A Hybrid Model for Task Completion Effort Estimation

Ali Dehghan
University of Victoria
Canada
dehghan@uvic.ca

Kelly Blincoe
University of Auckland
New Zealand
kblincoe@acm.org

Daniela Damian
University of Victoria
Canada
danielad@cs.uvic.ca

ABSTRACT

Predicting time and effort of software task completion has been an active area of research for a long time. Previous studies have proposed predictive models based on either text data or metadata of software tasks to estimate either completion time or completion effort of software tasks, but there is a lack of focus in the literature on integrating all sets of attributes together to achieve better performing models.

We first apply the previously proposed models on the datasets of two IBM commercial projects called RQM and RTC to find the best performing model in predicting task completion effort on each set of attributes. Then we propose an approach to create a hybrid model based on selected individual predictors to achieve more accurate and stable results in early prediction of task completion effort and to make sure the model is not bounded to some attributes and consequently is adoptable to a larger number of tasks. Categorizing task completion effort values into *Low* and *High* labels based on their measured median value, we show that our hybrid model provides 3-8% more accuracy in early prediction of task completion effort compared to the best individual predictors.

CCS Concepts

•Software and its engineering → Software development process management; Software defect analysis;

Keywords

Mining software repositories; machine learning; ensemble learning; task completion effort; effort estimation

1. INTRODUCTION

Early estimation of required effort to complete tasks is a common practice in software companies as it helps managers in planning and allocation of resources. A common approach in industry is to base these estimates on expert knowledge [17], however that creates extra work for the developers. Additionally, there is an optimism and a tendency toward underestimation of effort particularly in cost-competitive situations [11]. Automatic techniques to predict task completion effort can reduce the burden on developers and remove biases.

Many previous studies have proposed predictive models to estimate either completion time [5, 1, 16, 14] or completion effort [19, 17] of software development tasks and defects. The effort needed to complete a task (i.e. person-hours required to complete a task) does not always correlate with the calendar time needed to complete the task. For example, a simple task which requires only a few hours to complete may be postponed for months due to its low priority [14].

Previously proposed models use various metadata based attributes, i.e. non-text attributes, of tasks to predict task completion time or effort. The attributes vary in when they are available during the task life cycle. Some attributes, like *Reporter* and *Creation date*, are available very early when the task is created. Other attributes, like *Number of commits* and *Number of subscribers*, are not fully known until the task is complete. Some models also perform text analysis techniques such as document similarity analysis on text based attributes, like *Title* and *Description*. The use of text analysis has been more popular in predicting effort rather than completion time.

While many models have been proposed, none have been able to achieve accuracy that rivals predictions made by developers with expert knowledge. Previous literature suggests that the average effort estimates from different sources is likely to be more accurate than most individual effort estimates [11], but no studies have investigated this claim. Therefore, in this study, we combine the most accurate of the previously proposed models to produce a new hybrid model with greater accuracy. The hybrid model uses three independent attribute sets (1) early metadata based attributes, (2) title and (3) description of software tasks. The model is also adoptable to a larger number of tasks compared to previously proposed models, as it is not limited to only one type of data source that might not be available in all circumstances.

For this study, we analyzed two commercial projects of IBM called RQM and RTC. The two projects have a total of 61,729 work items, i.e. assignable and traceable units of work, of type *Task*.

To make the predictions, following the approach proposed by Giger et al. [5], we discretized effort values, known as *Time spent* in our datasets, into two binary labels based on the median value of time spent of the tasks. Label *Low* that

SWAN'16, November 13, 2016, Seattle, WA, USA
© 2016 ACM. 978-1-4503-4395-4/16/11...
http://dx.doi.org/10.1145/2989238.2989242

represents those having a time spent value less than median and label *High* for time spent values greater than the median value. Having this structure for the dependent variable, the prediction problem is transformed to a binary classification problem.

2. RELATED WORK

Many previous works have attempted to increase prediction accuracy of both task resolution time and effort by proposing different models and in some cases introducing new attributes, but there has been a lack of focus on investigating the feasibility of integrating the proposed attributes and combining previous models while improving prediction results. In this section, we will introduce some of the most notable studies that have either proposed the use of a new algorithm or method, a set of new attributes or had some new findings that led us in this study.

Weiss et al. [19] vectorized *Title* and *Description* of bugs in JBoss project and used kNN classifier to find most similar bugs based on text similarity techniques and used the average fix effort of similar bugs as the prediction of fix effort for each new bug and as a result, around 30% of predictions lied within a ±50% range of the actual effort. They also introduced α-kNN by adding the concept of thresholds to kNN to dismiss similarity scores less than a certain minimum threshold and classify the corresponding bugs as *Unknown* to increase the prediction accuracy. This way prediction accuracy improved specially for large α values, but at the same time it was possible to make predictions for fewer issues as α value of 0.9 left fix effort of 87% of issues unpredictable.

Pfahl et al. [17] looked into text based attributes of bugs of a commercial project of an Estonian software company and applied Spherical k-means clustering technique instead of kNN to find similar issues and predicted fix effort of bugs based on certain time intervals. They achieved up to 64% accuracy when accepting up to 0.5 hours error as correct classification and obtained up to 73% accuracy when accepting 1 hour error, however these numbers were based on the bugs that had both *Title* and *Description* and also the presented results were based on the last 50 and 200 issues of the dataset and prediction quality of models involving the whole dataset were not presented, although mentioned that was not as good.

Giger et al. [5] made a comparison of models using pre-submission data and models that use post-submission data by analyzing snapshots of bugs in certain time intervals to classify bugs into two groups of *Fast* and *Slow* based on median value of bug fix time. Performing analysis using CHAID Decision Tree learner on several open-source projects, Eclipse, Mozilla and Gnome, they achieved between 60-70% accuracy using pre-submission data and about 5-10% higher accuracy values when including post-submission data in the models.

Hewett and Kijsanayothin [9] used several supervised learners on metadata based attributes of the dataset of a medical software system to classify defect repair time into three groups of *Low*, *Medium* and *High*. They used algorithms like SVM which was not adopted in previous work, but they achieved the best results using C4.5 Decision Tree algorithm. They reported accuracy numbers as high as 94%, however post-assignment data including attributes such as *Assignee* and *Assign date* were included in their models.

Marks et al. [14] made use of Random Forest learner on bugs of Eclipse and Mozilla projects. They used many different attributes from three dimensions of *Location*, *Reporter* and *Description* of bugs and discretized fix times into several intervals and achieved 65% classification accuracy. They also performed some analysis particularly to measure the importance of the attributes in the three different dimensions to classification performance and showed that attributes affecting bug fix time vary between projects and also vary over time within the same project.

Abdelmoez et al. [1] used Naive Bayes classifier in order to predict resolution time of issues of the same open source projects as Giger et al. [5] studied using the same metadata based attributes. They made three sets of binary labels, *Fast* vs *Slow* based on median value of resolution time and also *Very fast* vs *Not very fast* and *Very slow* vs *Not very slow* based on the first and last quartile of resolution time of bugs respectively. For the *Slow* vs *Fast* binary classification they achieved precision values between 57% and 64% and for the other two sets of labels, precision values were much lower for the minority classes.

Lucas D. Panjer [16] applied 1-R, Decision Tree, Naive Bayes and Logistic Regression learners on early attributes of Eclipse bugs to classify them into certain intervals of resolution time. The results were not appealing as he achieved classification accuracies varying from 31.0% to 34.9%.

In an empirical study conducted based on datasets of Windows Vista and Windows 7, Guo et al. [7] showed that bugs reported by people with higher reputations are more likely to get fixed, however in another study by Bhattacharya and Neamtiu [3] on three open source projects, such correlation was not found confirming that many attributes and models proposed for task completion effort and time prediction are highly context dependent.

Despite the absence of using ensemble learners for task completion effort estimation, hybrid models have been successfully applied in other areas of software engineering research [15, 18, 12]. This study aims to make use of the findings of previous work and fill the research gap by using their achievements as inputs of a new analysis to build a hybrid solution on top of the approaches they have proposed.

3. TASK EFFORT PREDICTION

In this section, we describe our research method, introduce our datasets as well as the attributes that we use in our analysis. Then, we report the performance of some previously proposed stand-alone models based on our own datasets and finally propose our own approach to create a hybrid model.

3.1 Research Method

As the aim of this study was constructing a combined model of individual independent models proposed in previous work in order to enhance task completion effort prediction accuracy at early points of a task lifetime, i.e. before the task is assigned to a developer. We identified the best performing models based on the previous literature and applied the models on our own datasets.

According to literature, kNN classification and k-means clustering algorithms on text based attributes and Naive Bayes, Logistic Regression, SVM, Decision Tree and Random Forest learners on metadata based attributes provided most promising results. Therefore, the models proposed based on these algorithms in previous work were applied on our own datasets. We did not modify the core settings of the

proposed models except for (1) removing post-assignment attributes to comply with the aim of making prediction before a task is assigned to a developer (2) avoiding to follow settings that suggested to exclude some tasks from analysis or label them as *Unknown* to comply with the aim of having a model adoptable to all tasks.

We planned to create a hybrid model to increase the accuracy and stability of prediction results, therefore we needed several well-performing independent models. We had three independent attribute sets (1) a text based attribute called *Summary* (2) a text based attribute called *Description* and (3) a set of metadata based attributes, thus we selected those models which provided the best performance on each of these attribute sets. Finally we explored several approaches to integrate attribute sets and combine individual models. We started off by simply joining all attributes together and trained a model providing them as the input. Then we adopted some ensemble learning techniques to combine the individual models and train meta learners by supplying the prediction vote and confidence of individual models as input variables. In the end, we compared the results of hybrid models and examined their performance based on data of two commercial projects.

We used a variety of tools and libraries for our analysis, however RapidMiner [10] and WEKA [8] were the major ones. The evaluation method that we used for all the models we trained including the hybrid models was m-fold cross-validation with m=10. Accuracy was used as the performance measure of all trained models. Accuracy represents number of correctly classified test items divided by the size of test set.

3.2 The Dataset

For this study we analyzed two commercial projects of IBM called RQM and RTC having a total number of 61,729 work items of type *Task* as of May 29, 2016. These two projects are developed and maintained by two completely different IBM teams, headquartered in different locations.

RQM, Rational Quality Manager, is a collaborative application lifecycle management environment for test planning, construction, and execution. The project was started in June 2007, had a total number of 54,478 work items as of May 29, 2016 and its development team is headquartered in Raleigh, NC.

RTC, Rational Team Concert, is a collaborative software development environment that integrates planning, work item tracking, product builds, source control and reporting. The project was started in June 2005, had a total number of 173,549 work items as of May 29, 2016 and its development team is headquartered in Ottawa, ON.

We chose to study two different projects as training and validation on a single project pose a risk of bias in feature selection and overfitting to the context of project [11].

Based on the design of our study, we removed the work items that did not fit our study settings. The ones that were kept fulfilled the criteria presented in table 1.

The *Time spent* values in both datasets are measured in resolution of 1 minute and the values vary from 1 minute to 400 hours. The mean value of *Time spent* in RQM is 795 minutes and in RTC is 734 minutes and the median value is 300 minutes in both projects.

Following the approach Giger et al. [5] adopted, we discretized task completion effort into two binary labels based on the median value of *Time spent*. This approach was a good fit to the exploratory nature of this study and future research will look to provide more granular estimates.

Table 1: Work Item Selection Criteria

Criteria	#(RQM)	#(RTC)
Created before 2016-05-29	54,478	173,549
Type is Task	9,029	52,250
Status in (Resolved, Closed)	8,256	47,327
TimeSpent is not Null	5,522	3,968
TimeSpent is greater than 0	5,477	3,941

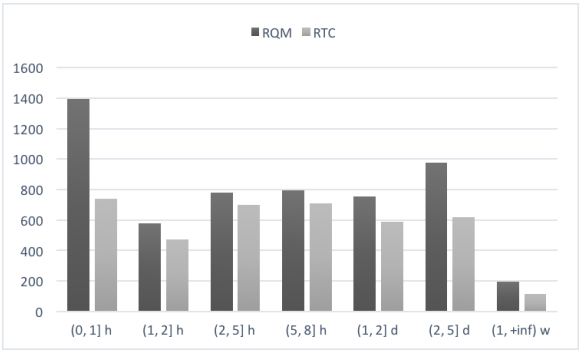

Figure 1: Distribution of *Time spent* values

$$completionEffort = \begin{cases} Low, & \text{if } TimeSpent \leq median \\ High, & \text{otherwise} \end{cases}$$

Label *Low* represents those tasks having a *Time spent* value less than median and label *High* represents *Time spent* values greater than the median value. This way we tackled a binary classification problem.

3.3 The Attributes

In this section, we introduce the attributes that we used in our models. All these attributes have been proposed in previous work . As our aim was to make early predictions of task completion effort, we only use the attributes that are usually available before the task is assigned to a developer. In summary these are the attributes that (1) have been proposed in some models proposed in previous work (2) are available in our datasets (3) are most of the time available in early stages of task creation, in particular, before the task is assigned to a developer. Some of these attributes have been referred to with different names in the literature than our datasets. Table 2 provides a list of these attributes, a short explanation for them and their common alternative name(s) in the literature.

3.4 Metadata Based Models

Among the algorithms that have been utilized in previous work for models predicting task completion time and effort that perform based on metadata based attributes, we selected the best-performing ones according to reported results and applied them on our datasets using the same attribute sets suggested by the authors. In this section, we present the results that we obtained for each selected model.

Table 2: Explanation of Dataset Attributes

Attribute Name	Alternative Name(s)	Explanation
Summary	Title	A half-line abstract of the problem
Description	-	What causes the problem, how the problem could be reproduced and in some cases some early instructions on how to solve the problem
Severity	-	How strong the impact is on the user
Priority	-	How important the problem is from managers/developers perspective
Creator	Reporter	The developer who created the task
Creation year	Year opened	In what year the task was created
Creation month	Month opened	In which month the task was created
Creation week	-	In what week of the year the task was created
How found	Testing phase	The testing phase in which the corresponding problem was found
Found in	Version	The product release in which the problem was found
Filed against	Component	The component to which the task corresponds
Planned for	Milestone	The iteration for which it was planned
Time spent	Resolution time, fix time, repair time, fix effort	The actual effort, time or man-hour spent to complete the task, with resolution of 1 minute

3.4.1 OneR Classifier

Following the idea of Panjer [16] we applied OneR classifier to find the most strongly correlated attribute with *Time spent* and have a baseline for the accuracy of our further predictive models. The resulting model showed that *Reporter* is the most important attribute in our datasets when predicting *Time spent*.

3.4.2 Naive Bayes classifier

Following the study of Abdolmoez et al. [1] adopting Naive Bayes classifier, we used their proposed attributes that were available in our datasets before task assignment. The equivalent attributes in our datasets are named *Creator*, *Priority*, *Severity*, *Creation year*, *Creation month*, *Planned for* and *Filed against*. Performing binary classification using Naive Bayes classifier, we achieved 68.27% accuracy on RQM project and 64.88% accuracy on RTC.

3.4.3 Logistic Regression

Following the proposal of Panjer [16] to apply Logistic regression learner using *Priority*, *Severity*, *Filed against*, *Found in*, *Planned for* and *Reporter*, we achieved 68.30% and 64.37% accuracy numbers for RQM and RTC datasets respectively. The suggestion in [17] to adopt Ordered Logistic Regression to improve results was not appropriate to our datasets as we were not performing multiclass classification.

3.4.4 Support Vector Machine

SVM has been successfully adopted in predictive analysis of many software engineering studies including studies on defect prediction [6, 2, 4]; nevertheless, the only work we found utilizing SVM learner for the particular purpose of task completion time prediction was the study of Hewett and Kijsanayothin [9]. They applied many learning algorithms; however, SVM was not among the promising ones according to the results. Therefore we decided to apply this algorithm using all metadata based attributes mentioned in subsection 3.2. The achieved classification accuracies were fair, 69.22% and 64.45% for RQM and RTC projects respectively.

3.4.5 C4.5 Decision Tree

We also applied C4.5 Decision Tree algorithm using attribute sets suggested in [9] and [16] separately. Using equivalent attributes suggested in [9], *How found*, *Filed against*, *Severity*, *Reporter*, *Creation year* and *Creation month* provided better results. Accuracy numbers were 68.72% and 65.30% for RQM and RTC respectively.

3.4.6 CHAID Decision Tree

CHAID Decision Tree algorithm was also applied following the proposal of Giger et al. [5]. The proposed attributes were the same as those proposed in [1] as they were studying the same projects. The performance on RQM was higher than many rivals, providing 69.31% accuracy, however results on RTC were not competitive.

3.4.7 Random Forest

Following the proposal of Marks et al. [14] to use bug attributes to train Random Forest algorithm, we also trained a model based on Random Forest with 80 as number of trees using the attributes they suggested and were available in our datasets, namely *Creation year*, *Creation week*, *How found*, *Found in*, *Severity*, *Priority*, *Filed against*, *Planned for* and *Reporter*. Random Forest classifier performed better than its rivals on both RQM and RTC projects providing 70.97% and 65.39% prediction accuracy respectively.

Table 3: Performance of Metadata Based Models

Predictor	Acc(RQM)	Acc(RTC)
OneR	66.66	63.26
Logistic Regression	68.30	64.37
Naive Bayes	68.27	64.88
SVM	69.22	64.45
CHAID Decision Tree	69.31	63.61
C4.5 Decision Tree	68.72	65.30
Random Forest	70.97	65.39

3.5 Text Based Models

There has been two major methods for predicting task completion effort using text analysis in previous studies. In this section we present the procedure of applying them to our datasets as well as the results we achieved.

3.5.1 Nearest Neighbour Approach (kNN)

Following the approach of Weiss et al. [19], we applied Nearest Neighbour algorithm to find the most similar tasks and considered their average label as the prediction. For both *Summary* and *Description* attributes each separately, after text preprocessing, removing numbers, punctuation marks and white spaces we tokenized the documents, filtered out one-character tokens, then transformed all tokens to lowercase, removed all common English stop words and then applied Porter stemming algorithm [20] to remove redundant synonyms of tokens. Then we created a weighted word matrix for each task using TF-IDF algorithm. We ran a kNN classifier utilizing Cosine similarity function as suggested by Weiss . We used different k values of 3, 5, 7, 9 and 11 and we got the best results with k=3 for the *Summary* attribute and with k=9 for *Description*. The detailed results are presented in table 4 which show a better performance when targeting attribute *Summary* consistent with the findings achieved by Weiss.

Table 4: Performance of kNN Model

Attribute	k	Acc(RQM)	Acc(RTC)
Summary	3	68.50	62.09
Summary	5	68.50	61.94
Summary	7	68.16	61.18
Summary	9	68.05	61.05
Summary	11	68.03	57.98
Description	3	64.67	56.48
Description	5	64.58	57.09
Description	7	65.86	57.50
Description	9	65.88	58.49
Description	11	65.44	57.09

Despite the suggestion of Weiss, we did not filter tasks without *Description* or *Summary* nor we used α-kNN, the approach which sets a minimum similarity threshold and predicts *Unknown* if finds no similar tasks, as we aimed to propose a hybrid model that is not limited to availability of a certain attribute, capable of making predictions for all tasks.

3.5.2 Spherical k-means Clustering Approach

Following the proposal of Pfahl et al. [17], the same procedure of text preprocessing described in subsection 3.5.1 was applied separately on both *Summary* and *Description* attributes and a weighted word vector was created for each task. We then applied Latent Semantic Analysis in order to reduce the dimension of word vector and capture potential hidden relationships between words in the target text based attribute of each task. Then we performed Spherical k-means analysis adopting cosine similarity function using various k values and evaluated the cluster quality using Silhouette index in each iteration. This way we determined the best k value for each dataset. Pfhal does not mention how they used the clusters and which classification algorithm they utilized in order to make predictions, therefore we trained several algorithms providing the clusters as their input, although the results in average were not as good as the kNN approach discussed in previous section. The accuracy numbers when using Naive Bayes algorithm is presented in table 5. The accuracy numbers that we achieved using this method were not as high as was obtained in [17], however their reported results were based on only the newest issues in their dataset. Although we performed the analysis on different subsets of tasks following their suggestion and results slightly changed depending and the selected subset, we did not base our analysis on those subsets as this approach was not a good fit to the aim of this study to have a hybrid model adoptable to all tasks.

Table 5: Performance of Spherical k-means

Attribute	Accuracy(RQM)	Accuracy(RTC)
Summary	66.48	61.42
Description	63.43	59.24

3.6 The Hybrid Model

As suggested by Jorgensen [11], combining and averaging several independent models is likely to produce a more stable model with a higher accuracy. According to the experiments and results described in previous sections, we selected the best-performing individual model on each independent set of attributes in order to combine them and create the hybrid model: (1) Random Forest learner on metadata based attributes (2) kNN learner on attribute *Description* (3) kNN learner on attribute *Summary*.

First by joining attributes together and then using some common ensemble learning methods, we investigated the feasibility of adopting four types of model integration techniques to gain better predictive performance.

3.6.1 Single Learner on all Attributes

We applied the same procedure described in subsection 3.5.1 on *Summary* and *Description* of tasks separately to generate a weighted vector for each and then joined the word vectors together providing them as input of a kNN classifier using Cosine similarity function. For k=3 we achieved a classification accuracy of 68.91% for tasks of RQM and 61.96% for tasks of RTC project which does not indicate an improvement in average. Next, metadata based attributes were also joined with *Summary* and *Description* which considerably damaged the results. These two simple methods of joining attributes were meant to provide a baseline of potential possible improvements without using ensemble learning techniques.

3.6.2 Voting on Label

Voting or Majority Voting [13] is a common ensemble learning technique that considers the majority prediction of input models as the final prediction. Using the three selected independent models mentioned in subsection 3.6, we applied this technique on both datasets and achieved approximately 4% improvement in prediction accuracy compared to the average accuracy of input predictors.

3.6.3 Voting on Confidence

For this method as well as the next one, we needed a measure of confidence for class predictions. Our best individual models were based on Random Forest and kNN algorithms which are not probabilistic classifiers, therefore they do not natively produce confidence values.

For kNN we used the number of the neighbours with the predicted class divided by k as the measure of prediction

confidence:

$$confidence_{kNN} = \sum_{i=1}^{k} (class_i = prediction)/k$$

And for the confidence of Random Forest, we used the average confidence of each individual tree.

This way we achieved confidence values between 0 and 1 alongside binary classifications for each individual model. For this method, we mimicked a voting system by summing the confidence of individual predictors when predicting *High* and chose the label with higher sum of confidence value as the prediction. In other words:

$$prediction = \begin{cases} Low, & \text{if } \sum_{i=1}^{n} confidence_i \leq n/2 \\ High, & \text{otherwise} \end{cases}$$

where n is the number of input predictors, in our case 3.

This method provides a slight improvement over the standard method of voting on labels. The accuracy improvement on RQM dataset is not tangible and on RTC is 0.73

3.6.4 Stacking Individual Models

Stacking [21] is a common ensemble learning technique that provides predictions of other models as input data of a combined learner to make the final predictions. Providing predictions of individual models as input of the meta learner did not improve results compared to previous approaches. Then we chose to use confidence of individual models rather than their predicted labels to supply more information for the combined learner and as a result this method yielded better results. Logistic Regression, SVM, Naive Bayes and C4.5 Decision Tree were used as the combined learners and we achieved the best results using Logistic Regression. Stacking individual models on each of the projects, we achieved 5.06% more accuracy on RQM and 4.24% accuracy improvement on RTC compared to the average accuracy of their input models.

Table 6: Accuracy of Hybrid Models

Method	RQM	RTC
I1 - Random Forest on metadata	70.97	65.39
I2 - kNN on summary	68.50	62.09
I3 - kNN on description	65.86	58.49
H1a - kNN on summary + description	68.91	61.96
H1b - kNN on all attributes	56.14	55.44
H2 - Voting on label	72.35	65.69
H3 - Voting on confidence	72.37	<u>66.42</u>
H4 - Stacking on label	72.43	65.57
H5 - Stacking on confidence	<u>73.51</u>	66.23

Table 6 shows the performance of different approaches in creation of a hybrid model as well as the performance of their three input models for both projects. For RQM dataset, the Stacking on confidence approach provides the most significant boost on performance while simple Voting on confidence performs better on the RTC dataset. The results confirm that all the approaches based on standard ensemble learning techniques provide better results compared to the individual predictors.

4. DISCUSSION

According to the results presented in previous section, our proposed hybrid models not only increase the task completion effort estimation accuracy, but also the adoptability of automatic effort estimation to a larger number of tasks. As an example, by avoiding to include the concept of thresholds to the kNN classification technique using document similarity, this individual model will always have a prediction regardless of how confident it is but with different confidence levels which will be leveraged in a hybrid model working based on confidence values.

We analyzed the data of two different commercial projects developed and maintained by two separate teams. Although we achieved better accuracy numbers in one project compared to the other one, there is a promising consistency in the improvements made by the hybrid models within the two projects. This could be an indication of generalizability of the proposed hybrid models to other projects from different contexts being developed by different teams. Software companies could follow the same procedure as in this study to apply the previously proposed models on their projects and find the best performing ones based on the context of their projects and the attributes they measure and eventually create their own hybrid models using the approaches suggested in this study.

Some companies might find expert-based estimation more reliable and therefore have not adopted such machine learning techniques yet. Following the idea of this study, using the hybrid prediction approaches that we proposed, those companies can use the machine learning predictive models in conjunction with their own expert-based estimations to enhance estimation precision while expanding the predictions to more work items.

Predicting task completion effort and time is still an interesting and active area of research. Individual predictors have evolved over time and will continue to evolve in future. This would also be true about model integration techniques. This study would be a good starting point for other researchers to investigate more sophisticated model integration techniques while studying on improving individual predictors.

5. LIMITATIONS

Like any other empirical study, this study has some limitations that should be taken into account when interpreting the results. In this study, we analyzed only those tasks having a measured and entered value for the attribute *Time spent*. Those tasks might not necessarily be a good representative for all of the tasks in our datasets and therefore the results might not be generalizable to all development tasks. We applied our analysis on two commercial projects of IBM which supposedly have different standards and work practices from average software companies as well as open source projects. The datasets of this study did not have historical snapshots to make sure that the final value of included attributes for all tasks are equal to their value before they were assigned to a developer. However looking into historical versions of 10 tasks randomly chosen from each project, including those tasks with large *Time spent* values, the value of only one attribute, namely *Planned for*, was changed for two tasks in total. That shows that the values of these attributes rarely change and could not have affected our results in a tangible way.

6. CONCLUSION AND FUTURE WORK

In this study, we explored various previously proposed models working on different attribute sets, applied them on two different commercial projects, found the best performing ones and proposed several approaches to integrate those individual models in order to increase task completion effort estimation accuracy. We showed that our integration approaches consistently increase early effort prediction performance on both projects providing 3-8% more accurate predictions compared to the best individual predictors.

Future work includes extending this study to other projects to see how the results generalize. The target projects could be from other large-scale companies, average software companies or from open source communities. We also plan to extend this research to other types of work items to see whether the prediction improvement of proposed hybrid models pertain. As time and effort estimation could be also beneficial in the middle of a task's lifecycle, we also would like to investigate the inclusion of post-assignment data into our models and assess our approaches in the consequent settings. The focus of this study was on task completion effort estimation but further research is required to examine if the same hybrid techniques are also applicable to completion time estimation models. Assessing the performance of these hybrid models on multiclass predictions instead of binary classification is another future work. Further prediction integration techniques also need to be explored, including integration of automatic predictive models with expert-based estimations.

Another avenue of future work includes extending this research to predict completion effort for work items at higher abstraction levels like stories or plans that encompass multiple bugs and development tasks. Future studies should investigate the adoptability of previously proposed models as well as the hybrid model proposed in this study on high level work items.

We also plan to validate the results of our analysis with the IBM developers and investigate whether the type of predictions from our models resemble the reality of task completion effort in their projects.

7. ACKNOWLEDGMENTS

We are grateful to Kevin Ryan for his early feedback on this analysis. Thanks also to Adam Neal and Alan Yeung from IBM for their valuable inputs to this study.

8. REFERENCES

[1] W. Abdelmoez, M. Kholief, and F. M. Elsalmy. Bug fix-time prediction model using naïve bayes classifier. In *Comp. Theory and Applications (ICCTA), 2012 22nd Int'l Conf. on*, pages 167–172. IEEE, 2012.

[2] J. Anvik, L. Hiew, and G. C. Murphy. Who should fix this bug? In *Proc. of the 28th international conference on Software engineering*, pages 361–370. ACM, 2006.

[3] P. Bhattacharya and I. Neamtiu. Bug-fix time prediction models: can we do better? In *Proc. of the 8th Working Conference on Mining Software Repositories*, pages 207–210. ACM, 2011.

[4] K. O. Elish and M. O. Elish. Predicting defect-prone software modules using support vector machines. *Journal of Systems and Software*, 81(5):649–660, 2008.

[5] E. Giger, M. Pinzger, and H. Gall. Predicting the fix time of bugs. In *Proc. of the 2nd International Workshop on Recommendation Systems for Software Engineering*, pages 52–56. ACM, 2010.

[6] D. Gray, D. Bowes, N. Davey, Y. Sun, and B. Christianson. Using the support vector machine as a classification method for software defect prediction with static code metrics. In *Int'l Conf. on Eng. Apps. of Neural Networks*, pages 223–234. Springer, 2009.

[7] P. J. Guo, T. Zimmermann, N. Nagappan, and B. Murphy. Characterizing and predicting which bugs get fixed: an empirical study of microsoft windows. In *2010 ACM/IEEE 32nd International Conf. on Software Eng.*, volume 1, pages 495–504. IEEE, 2010.

[8] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

[9] R. Hewett and P. Kijsanayothin. On modeling software defect repair time. *Empirical Software Engineering*, 14(2):165–186, 2009.

[10] M. Hofmann and R. Klinkenberg. *RapidMiner: Data mining use cases and business analytics applications*. CRC Press, 2013.

[11] M. Jorgensen. What we do and don't know about software development effort estimation. *IEEE software*, 31(2), 2014.

[12] Y. Kamei, T. Fukushima, S. McIntosh, K. Yamashita, N. Ubayashi, and A. E. Hassan. Studying just-in-time defect prediction using cross-project models. *Empirical Software Engineering*, pages 1–35, 2015.

[13] J. Kittler, M. Hatef, R. P. Duin, and J. Matas. On combining classifiers. *IEEE transactions on pattern analysis and machine intelligence*, 20(3):226–239, 1998.

[14] L. Marks, Y. Zou, and A. E. Hassan. Studying the fix-time for bugs in large open source projects. In *Proc. of the 7th International Conf. on Predictive Models in Software Eng.*, page 11. ACM, 2011.

[15] A. T. Mısırlı, A. B. Bener, and B. Turhan. An industrial case study of classifier ensembles for locating software defects. *Software Quality Journal*, 19(3):515–536, 2011.

[16] L. D. Panjer. Predicting eclipse bug lifetimes. In *Proc. of the Fourth International Workshop on mining software repositories*, page 29. IEEE Computer Society, 2007.

[17] D. Pfahl, S. Karus, and M. Stavnycha. Improving expert prediction of issue resolution time. In *Proc. of the 20th Int'l Conf. on Evaluation and Assessment in Software Engineering*, page 42. ACM, 2016.

[18] S. W. Thomas, M. Nagappan, D. Blostein, and A. E. Hassan. The impact of classifier configuration and classifier combination on bug localization. *IEEE Transactions on Soft. Eng.*, 39(10):1427–1443, 2013.

[19] C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller. How long will it take to fix this bug? In *Proc. of the Fourth International Workshop on Mining Software Repositories*, page 1. IEEE Computer Society, 2007.

[20] P. Willett. The porter stemming algorithm: then and now. *Program*, 40(3):219–223, 2006.

[21] D. H. Wolpert. Stacked generalization. *Neural networks*, 5(2):241–259, 1992.