20
	median	1.39	1.00	4.50		median	0.99	1.00	44.17
CV	**Deep-SE**	**1.15**	**0.79**	**23.29**	TE	**Deep-SE**	**0.40**	**0.05**	**74.58**
	lstm+rf	1.16	1.05	22.55		lstm+rf	0.47	0.46	70.39
	bow+rf	1.22	1.10	18.95		bow+rf	0.48	0.48	69.52
	d2v+rf	1.20	1.09	20.30		d2v+rf	0.48	0.48	69.41
	lstm+svm	1.22	1.15	18.77		lstm+svm	0.45	0.41	71.77
	lstm+atlm	1.47	1.28	2.22		lstm+atlm	0.49	0.48	69.14
	lstm+lr	1.47	1.28	2.22		lstm+lr	0.49	0.48	69.14
	mean	1.27	1.11	15.18		mean	0.99	0.60	37.28
	median	1.29	1.00	13.92		median	1.39	1.00	12.09

0098-5589 (c) 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2018.2792473, IEEE Transactions on Software Engineering

TABLE 11
Mean Absolute Error (MAE) and comparison of Deep-SE and the Porru's approach using Wilcoxon test and $\hat{A}_{XY}$ effect size (in brackets)

Proj	Deep-SE	Porru	Deep-SE vs Porru
APSTUD	2.67	5.69	<0.001 [0.63]
DNN	0.47	1.08	<0.001 [0.74]
MESOS	0.76	1.23	0.003 [0.70]
MULE	2.32	3.37	<0.001 [0.61]
NEXUS	0.21	0.39	0.005 [0.67]
TIMOB	1.44	1.76	0.047 [0.57]
TISTUD	1.04	1.28	<0.001 [0.58]
XD	1.00	1.86	<0.001 [0.69]
avg	1.24	2.08	

TABLE 12
The pre-training, training, and testing time at 50 embedding dimensions of our Deep-SE model

Repository	Pre-training time	Proj.	Training time	Testing time
Apache	6 h 28 min	ME	23 min	1.732 s
		UG	15 min	0.395 s
Appcelerator	5 h 11 min	AS	27 min	2.209 s
		AP	18 min	0.428 s
		TI	32 min	2.528 s
Duraspace	3 h 34 min	DC	18 min	1.475 s
Jira	6 h 42 min	BB	15 min	0.267 s
		CV	14 min	0.219 s
		JI	13 min	0.252 s
Moodle	6 h 29 min	MD	15 min	1.789 s
Lsstcorp	3 h 26 min	DM	40 min	5.293 s
Mulesoft	2 h 39 min	MU	21 min	0.535 s
		MS	17 min	0.718 s
Spring	5 h 20 min	XD	40 min	2.774 s
Talendforge	6 h 56 min	TD	19 min	1.168 s
		TE	16 min	0.591 s

trained, getting an estimation from the model was very fast. As can be seen from Table 12, the time it took for testing all issues in the test sets was in the order of seconds. Hence, for a given new issue, it would take less than a second for the machinery to come back with an story point estimation. All the experiments were run on a MacOS laptop with 2.4 GHz Intel Core i5 and 8 GB of RAM and the embedding dimensions of 50. Hence, this result suggests that using our proposed approach to estimate story points is applicable in practice.

5.9 Verifiability

We have created a replication package and made it available at http://www.dsl.uow.edu.au/sasite/index.php/storypoint/. The package contains the full dataset and the source code of our Deep-SE model and the benchmark models (i.e. the baselines, LSTM+RF, Doc2vec+RF, BoW+RF, LSTM+SVM, and LSTM+ATLM). On this website, we also provide detailed instructions on how to run the code and replicate all the experiments we reported in this paper so that our results can be independently verified.

5.10 Threats to validity

We tried to mitigate threats to construct validity by using real world data from issues recorded in large open source projects. We collected the title and description provided with these issue reports and the actual story points that were assigned to them. We are aware that those story points were estimated by human teams, and thus may contain biases and in some cases may not be accurate. We have mitigated this threats by performing two set of experiments: one on the orgianal story points and the other on the adjusted normalized story points. We further note that for story points, the raw values are not as important as the relative values [80]. A user story that is assigned 6 story points should be three times as much as a user story that is assigned 2 story points. Hence, when engineers determine an estimate for a new issue, they need to compare the issue to other issues in the past in order to make the estimation consistently. The problem is thus suitable for a machine learner. The trained prediction system works in a similar manner as human engineers: using past estimates as baselines for new estimation. The prediction system tries to reproduce an estimate that human engineers would arrive at.

However, since we aim to mimic the team's capability in effort estimation, the current set of ground-truths sufficiently serves this purpose. When other sets of ground-truths become available, our model can be easily retrained. To minimize threats to conclusion validity, we carefully selected unbiased error measures, applied a number of statistical tests, and applied multiple statistical testing correction to verify our assumptions [81]. Our study was performed on datasets of different sizes. In addition, we carefully followed recent best practices in evaluating effort estimation models [55, 57, 74] to decrease conclusion instability [82].

The original implementation of Porru et. al.'s method [64] was not released, thus we have re-implemented our own version of their approach. We strictly followed the described provided in their work, however we acknowledge that our implementation may not reflect all the implementation details in their approach. To mitigate this threat, we have tested our implementation using the dataset provided in their work. We have found that our results were consistent with the results reported in their work.

To mitigate threats to external validity, we have considered 23,313 issues from sixteen open source projects, which differ significantly in size, complexity, team of developers, and community. We however acknowledge that our dataset would not be representative of all kinds of software projects, especially in commercial settings (although open source projects and commercial projects are similar in many aspects). One of the key differences between open source and commercial projects that may affect the estimation of story points is the nature of contributors, developers, and project's stakeholders. Further investigation for commercial agile projects is needed.

5.11 Implications

In this section, we discuss a number of implications of our results.

What do the results mean for the research on effort estimation? Existing work on effort estimation mainly focus on estimating the whole project with a small number of data points (see the datasets in the PROMISE repository [83] for example). The fast emergence of agile development demand

0098-5589 (c) 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2018.2792473, IEEE Transactions on Software Engineering

more research on estimation at the issue or user story level. Our work opens a new research area for the use of software analytics in estimating story points. The assertion demonstrated by our results is that our current method works and no other methods has been demonstrated to work at this scale of above 23,000 data points. Existing work in software effort estimation have dealt with a much smaller number of observations (i.e. data points) than our work did. For example, the China dataset has only 499 data points, Desharnais has 77, and Finish has 38 (see the datasets for effort estimation on the PROMISE repository) – these are commonly used in existing effort estimation work (e.g. [19, 84]). By contrast, in this work we deal with the scale of thousands of data points. Since we make our dataset publicly available, further research (e.g. modeling the codebase and adding team-specific features into the estimation model) can be advanced in this topic, and our current results can serve as the baseline.

Should we adopt deep learning? To the best of our knowledge, our work is the first major research in using deep learning for effort estimation. The use of deep learning has allowed us to automatically learn a good representation of an issue report and use this for estimating the effort of resolving the issue. The evaluation results demonstrates the significant improvement that our deep learning approach has brought in terms of predictive performance. This is a powerful result since it helps software practitioners move away from the manual feature engineering process. Feature engineering usually relies on domain experts who use their specific knowledge of the data to create features for machine learners to work. In our approach, features are automatically learned from a textual description of an issue, thus obviating the need for designing them manually. We of course need to collect the labels (i.e. story points assigned to issues) as the ground truths used for learning and testing. Hence, we believe that the wide adoption of software analytics in industry crucially depends on the ability to automatically derive (learn) features from raw software engineering data.

In our context of story point estimation, if the number of new words is large, transfer learning is needed, e.g. by using the existing model as a strong prior for the new model. However, this can be mitigated by pre-training on a large corpus so that most of the terms are covered. After pre-training, our model is able to automatically learn semantic relations between words. For example, words related to networking like “soap”, “configuration”, “tcp”, and “load” are in one cluster (see Figure 7). Hence, even when a user story has several unique terms (but already pre-trained), retraining the main model is not necessary. Pre-training may however take time and effort. One potential research direction is therefore building up a community for sharing pre-trained networks, which can be used for initialization, thus reducing training times (similar to Model Zoo [85]). As the first step towards this direction, we make our pre-trained models publicly available for the research community.

We acknowledged that the explainability of a model is important for full adoption of machine learning techniques. This is not a unique problem only for recurrent networks (RNN), but also for many powerful modern machine learning techniques (e.g. Random Forests and SVM). However, RNN is not entirely a black-box as it seems (e.g. see [86]). For example, word importance can be credited using various techniques (e.g., using gradient with respect to word value). Alternatively, there are model agnostic technique to explain any prediction [87]. Even with partly interpretable RNN, if the prediction is accurate, then we can still expect a high level of adoption.

What do the results mean for project managers and developers?

Our positive results indicate that it is possible to build a prediction system to support project managers and developers in estimating story points. Our proposal enables teams to be *consistent* in their estimation of story points. Achieving this consistency is central to effectively leveraging story points for project planning. The machine learner learns from past estimates made by the specific team which it is deployed to assist. The insights that the learner acquires are therefore *team-specific*. The intent is not to have the machine learner supplant existing agile estimation practices. The intent, instead, is to deploy the machine learner to *complement* these practices by playing the role of a decision support system. Teams would still meet, discuss user stories and generate estimates as per current practice, but would have the added benefit of access to the insights acquired by the machine learner. Teams would be free to reject the suggestions of the machine learner, as is the case with any decision support system. In every such estimation exercise, the actual estimates generated are recorded as data to be fed to the machine learner, independent of whether these estimates are based on the recommendations of the machine learner or not. This estimation process helps the team not only understand sufficient details about what it will take to to resolve those issues, but also align with their previous estimations.

6 Related work

Existing estimation methods can generally be classified into three major groups: expert-based, model-based, and hybrid approaches. Expert-based methods rely on human expertise to make estimations, and are the most popular technique in practice [88, 89]. Expert-based estimation however tends to require large overheads and the availability of experts each time the estimation needs to be made. Model-based approaches use data from past projects but they are also varied in terms of building customized models or using fixed models. The well-known construction cost (COCOMO) model [11] is an example of a fixed model where factors and their relationships are already defined. Such estimation models were built based on data from a range of past projects. Hence, they tend to be suitable only for a certain kinds of project that were used to build the model. The customized model building approach requires context-specific data and uses various methods such as regression (e.g. [12, 13]), Neural Network (e.g. [14, 90]), Fuzzy Logic (e.g. [15]), Bayesian Belief Networks (e.g.[16]), analogy-based (e.g. [17, 18]), and multi-objective evolutionary approaches (e.g. [19]). It is however likely that no single method will be the best performer for all project types [10, 20, 91]. Hence, some recent work (e.g. [20]) proposes to combine the estimates from multiple estimators. Hybrid approaches (e.g. [21, 22])

0098-5589 (c) 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2018.2792473, IEEE Transactions on Software Engineering

combine expert judgements with the available data – similarly to the notions of our proposal.

While most existing work focuses on estimating a whole project, little work has been done in building models specifically for agile projects. Today's agile, dynamic and change-driven projects require different approaches to planning and estimating [24]. Some recent approaches leverage machine learning techniques to support effort estimation for agile projects. Recently, the work in [64] proposed an approach which extracts TF-IDF features from issue description to develop an story-point estimation model. The univariate feature selection technique are then applied on the extracted features and fed into classifiers (e.g. SVM). In addition, the work in [92] applied Cosmic Function Points (CFP) [93] to estimate the effort for completing an agile project. The work in [94] developed an effort prediction model for iterative software development setting using regression models and neural networks. Differing from traditional effort estimation models, this model is built after each iteration (rather than at the end of a project) to estimate effort for the next iteration. The work in [95] built a Bayesian network model for effort prediction in software projects which adhere to the agile Extreme Programming method. Their model however relies on several parameters (e.g. process effectiveness and process improvement) that require learning and extensive fine tuning. Bayesian networks are also used in [96] to model dependencies between different factors (e.g. sprint progress and sprint planning quality influence product quality) in Scrum-based software development project in order to detect problems in the project. Our work specifically focuses on estimating issues with story points using deep learning techniques to automatically learn semantic features representing the actual meaning of issue descriptions, which is the key difference from previous work. Previous research (e.g. [97–100]) has also been done in predicting the elapsed time for fixing a bug or the delay risk of resolving an issue. However, effort estimation using story points is a more preferable practice in agile development.

LSTM has shown successes in many applications such as language models [35], speech recognition [36] and video analysis [37]. Our Deep-SE is a generic in which it maps text to a numerical score or a class, and can be used for other tasks, e.g. mapping a movie review to a score, or assigning scores to essays, or sentiment analysis. Deep learning has recently attracted increasing interests in software engineering. Our previous work [101] proposed a generic deep learning framework based on LSTM for modeling software and its development process. White *et. al.* [102] has employed recurrent neural networks (RNN) to build a language model for source code. Their later work [103] extended these RNN models for detecting code clones. The work in [104] also used RNNs to build a statistical model for code completion. Our recent work [105] used LSTM to build a language model for code and demonstrated the improvement of this model compared to the one using RNNs. Gu *et. al.* [106] used a special RNN Encoder–Decoder, which consists of an encoder RNN to process the input sequence and a decoder RNN with attention to generate the output sequence. This model takes as input a given API-related natural language query and returns API usage sequences. The work in [107] also uses RNN Encoder–Decoder but for fixing common errors in C programs. Deep Belief Network [108] is another common deep learning model, which has been used in software engineering, e.g. for building defection prediction models [109, 110].

7 Conclusion

In this paper, we have contributed to the research community the dataset for story point estimations, sourcing from 16 large and diverse software projects. We have also proposed a deep learning-based, *fully end-to-end* prediction system for estimating story points, removing the users from manually designing features from the textual description of issues. A key novelty of our approach is the combination of two powerful deep learning architectures: Long Short-Term Memory (to learn a vector representation for issue reports) and Recurrent Highway Network (for building a deep representation).

The proposed approach has consistently outperformed three common baselines and four alternatives according to our evaluation results. Compared against the Mean and Median techniques, the proposed approach has improved 34.06% and 26.77% respectively in MAE averaging across 16 projects we studied. Compared against the BoW and Doc2Vec techniques, our approach has improved 23.68% and 17.90% in MAE. These are significant results in the literature of effort estimation. A major part of those improvement were brought by our use of the deep learning LSTM architecture to model the textual description of an issue. The use of highway recurrent networks (on top of LSTM) has also improved the predictive performance, but not as significantly as the LSTM itself (especially for those project which have very small number of issues).

Our future work would involve expanding our study to commercial software projects and other large open source projects to further evaluate our proposed method. We also consider performing team analytics (e.g. features characterizing a team) to model team changes over time and feed it into our prediction model. We also plan to investigate how to learn a semantic representation of the codebase and use it as another input to our model. Furthermore, we will look into experimenting with a sliding window setting to explore incremental learning. In addition, we will also investigate how to best use the issue's metadata (e.g. priority and type) and still maintain the end-to-end nature of our entire model. Our future work also involve comparing our use of the LSTM model against other state-of-the-art models of natural language such as paragraph2vec [59] or Convolutional Neural Network [111]. We have discussed (informally) our work with several software developers who has been practising agile and estimating story points. They all agreed that our prediction system could be useful in practice. However, to make such a claim, we need to implement it into a tool and perform a user study. Hence, we would like to evaluate empirically the impact of our prediction system for story point estimation in practice by project managers and/or software developers. This would involve developing the model into a tool (e.g. a JIRA plugin) and then organising trial use in practice. This is an important part of our future work to confirm the ultimate benefits of our approach in general.

0098-5589 (c) 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2018.2792473, IEEE Transactions on Software Engineering

REFERENCES

[1] B. Michael, S. Blumberg, and J. Laartz, "Delivering large-scale IT projects on time, on budget, and on value," 2012. [Online]. Available: http://www.mckinsey.com/business-functions/digital-mckinsey/our-insights/delivering-large-scale-it-projects-on-time-on-budget-and-on-value

[2] B. Flyvbjerg and A. Budzier, "Why Your IT Project May Be Riskier Than You Think," *Harvard Business Review*, vol. 89, no. 9, pp. 601–603, 2011.

[3] A. Trendowicz and R. Jeffery, *Software project effort estimation: Foundations and best practice guidelines for success*. Springer, 2014.

[4] L. C. Briand and I. Wieczorek, "Resource estimation in software engineering," *Encyclopedia of software engineering*, 2002.

[5] E. Kocaguneli, A. T. Misirli, B. Caglayan, and A. Bener, "Experiences on developer participation and effort estimation," in *Software Engineering and Advanced Applications (SEAA), 2011 37th EUROMICRO Conference on*. IEEE, 2011, pp. 419–422.

[6] M. Jorgensen, "What we do and don't know about software development effort estimation," *IEEE software*, vol. 31, no. 2, pp. 37–40, 2014.

[7] S. McConnell, *Software estimation: demystifying the black art*. Microsoft press, 2006.

[8] T. Menzies, Z. Chen, J. Hihn, and K. Lum, "Selecting best practices for effort estimation," *IEEE Transactions on Software Engineering*, vol. 32, no. 11, pp. 883–895, 2006.

[9] I. Sommerville, *Software Engineering*, 9th ed. Pearson, 2010.

[10] M. Jørgensen and M. Shepperd, "A Systematic Review of Software Development Cost Estimation Studies," *IEEE Transactions on Software Engineering*, vol. 33, no. 1, pp. 33–53, 2007.

[11] B. W. Boehm, R. Madachy, and B. Steece, *Software cost estimation with Cocomo II*. Prentice Hall PTR, 2000.

[12] P. Sentas, L. Angelis, and I. Stamelos, "Multinomial Logistic Regression Applied on Software Productivity Prediction," in *Proceedings of the 9th Panhellenic conference in informatics*, 2003, pp. 1–12.

[13] P. Sentas, L. Angelis, I. Stamelos, and G. Bleris, "Software productivity and effort prediction with ordinal regression," *Information and Software Technology*, vol. 47, no. 1, pp. 17–29, 2005.

[14] S. Kanmani, J. Kathiravan, S. S. Kumar, M. Shanmugam, and P. E. College, "Neural Network Based Effort Estimation using Class Points for OO Systems," *Evaluation*, 2007.

[15] S. Kanmani, J. Kathiravan, S. S. Kumar, and M. Shanmugam, "Class point based effort estimation of OO systems using Fuzzy Subtractive Clustering and Artificial Neural Networks," *Proceedings of the 1st India Software Engineering Conference (ISEC)*, pp. 141–142, 2008.

[16] S. Bibi, I. Stamelos, and L. Angelis, "Software cost prediction with predefined interval estimates," in *Proceedings of the First Software Measurement European Forum*, Rome, Italy, 2004, pp. 237–246.

[17] M. Shepperd and C. Schofield, "Estimating Software Project Effort Using Analogies," *IEEE Transactions on Software Engineering*, vol. 23, no. 12, pp. 736–743, 1997.

[18] L. Angelis and I. Stamelos, "A Simulation Tool for Efficient Analogy Based Cost Estimation," *Empirical Software Engineering*, vol. 5, no. 1, pp. 35–68, 2000.

[19] F. Sarro, A. Petrozziello, and M. Harman, "Multi-objective Software Effort Estimation," in *Proceedings of the 38th International Conference on Software Engineering (ICSE)*, 2016, pp. 619–630.

[20] E. Kocaguneli, T. Menzies, and J. W. Keung, "On the value of ensemble effort estimation," *IEEE Transactions on Software Engineering*, vol. 38, no. 6, pp. 1403–1416, 2012.

[21] R. Valerdi, "Convergence of expert opinion via the wideband delphi method: An application in cost estimation models," 2011.

[22] S. Chulani, B. Boehm, and B. Steece, "Bayesian analysis of empirical software engineering cost models," *IEEE Transactions on Software Engineering*, vol. 25, no. 4, pp. 573–583, 1999.

[23] H. F. Cervone, "Understanding agile project management methods using Scrum," *OCLC Systems & Services: International digital library perspectives*, vol. 27, no. 1, pp. 18–22, 2011.

[24] M. Cohn, *Agile estimating and planning*. Pearson Education, 2005.

[25] 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 (PROMISE)*, 2014, pp. 82–91.

[26] Y. Shin, A. Meneely, L. Williams, and J. A. Osborne, "Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities," *IEEE Transactions on Software Engineering*, vol. 37, no. 6, pp. 772–787, 2011.

[27] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy, "Cross-project defect prediction: a large scale experiment on data vs. domain vs. process," in *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*. ACM, 2009, pp. 91–100.

[28] Atlassian, "Atlassian JIRA Agile software," 2016. [Online]. Available: https://www.atlassian.com/software/jira

[29] Spring, "Spring XD issue XD-2970," 2016. [Online]. Available: https://jira.spring.io/browse/XD-2970

[30] J. Grenning, *Planning poker or how to avoid analysis paralysis while release planning*, 2002, vol. 3.

[31] M. Choetkiertikul, H. K. Dam, T. Tran, A. Ghose, and J. Grundy, "Predicting Delivery Capability in Iterative Software Development," *IEEE Transactions on Software Engineering*, vol. 14, no. 8, pp. 1–1, 2017. [Online]. Available: http://ieeexplore.ieee.org/document/7898472/

[32] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[33] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with lstm," *Neural computation*, vol. 12, no. 10, pp. 2451–2471, 2000.

[34] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber, "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies," 2001.

[35] M. Sundermeyer, R. Schlüter, and H. Ney, "LSTM Neural Networks for Language Modeling," in *INTERSPEECH*, 2012, pp. 194–197.

[36] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE, 2013, pp. 6645–6649.

[37] J. Yue-Hei Ng, M. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga, and G. Toderici, "Beyond short snippets: Deep networks for video classification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 4694–4702.

[38] T. Pham, T. Tran, D. Phung, and S. Venkatesh, "Faster training of very deep networks via p-norm gates," *The 23rd International Conference on Pattern Recognition*, 2016, To Appear.

[39] ——, "Predicting healthcare trajectories from medical records: A deep learning approach," *Journal of Biomedical Informatics*, vol. 69, pp. 218–229, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1532046417300710

[40] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1026–1034.

[41] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *Signal Processing Magazine, IEEE*, vol. 29, no. 6, pp. 82–97, 2012.

[42] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in Neural Information Processing Systems*, 2014, pp. 3104–3112.

[43] A. Kumar, O. Irsoy, J. Su, J. Bradbury, R. English, B. Pierce, P. Ondruska, I. Gulrajani, and R. Socher, "Ask me anything: Dynamic memory networks for natural language processing," *arXiv preprint arXiv:1506.07285*, 2015.

[44] Y. Bengio, "Learning deep architectures for AI," *Foundations and trends® in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

[45] S. Liang and R. Srikant, "Why Deep Neural Networks for Function Approximation?" 2016.

[46] H. Mhaskar, Q. Liao, and T. A. Poggio, "When and Why Are Deep Networks Better Than Shallow Ones?" in *AAAI*, 2017, pp. 2343–2349.

[47] M. Raghu, B. Poole, J. Kleinberg, S. Ganguli, and J. Sohl-Dickstein, "On the expressive power of deep neural networks," in *ICML*, 2017.

[48] G. F. Montufar, R. Pascanu, K. Cho, and Y. Bengio, "On the number of linear regions of deep neural networks," in *Advances in neural information processing systems*, 2014, pp. 2924–2932.

[49] M. Bianchini and F. Scarselli, "On the complexity of neural network classifiers: A comparison between shallow and deep architectures," *IEEE transactions on neural networks and learning systems*, vol. 25, no. 8, pp. 1553–1565, 2014.

[50] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Training very deep networks," *arXiv preprint arXiv:1507.06228*, 2015.

[51] J. Schmidhuber, "Deep Learning in neural networks: An

0098-5589 (c) 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2018.2792473, IEEE Transactions on Software Engineering

overview," *Neural Networks*, vol. 61, pp. 85–117, 2015. [Online]. Available: http://dx.doi.org/10.1016/j.neunet.2014.09.003

[52] M. U. Gutmann and A. Hyvärinen, "Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics," *Journal of Machine Learning Research*, vol. 13, no. Feb, pp. 307–361, 2012.

[53] T. D. Team, "Theano: A {Python} framework for fast computation of mathematical expressions," *arXiv e-prints*, vol. abs/1605.0, 2016. [Online]. Available: http://deeplearning.net/software/theano

[54] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

[55] M. Shepperd and S. MacDonell, "Evaluating prediction systems in software project estimation," *Information and Software Technology*, vol. 54, no. 8, pp. 820–827, 2012. [Online]. Available: http://dx.doi.org/10.1016/j.infsof.2011.12.008

[56] R. Moraes, J. F. Valiati, and W. P. Gavião Neto, "Document-level sentiment classification: An empirical comparison between SVM and ANN," *Expert Systems with Applications*, vol. 40, no. 2, pp. 621–633, 2013.

[57] P. A. Whigham, C. A. Owen, and S. G. Macdonell, "A Baseline Model for Software Effort Estimation," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 24, no. 3, p. 20, 2015.

[58] P. Tirilly, V. Claveau, and P. Gros, "Language modeling for bag-of-visual words image categorization," in *Proceedings of the 2008 international conference on Content-based image and video retrieval*, 2008, pp. 249–258.

[59] Q. Le and T. Mikolov, "Distributed Representations of Sentences and Documents," in *Proceedings of the 31st International Conference on Machine Learning (ICML)*, vol. 32, 2014, pp. 1188–1196.

[60] E. Kocaguneli, S. Member, and T. Menzies, "Exploiting the Essential Assumptions of Analogy-Based Effort Estimation," vol. 38, no. 2, pp. 425–438, 2012.

[61] E. Kocaguneli, T. Menzies, and E. Mendes, "Transfer learning in effort estimation," *Empirical Software Engineering*, vol. 20, no. 3, pp. 813–843, 2015.

[62] E. Mendes, I. Watson, and C. Triggs, "A Comparative Study of Cost Estimation Models for Web Hypermedia Applications," *Empirical Software Engineering*, vol. 8, pp. 163–196, 2003.

[63] Y. F. Li, M. Xie, and T. N. Goh, "A Study of Project Selection and Feature Weighting for Analogy Based Software Cost Estimation," *J. Syst. Softw.*, vol. 82, no. 2, pp. 241–252, feb 2009. [Online]. Available: http://dx.doi.org/10.1016/j.jss.2008.06.001

[64] S. Porru, A. Murgia, S. Demeyer, and M. Marchesi, "Estimating Story Points from Issue Reports," 2016.

[65] S. D. Conte, H. E. Dunsmore, and V. Y. Shen, *Software Engineering Metrics and Models*. Redwood City, CA, USA: Benjamin-Cummings Publishing Co., Inc., 1986.

[66] T. Foss, E. Stensrud, B. Kitchenham, and I. Myrtveit, "A simulation study of the model evaluation criterion MMRE," *IEEE Transactions on Software Engineering*, vol. 29, no. 11, pp. 985–995, 2003.

[67] B. Kitchenham, L. Pickard, S. MacDonell, and M. Shepperd, "What accuracy statistics really measure," *IEE Proceedings - Software*, vol. 148, no. 3, p. 81, 2001.

[68] M. Korte and D. Port, "Confidence in software cost estimation results based on MMRE and PRED," *Proceedings of the 4th international workshop on Predictor models in software engineering (PROMISE)*, pp. 63–70, 2008.

[69] D. Port and M. Korte, "Comparative studies of the model evaluation criterions mmre and pred in software cost estimation research," in *Proceedings of the 2nd ACM-IEEE international symposium on Empirical software engineering and measurement*. ACM, 2008, pp. 51–60.

[70] T. Menzies, E. Kocaguneli, B. Turhan, L. Minku, and F. Peters, *Sharing data and models in software engineering*. Morgan Kaufmann, 2014.

[71] K. Muller, "Statistical power analysis for the behavioral sciences," *Technometrics*, vol. 31, no. 4, pp. 499–500, 1989.

[72] H. H. Abdi, "The Bonferonni and Sidak Corrections for Multiple Comparisons," *Encyclopedia of Measurement and Statistics.*, vol. 1, pp. 1–9, 2007. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.78.8747{\&}rep=rep1{\&}type=pdf

[73] A. Vargha and H. D. Delaney, "A Critique and Improvement of the CL Common Language Effect Size Statistics of McGraw and Wong," *Journal of Educational and Behavioral Statistics*, vol. 25, no. 2, pp. 101–132, 2000. [Online]. Available: http://jeb.sagepub.com/cgi/doi/10.3102/10769986025002101

[74] A. Arcuri and L. Briand, "A practical guide for using statistical tests to assess randomized algorithms in software engineering," in *Proceedings of the 33rd International Conference on Software Engineering (ICSE)*, 2011, pp. 1–10.

[75] L. van der Maaten and G. Hinton, "Visualizing high-dimensional data using t-sne," *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, Nov 2008.

[76] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio, "Why does unsupervised pre-training help deep learning?" *J. Mach. Learn. Res.*, vol. 11, pp. 625–660, Mar. 2010.

[77] J. Weston, F. Ratle, and R. Collobert, "Deep learning via semi-supervised embedding," in *Proceedings of the 25th International Conference on Machine Learning*, ser. ICML '08. New York, NY, USA: ACM, 2008, pp. 1168–1175. [Online]. Available: http://doi.acm.org/10.1145/1390156.1390303

[78] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," *J. Mach. Learn. Res.*, vol. 12, pp. 2493–2537, Nov. 2011. [Online]. Available: http://dl.acm.org/citation.cfm?id=1953048.2078186

[79] D. Zwillinger and S. Kokoska, *CRC standard probability and statistics tables and formulae*. Crc Press, 1999.

[80] J. McCarthy, "From here to human-level AI," *Artificial Intelligence*, vol. 171, no. 18, pp. 1174–1182, 2007.

[81] A. Arcuri and L. Briand, "A Hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering," *Software Testing, Verification and Reliability*, vol. 24, no. 3, pp. 219–250, 2014.

[82] T. Menzies and M. Shepperd, "Special issue on repeatable results in software engineering prediction," *Empirical Software Engineering*, vol. 17, no. 1-2, pp. 1–17, 2012.

[83] T. Menzies, B. Caglayan, Z. He, E. Kocaguneli, J. Krall, F. Peters, and B. Turhan, "The PROMISE Repository of empirical software engineering data," 2012.

[84] P. L. Braga, A. L. I. Oliveira, and S. R. L. Meira, "Software Effort Estimation using Machine Learning Techniques with Robust Confidence Intervals," in *Proceedings of the 7th International Conference on Hybrid Intelligent Systems (HIS)*, 2007, pp. 352–357.

[85] T. Jia, Yangqing and Shelhamer, Evan and Donahue, Jeff and Karayev, Sergey and Long, Jonathan and Girshick, Ross and Guadarrama, Sergio and Darrell, "Caffe: Convolutional Architecture for Fast Feature Embedding," *arXiv preprint arXiv:1408.5093*, 2014.

[86] A. Karpathy, J. Johnson, and L. Fei-Fei, "Visualizing and Understanding Recurrent Networks," in *arXiv preprint arXiv:1506.02078*, 2015, pp. 1–12. [Online]. Available: http://arxiv.org/abs/1506.02078

[87] M. T. Ribeiro, S. Singh, and C. Guestrin, ""Why Should I Trust You?": Explaining the Predictions of Any Classifier," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2016, pp. 1135–1144. [Online]. Available: http://arxiv.org/abs/1602.04938

[88] M. Jorgensen, "A review of studies on expert estimation of software development effort," *Journal of Systems and Software*, vol. 70, no. 1-2, pp. 37–60, 2004.

[89] M. Jorgensen and T. M. Gruschke, "The impact of lessons-learned sessions on effort estimation and uncertainty assessments," *IEEE Transactions on Software Engineering*, vol. 35, no. 3, pp. 368–383, 2009.

[90] A. Panda, S. M. Satapathy, and S. K. Rath, "Empirical Validation of Neural Network Models for Agile Software Effort Estimation based on Story Points," *Procedia Computer Science*, vol. 57, pp. 772–781, 2015.

[91] F. Collopy, "Difficulty and complexity as factors in software effort estimation," *International Journal of Forecasting*, vol. 23, no. 3, pp. 469–471, 2007.

[92] R. D. A. Christophe Commeyne, Alain Abran, "Effort Estimation with Story Points and COSMIC Function Points - An Industry Case Study," pp. 25–36, 2008. [Online]. Available: http://cosmic-sizing.org/wp-content/uploads/2016/03/Estimation-model-v-Print-Format-adapter.pdf

[93] C. Group, *INTERNATIONAL STANDARD ISO/IEC Software engineering COSMIC: a functional size measurement method*, 2011, vol. 2011.

[94] P. Abrahamsson, R. Moser, W. Pedrycz, A. Sillitti, and G. Succi, "Effort prediction in iterative software development processes – incremental versus global prediction models," *1st International Symposium on Empirical Software Engineering and Measurement*

0098-5589 (c) 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2018.2792473, IEEE Transactions on Software Engineering

(ESEM), pp. 344–353, 2007.
[95] P. Hearty, N. Fenton, D. Marquez, and M. Neil, "Predicting Project Velocity in XP Using a Learning Dynamic Bayesian Network Model," *IEEE Transactions on Software Engineering*, vol. 35, no. 1, pp. 124–137, 2009.
[96] M. Perkusich, H. De Almeida, and A. Perkusich, "A model to detect problems on scrum-based software development projects," *The ACM Symposium on Applied Computing*, pp. 1037–1042, 2013.
[97] E. Giger, M. Pinzger, and H. Gall, "Predicting the fix time of bugs," in *Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering (RSSE)*. ACM, 2010, pp. 52–56.
[98] L. D. Panjer, "Predicting Eclipse Bug Lifetimes," in *Proceedings of the 4th International Workshop on Mining Software Repositories (MSR)*, 2007, pp. 29–32.
[99] P. Bhattacharya and I. Neamtiu, "Bug-fix time prediction models: can we do better?" in *Proceedings of the 8th working conference on Mining software repositories (MSR)*. ACM, 2011, pp. 207–210.
[100] P. Hooimeijer and W. Weimer, "Modeling bug report quality," in *Proceedings of the 22 IEEE/ACM international conference on Automated software engineering (ASE)*. ACM Press, nov 2007, pp. 34 – 44.
[101] H. K. Dam, T. Tran, J. Grundy, and A. Ghose, "DeepSoft: A vision for a deep model of software," in *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE '16. ACM, To Appear., 2016.
[102] M. White, C. Vendome, M. Linares-v, and D. Poshyvanyk, "Toward Deep Learning Software Repositories," in *Proceedings of the 12th Working Conference on Mining Software Repositories (MSR)*, 2015, pp. 334–345.
[103] M. White, M. Tufano, C. Vendome, and D. Poshyvanyk, "Deep learning code fragments for code clone detection," in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE 2016. New York, NY, USA: ACM, 2016, pp. 87–98. [Online]. Available: http://doi.acm.org/10.1145/2970276.2970326
[104] V. Raychev, M. Vechev, and E. Yahav, "Code completion with statistical language models," *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation - PLDI '14*, pp. 419–428, 2013. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2594291.2594321
[105] H. Dam, T. Tran, and T. Pham, "A deep language model for software code," *arXiv preprint arXiv:1608.02715*, no. August, pp. 1–4, 2016. [Online]. Available: http://arxiv.org/abs/1608.02715
[106] X. Gu, H. Zhang, D. Zhang, and S. Kim, "Deep API Learning," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE 2016. New York, NY, USA: ACM, 2016, pp. 631–642. [Online]. Available: http://doi.acm.org/10.1145/2950290.2950334
[107] R. Gupta, S. Pal, A. Kanade, and S. Shevade, "Deepfix: Fixing common C language errors by deep learning," in *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*. AAAI Press, 2017, pp. 1345–1351. [Online]. Available: http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14603
[108] G. Hinton and R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504 – 507, 2006.
[109] S. Wang, T. Liu, and L. Tan, "Automatically learning semantic features for defect prediction," in *Proceedings of the International Conference on Software Engineering (ICSE)*, vol. 14-22, 2016, pp. 297–308. [Online]. Available: http://dx.doi.org/10.1145/2884781.2884804
[110] X. Yang, D. Lo, X. Xia, Y. Zhang, and J. Sun, "Deep Learning for Just-in-Time Defect Prediction," in *Proceedings of the IEEE International Conference on Software Quality, Reliability and Security (QRS)*, no. 1, 2015, pp. 17–26.
[111] N. Kalchbrenner, E. Grefenstette, and P. Blunsom, "A Convolutional Neural Network for Modelling Sentences," *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 655–665, 2014.

PLACE PHOTO HERE

Morakot Choetkiertikul Biography text here.

PLACE PHOTO HERE

Hoa Khanh Dam Biography text here.

PLACE PHOTO HERE

Truyen Tran Biography text here.

PLACE PHOTO HERE

Trang Pham Biography text here.

PLACE PHOTO HERE

Aditya Ghose Biography text here.

0098-5589 (c) 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2018.2792473, IEEE Transactions on Software Engineering



Tim Menzies Biography text here.

0098-5589 (c) 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.