2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)

Predicting Likelihood of Requirement Implementation within the Planned Iteration: An Empirical Study at IBM

Ali Dehghan*, Adam Neal†, Kelly Blincoe‡, Johan Linaker§ and Daniela Damian*
*Computer Science Department, University of Victoria, BC, Canada, {dehghan, danielad}@uvic.ca
†Persistent Systems, 515 Legget Drive, Kanata, ON, Canada, nealadam@ca.ibm.com
‡Department of Electrical and Computer Engineering, University of Auckland, New Zealand, kblincoe@acm.org
§ Computer Science Department, Lund University, Sweden, johan.linaker@cs.lth.se

***Abstract*—There has been a significant interest in the estimation of time and effort in fixing defects among both software practitioners and researchers over the past two decades. However, most of the focus has been on prediction of time and effort in resolving bugs, without much regard to predicting time needed to complete high-level requirements, a critical step in release planning. In this paper, we describe a mixed-method empirical study on three large IBM projects in which we developed and evaluated a process of training a predictive model constituting a set of 29 features in nine categories in order to predict if a requirement will be completed within its planned iteration. We conducted feature engineering through iterative interviews with IBM practitioners as well as analysis of large development repositories of these three projects. Using machine learning techniques, we were able to make predictions on completion time of requirements at four different stages of their lifetime. Using our industrial partner's interest in high precision over recall, we then adopted a cost sensitive learning method and maximized precision of predictions (ranging from 0.8 to 0.97) while maintaining an acceptable recall. We also ranked the features based on their relative importance to the optimized predictive model. We show that although satisfying predictions can be made at early stages, performance of predictions improves over time by taking advantage of requirements' progress data. Furthermore, feature importance ranking results show that although importance of features are highly dependent on project and prediction stage, there are certain features (e.g. requirement creator, time remained to the end of iteration, time since last requirement summary change and number of times requirement has been replanned for a new iteration) that emerge as important across most projects and stages, implying future worthwhile research directions for both researchers and practitioners.**

***Keywords*-mining software repositories; machine learning; completion time prediction; release planning;**

I. Introduction

With the wide adoption of agile practices in the increasingly competitive software industry, many product managers strive for fast, short and incremental releases [1]. Each release has a set of requirements to implement new functionality, and a set of bugs that need to be fixed. Frequent releasing allows new functionality to be delivered quickly, ahead of the competition. Thus, late or incomplete releases can impact product success [1]. Enterprises adopting agile practices for their release planning normally plan requirements for iterations. Therefore, it would be useful for product managers to know early when certain requirements will not make it into an iteration.

While there is an extensive body of literature on predicting various aspects of software releases (such as release readiness [2], [1]) and of software tasks (such as completion time [3], [4], [5], [6], [7], [8] and completion effort [9], [10], [11], [12]), we are not aware of studies that investigate the likelihood of a requirement being implemented in the planned iteration.

To fill this research gap, this paper describes an empirical, design study of requirements from three large projects at IBM to answer the following research questions:

> *RQ1:* Can we predict whether or not a requirement will be completed within the planned iteration?
> *RQ2:* Can we optimize the predictive model to maximize precision of predictions, while maintaining an acceptable recall?
> *RQ3:* What are the features[1] that can be used in this prediction and how important are they relatively?

To answer these questions, we used a combination of qualitative and quantitative methods. Throughout the study, we worked closely with an IBM Analytics Architect (the second author) and other IBM practitioners. This close connection allowed us to obtain a concrete understanding of the IBM ecosystem and their workflow [13].

Our study has three main contributions:

1) Through a process of feature engineering we propose a predictive model that is capable of making predictions at different stages of a requirement lifetime and results in an F1-score between 0.56 and 0.78.
2) We optimize the predictive model to maximize precision of predictions (at the expense of recall) to address IBM business interest as they find low precision predictions of little utility in their practice. This model obtains precision values between 0.80 and 0.97.
3) We rank the engineered features according to their relative importance to our optimized model. This helps other researchers know what features to consider in their

[1]Note that the term feature refers to a model feature, not to be confused with a software feature.

978-1-5386-1544-7/17 $31.00 © 2017 IEEE
DOI 10.1109/MSR.2017.53

future studies. It also helps software organizations know what kind of data they should record for future analysis.

II. Related Work

Although no other studies propose methods to predict whether a high-level requirement will be completed in a planned iteration, there are related areas of research. Below we present related work covering prediction on various aspects of software tasks and releases.

A. Software Task Predictions

Prior studies on low-level software tasks (i.e. bug fixes and issue resolutions) have foused on predicting resolution time [3], [4], [5], [6], [7], [8], effort involved [9], [10], [11], [12], and probability of completion [14]. However, none of them have investigated completion time of high-level requirements, yet we draw on many of the techniques and features used.

These studies employ various machine learning techniques to make their predictions. Unsupervised learning techniques [15], [10], [11], kNN [9], linear [3], probabilistic [3], [5] and decision tree [3], [4] classifiers as well as random forest [6], [8] and other ensemble learning techniques [12] are among the popular ones. In our study we adopt the random forest ensemble learning technique for model training.

Various features are used to make predictions on software tasks. A number of studies utilize task meta attributes, such as task creator, owner, priority, severity, etc. to make predictions [3], [4], [5], [14]. Some also apply text analysis techniques on text attributes [9], [16], [10], [11], [12] and some adopt social network analysis techniques [7]. Yet, not all of the studies have come to the same conclusions in regards to which features are best to include. Guo et al. found that bug reporter's reputation has a significant impact on bug resolution time [14]. However, Bhattacharya and Neamtiu [17] found no such correlation and stated that importance of features are highly project dependent. Inspired by these past studies, we identify 29 features for our models (described in detail in Section IV).

Marks et al. [6] and Kikas et al. [8] both classified their engineered features into several groups based on the source of features and measured relative importance of features to their models. Likewise, we rank the importance of the engineered features to our models, and also classify our features into nine categories based on their potential impact on completion time.

Making early predictions can be useful, but predictions likely become more accurate as more data become available. Some of the studies made predictions at different stages. For example, Giger et al. made predictions at six different stages of a bug lifetime [4]. Kikas et al. made predictions at different stages of an issue lifetime [8]. Similarly, we make predictions at various stages of a requirement lifecycle.

B. Software Release Predictions

In regards to the domain of release planning and release readiness, prior studies have conducted predictive analysis using various machine learning techniques. In an empirical study on three open source projects, Alam et al. [2] formulated software release readiness in each week as a binary classification problem and used eight features such as release duration and number of open requirements and defects while comparing different classifiers. In another study [1] on the same data they proposed an improved predictive model using incremental and sliding window techniques and then empirically evaluated the applicability of their model for varying project characteristics.

III. Study Design

This section describes our research setting, research method, and dataset.

A. Research Setting

We studied three large projects of an IBM product that aim to provide IBM customers with an enterprise platform (referred to as the IBM enterprise platform henceforth). This platform provides a community in which developers of the IBM ecosystem and customers of the product can collaborate and communicate. Members of this community can create, modify, resolve or comment on a work item. IBM adopts an agile methodology for development of these projects. Their adopted workflow is illustrated in Figure 1.

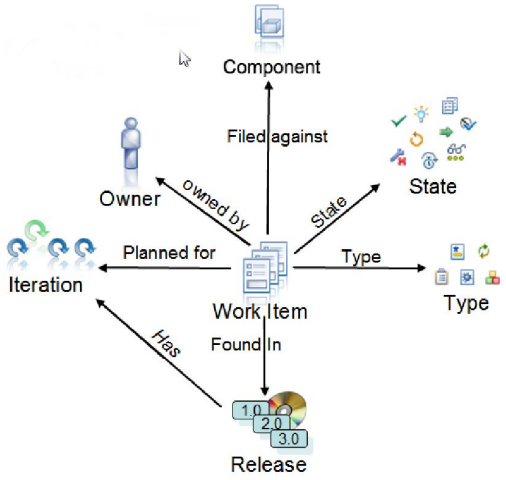

Fig. 1. IBM enterprise platform workflow

To understand this workflow, we define some terminology:

Work items represent work that needs to be done. Work items vary in size from small chunks of work to very large. Work items are also hierarchical, which means a work item could have one or multiple children or grandchildren. A work item has a type attribute which defines the workflow of the work item. Work item types are: Plan Item, Story, Enhancement, Task and Defect. The ideal hierarchy of work items in the IBM enterprise platform is shown in Figure 2.

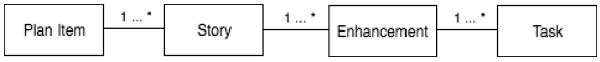

Fig. 2. Hierarchy of work items in IBM enterprise platform

The hierarchy in practice, however, is sometimes inconsistent with this ideal structure. For instance, work items might have children with the same work item type (e.g. a Story could have a child of type Story), and, in some rare cases, this hierarchical order is violated. Work items of type Defect could be a child of any of these other work item types.

PLAN ITEMS are top level work items that represent software requirements and functionalities that should be included in next release(s). During the lifetime of a Plan Item, new children and grandchildren can always be added. A Plan Item is not completed until all of its children are completed.

STORIES are also high level work items that are breakdowns of Plan Items. Stories and Plan items have many common characteristics in this platform.

WORK ITEM HISTORY: Every time a change happens to a direct attribute of a work item (for instance if the status or owner of a work item changes) or a new subscriber or comment is added to a work item, this change is recorded as a work item history. However, the addition or removal of children is not recorded as a work item history.

ITERATION: A time-box during which development takes place. Iterations are hierarchical where top-level iterations represent releases and child iterations represent milestones within those releases. A milestone iteration is typically of 1 to 4 weeks length. Release iterations vary in duration from 1 month up to 1 year. Work items could be planned for a milestone or a release iteration, and if they don't get finished through an iteration, they could be planned for another one.

According to the interest of IBM managers in having predictions for both Plan Items and Stories and considering the many inherent similarities between the two in terms of measured attributes as well as the enterprise internal processes, we decided to analyze both of them in this study and we will refer to them as requirements in the rest of this paper.

B. Research Method

This study was led and designed by our research team in collaboration with an IBM Analytics Architect. The idea and the initial design of the study were initiated in a face-to-face meeting with a program director and the analytics architect at IBM. The collaboration continued by another face-to-face meeting and 16 semi-structured 30-60 min interviews with the analytics architect and three other IBM practitioners.

Through interviews with the IBM practitioners, the first step in our research method was to obtain a rich understanding of the IBM enterprise platform and to define the appropriate projects for our study. We eventually chose three of their projects based on duration and activity and number of plan items and stories in their development and project management repositories. After the initial interviews and selecting three projects, the process of feature engineering started. This was an iterative long process and involved significant time in brainstorming and discussing with the IBM engineers. Details on the feature engineering process are provided in Section III-E and Section IV. Another important step in this process was the review of related work in task completion time and effort prediction, defect prediction and bug triaging. In our study, an important step was the achievement of an understanding of the differences between low-level software tasks and high-level abstractions of software tasks, in particular Stories and Plan Items, so that adaptations to the proposed features in literature could have been made.

For our learning algorithm, we used Random Forest (RF). RF has shown high performance on many types of datasets compared to many well-known algorithms such as SVM and Naive Bayes [18]. It is robust to noise in data, is applicable to datasets with a mixture of continuous, semi-continuous and categorical features, and is capable of handling missing values as well as correlated features [19].

In terms of the prediction outcome, we formulated our analysis as a binary classification problem. Comparing the completion date of a requirement to the end date of the planned iteration, we derived the prediction outcome:

$$iteration\_met = \begin{cases} YES, & \text{if } completion\_date \leq end\_date \\ NO, & \text{otherwise} \end{cases} \tag{1}$$

As we assumed that requirements are supposed to be completed within their planned iteration, a trend we confirmed from our examination of the historical data of the projects, the NO class normally made up the minority of requirements, and thus was considered as the positive class that our learners were meant to predict.

Although there was an imbalance in class distribution of most of our datasets, the skewness was not critical to demand a cost sensitive learning process for RQ1 (we refer to the models trained for RQ1 as cost insensitive models). However, to address RQ2 and RQ3, since we favored high precision over high recall, we used cost sensitive learning with a high penalty for false positives to make our models more cautious when making predictions on the positive class. There exist many techniques for cost sensitive learning, most of which mentioned in a literature review by He and Garcia [20], which could be classified into two general categories: 1) cost sensitive learning which perform resampling or sample-weighting on the minority class to make the data more balanced; and 2) approaches that minimize the expected cost of classification utilizing the confidence of the base classifier in predictions. We use a sample-reweighting technique through the CostSensitiveClassifier class of WEKA [21] library which is an implementation of the approach introduced by Ting [22]. We determined the false positive penalty by considering the original class balance to make it balanced. In addition, we increased the false positive penalty in order to comply with the goal of maximizing precision of predictions. As a result, a false positive penalty between 3 to 5 was applied depending on the skewness of dataset.

We used a variety of tools and libraries for our analysis, however RapidMiner [23] and WEKA [21] were the major ones. The final implementation was done in Java code using WEKA libraries.

C. When to Predict?

We decided to make predictions at four different stages for each project and work item type: The first day of creation of a requirement and the end of the first, second and third quarter of the planned iteration. This allows us to make predictions at different stages of a requirement lifetime, something that is beneficial to enterprises. It also enables us to compare the significance of features at different points in time. The selected stages are meaningful and derived from the actual needs of the enterprise. Another benefit to this approach is that training and testing data points will be automatically selected based on the same percentage of the progress of requirements. Therefore, these data points are more meaningful and related to each other, assuming that requirements at the same time slot of their corresponding iteration are approximately at the same point of their progress. For simplicity, we will onwards refer to these four prediction stages with 0th, 1st, 2nd and 3rd short form notations.

Considering that work items in this platform might have many histories within the same quarter of an iteration, we defined the history selection criteria as following:

- 0th: The first history of a work item
- 1st, 2nd and 3rd: The last history of a work item within the corresponding quarter of its planned iteration.

D. Datasets

We studied three projects of the IBM enterprise platform, code-named as A, B and C due to confidentiality reasons. Table I shows some statistics of the development repository of these three projects in our observation period.

TABLE I
SUMMARY OF PROJECTS

Att / Project	A	B	C
start date	Jun 2006	Jan 2009	Jun 2006
end date	Oct 10, 16	Oct 11, 16	Oct 24, 16
# work items	75k	71k	177k
# histories	816k	835k	1.73m
# plan items*	839	749	447
# PI histories	20k	20k	19k
# stories*	1,286	3,640	4,471
# S histories	16k	61k	50k
# comments	374k	312k	312k
# developers	594	481	796

*: As type of a work item is subject to change, number of Plan Items and Stories is based on any work item that has at least one history of that type.

As a result of having three different projects (A, B and C), two different requirement types (Plan Item, Story) and four different prediction stages (0th, 1st, 2nd, 3rd), we have 24 datasets in total. For simplicity from now on, we will refer to a specific dataset by the Project-Type-Stage notation. For instance B-plan-2nd will refer to the dataset of Plan Items of project B at the end of the 2nd quarter of their planned iteration.

As could be observed from Table III, the sample size of each of these 24 datasets is smaller than the number of requirements mentioned in Table I, because of two major filters: First, we only studied those requirements that are already completed. Second, We only studied those selected requirement histories that were planned for an iteration with an end date.

E. Feature Engineering

Feature engineering is the process of using domain knowledge of the data and the processes behind the data to create predictors for a machine learning algorithm in order to build a predictive model. Feature engineering could be seen as the most important factor in the success or failure of a machine learning task. It is where intuition, creativity and black art are as important as technical knowledge. It is the part that usually takes the most effort as it is normally done manually. Besides manual feature engineering, there are automatic techniques such as automatically generating large numbers of candidate features and selecting the best by their information gain, but these techniques are usually very time-consuming and could cause over-fitting [24].

Feature engineering was the most effort-consuming machine learning task in this study and involved iterative brainstorming, data visualization, digging into data, interviews with IBM software practitioners as well as literature review. As a result, a set of 29 features were engineered. Each of these model features come from one or a multiple of these sources:

1) Prior work, chiefly in bug resolution time and effort prediction.
2) Suggestions made by IBM developers.
3) Domain knowledge achieved as a result of interviews with enterprise practitioners and studying the data.

The motivation behind choosing these features and their descriptions are stated in Section IV. Despite differences in practices across teams and inherent differences in the characteristics of the two work item types, the commonalities between them were high enough to enable us using almost the same feature set for each dataset, except for a few minor exceptions which will be described in Section IV-I.

F. Model Performance Metrics

When it comes to choosing a good performance metric, one of the important factors to consider is the balance of data. In Table II, we see the distribution of class values in all datasets. NO% denotes the ratio of NO class to the whole dataset size.

TABLE II
SKEWNESS IN EACH DATASET

dataset	NO%			
	0th	1st	2nd	3rd
A - plan	0.53	0.44	0.49	0.49
B - plan	0.30	0.54	0.52	0.32
C - plan	0.35	0.38	0.39	0.39
A - story	0.50	0.38	0.40	0.40
B - story	0.44	0.20	0.19	0.18
C - story	0.50	0.37	0.36	0.36

In average, we see a minor skewness in datasets of Plan Items and a relatively major skewness in datasets of Stories. To address that, we based our performance evaluations on

precision and recall of the positive class. As a result, to have a single performance metric, we used F1-score which is the harmonic mean of precision and recall. F1-score was used to evaluate the models trained to address RQ1.

IBM managers stated they are interested in having the highest possible precision, even though it might result in lower recall values. They stated that a precision of below 0.8 would be of little utility in practice and a precision of 0.9 or above would be ideal for them. At the same time, they pointed out that having a high precision, a recall value around 0.2 or 0.3 would be good enough. That is why we came up with this weighted arithmetic mean for performance measure:

$$WA = (3 * precision(NO) + recall(NO))/4 \qquad (2)$$

Although we will still report performance metrics such as precision and recall, we will use WA as defined above to compare performance of our models across different datasets as well as to measure and compare feature importance. The WA measure will be used for performance evaluation of models trained to answer RQ2, as well as to measure feature importance to address RQ3, so that we can rank feature importance based on the business interest of our industrial partner, which was the main motivation behind this study.

G. Feature Importance Measure

There exists techniques such as Cohen's F2 test [25] or Chi-square measure [26] to measure the effect size on or relevance of features to the class feature out of the box, but we want a measure that can tell us how important each feature is to a trained model and thus, how vital a certain feature could be in order to achieve the actual predictive goal.

Random forests can be used to rank the importance of features in a regression or classification problem in a natural way by measuring the out-of-bag error for each data point [27]. However, due to the goal of maximizing the precision of positive class and ranking the importance of features for that prediction, we decided to use another measure for feature importance that can be applied to any learning algorithm. Our approach was inspired by the study of Marks et al. [6].

To rank feature importance, for each dataset, a model was trained for all 29 features and WA was calculated. Then in 29 iterations, each time a single feature was excluded and a model was trained and tested using cross validation and the extent of decrease (or increase) in WA measure was calculated. Finally, features in each dataset were sorted and ranked based and the extent of change in WA measure.

H. Model Validation

K-fold cross validation is a common technique for evaluating the performance of a process of creating a predictive model. Although choosing the appropriate value for k might be problem-dependent, values such as 10 or 5 as number of folds are known as convenient good choices according to the variance-bias tradeoff [28], [29]. However Kocaguneli et al. [30] showed that in effort estimation problems, Leave-One-Out cross validation (LOOCV) - an extreme case of CV in which k equals the number of instances - performs better than 10-fold CV. Moreover, due to the relatively small sample size of our datasets, the choice of LOOCV was preferable. However, some of our datasets had relatively larger sample size (B-story-Sth and C-story-Sth), thus, we chose to do 200-fold cross validation for them to make it computationally possible. The value of 200 was selected to maintain consistency of analysis so that number of folds would be closer to the ones validated using LOOCV.

I. Other Important Decisions

In this section, we describe some of the other important decisions that we made in the study design which we did not mention in previous sections.

No Manual Feature Selection: We ran Pearson correlation analysis between each pair of features, the highest correlation values were between 0.6 and 0.8 which can still be handled by the RF algorithm. The RF algorithm is designed to be robust against correlated or non-informative features. Thus, as one of the objectives of this research was ranking features based on their importance to the trained models, we did not exclude any of the features.

No Automatic Feature Selection: Automatic feature selection is generally time consuming and could cause overfitting [24], especially when dealing with small sample sizes and a large number of attributes in raw data. Additionally, in interacting with our industrial partner, we had access to expert knowledge. Therefore, there was no need for automatic procedures in feature engineering or selection.

No Parameter Tuning: Although there are studies in areas such as effort estimation [31] and defect prediction [32] that show parameter tuning could have positive effects on performance of predictions, we did not attempt to optimize learner hyper parameters because: 1) In RF there are two main parameters; first, the number of trees, which should be large enough to fit the computational power (we chose 100) and second, the number of randomly selected features on each split which the original number used by Breiman [33] usually performs well. 2) To avoid biased results, one should do nested CV for parameter tuning or feature selection and due to small sample size of our datasets, we did not have enough data to do so.

IV. Model Features

In this section, we introduce all the engineered features used in our analysis and motivate them. There are 29 features, which we categorized into 9 logical categories. Some features logically fit into more than one category. In those cases, we put them in the most relevant category only. In following sections, we describe each category and the motivation for including each feature. We also cite prior work that has adopted similar features in other prediction models. Note that a citation right after the feature name indicates that feature or a similar one was used in the referenced prior work, but it does not necessarily mean that they had the same motivation as in this work since we had different goals and settings in this study.

A. General Features

There are three features that are available early on in a requirement lifetime. We call these the general features.

- CREATOR_IDENTIFIER [3], [4], [5], [6], [8], [12], [14], [34]: the stakeholder who created the requirement. Available at requirement creation.
- CREATION_MONTH [4], [5], [6], [12]: the month the requirement was created. Available at requirement creation. Depending on the project and timeline of the corresponding team, it could capture factors such as workload of a team that could have impact on completion time of a requirement.
- OWNER_IDENTIFIER [3], [4], [5], [14], [6]: the developer assigned to a requirement. Available when the requirement is assigned to a developer.

B. Complexity Indicating Features

The more complex a problem is, the more time it is likely to take. Thus, we include features which can indicate the complexity of a requirement.

- SUBSCRIBER_COUNT [3], [4], [5], [6], [8]: the number of stakeholders subscribed to a certain work item to receive updates. Whenever someone comments on a work item, they are automatically subscribed to the work item. Stakeholders could also manually subscribe to or unsubscribe from work items. A high interest in receiving updates could indicate high complexity.
- FILED_AGAINST [3], [4], [5], [6], [12]: the software component against which the requirement is filed. Some components are inherently more complicated than others. For instance a database related component is likely to entail more complexity than a UI related component.
- ITERATION_CHANGE_COUNT: the number of times a requirement was replanned for a new iteration. If the requirement gets carried over to the next iteration multiple times, it could be an indication of high complexity.

C. Progress Implying Features

The current progress of a requirement will influence whether it will be completed in time or not. Thus, we have three features that enable the model to gauge current progress:

- ITERATION_DAYS_REMAINED [4], [5], [2], [1]: the number of days remaining to the end of the planned iteration.
- DAYS_SINCE_CREATION [3], [4], [5]: the number of days since the work item was created.
- STATUS [4], [5], [14]: the current status of a work item, such as new, in exploration phase, in progress, in testing phase, etc.

D. Priority Implying Features

Having a higher priority usually helps a requirement to receive more attention and activity and thus to get completed earlier.

- PRIORITY: an explicit measure of importance from the developers perspective. Not available for many work items.
- SEVERITY: an explicit measure of importance from the customers perspective. Not available for many work items.
- DAYS_WITHOUT_OWNER: number of days the work item was not assigned to a developer since creation; many days could indicate low priority.
- DAYS_SINCE_LAST_COMMENT: the number of days since the last comment, many days could indicate low priority.
- OWNER_CHANGE_COUNT [14], [8], [35]: the number of times a work item has been reassigned. Reassignments could be an indication of high priority as people reassign a task to find the optimal developer who can address the problem quickly [14]. However, too many reassignments could mean that no one is taking responsibility for handling the task [35], indicating low priority.
- DAYS_SINCE_LAST_OWNER: the number of days since the current work item owner was assigned. This is meant to capture a similar effect to the previous feature by considering how recent the last reassignment is.

E. Problem Change Indicating Features

A change in the problem definition of a requirement can impact completion time. Frequent changes could indicate additional changes in the future. This is captured by:

- SUMMARY_CHANGE_COUNT: the number of times the work item summary has changed.
- DESCRIPTION_CHANGE_COUNT: the number of times the work item description has changed.

Also, if the problem definition has changed recently, effort may still be underway to deal with the change. Thus, we consider:

- DAYS_SINCE_LAST_SUMMARY: the number of days since the last summary change.
- DAYS_SINCE_LAST_DESCRIPTION: the number of days since the last description change.

Previous studies attempted to capture changes in requirements by proposing basic features such as the number of total changes to bug attributes [4], [5] (equivalent to number of histories in our datasets), but to the best of our knowledge these particular four features are novel to this study.

F. Process Change Indicating Features

In addition to a change in problem definition, a change in the software process could also impact completion time. This is especially true if the process change is recent. It's not uncommon in IBM enterprise platform for a Story to be transformed to a Plan Item and vice a versa. It also sometimes happens that a work item is created with type of Task and then the developers decide that it should be transformed to a requirement. Such process changes could potentially delay implementation of a requirement.

- DAYS_SINCE_LAST_TYPE_S_P: number of days since last time the requirement type was changed from Plan Item to Story or vice a versa.

- DAYS_SINCE_LAST_TYPE_CHILD: number of days since last time the work item type was changed from a low-level work item type to a requiremnt.

G. Stakeholder Characteristics

The types of stakeholders of a work item can impact its completion time. For this, we consider:

- DAYS_SINCE_LAST_DE_COMMENT: number of days since a distinguished engineer (DE) commented on the work item. In IBM, DE is a title reserved for very respected developers. If a DE participates in the discussion of a work item, it will likely receive prompt attention.
- COMPONENT_RESOLVER: the total number of work items that the work item owner has resolved in the same component up until the modified date of the corresponding history of each requirement. This indicates the expertise of the owner in the domain of the problem.
- CREATOR_TEAM_RELATIONSHIP [14], [6]: the relationship of the work item creator to the assigned team. Prior work suggests that bugs reported by people on the same team are likely to get fixed faster [14]. For this feature, the creator could be an IBM developer from the same team, an IBM developer from another team, or a customer from outside the company.

H. Stakeholder Communication

Communication between stakeholders of a work item can impact completion time. A large volume of communication may indicate high complexity and, thus, result in longer completion time [14]. On the other hand, increased communication can indicate a good information flow and a high level of awareness and engagement and, thus, results in shorter completion time [14]. Communication in this enterprise platform usually happens via commenting on work items, so we consider:

- COMMENT_COUNT [3], [4], [5], [6], [8]: number of comments on the work item.
- COMMENTER_COUNT [8]: the number of unique stakeholders involved in the discussion on a work item.

I. Child Features

As described in Section III-A, work items are hierarchical in the IBM enterprise platform. Requirements normally have children and grandchildren. The number of children a requirement has can impact its completion time for a variety of reasons, e.g.:

1) child work items are a breakdown of the requirement,
2) the number of children impacts the effort required for planning and breaking down of the requirement as well as the effort required to integrate and test the children,
3) each child has its own idle time (such as time spent on triaging and assignment).

The type of a child work item can play an important role in this effect since plan items and stories often involve a more significant amount of work. We also want to consider the progress of the children, since a child work item that has been completed or is near completion should not greatly impact the completion time of the parent. Therefore, we consider:

- SAME_TYPE_CHILD_COUNT_NEW: the number of first descendant child work items of the same type as the requirement, with progress status New
- LARGE_SIZE_CHILD_COUNT_NEW : the number of first descendant child work items of type Story (or Enhancement if requirement type is Story), with progress status New
- MEDIUM_SIZE_CHILD_COUNT_NEW: the number of first descendant child work items of type Enhancement (or Task if requirement type is Story), with progress status New

V. Results

In this section, we present the answers to our RQs.

A. Cost Insensitive Model Performance (RQ1)

RQ1: Can we predict whether or not a requirement will be completed within the planned iteration?

Table III shows the performance estimation of our cost insensitive learning process in each of the 24 datasets. It is evident that there is some skewness in our datasets as evidenced by the NO% column which shows the ratio of positive class to the dataset size. Despite this, our predictions result in precisions higher than 0.60 for all projects, work item types and stages. In some cases, precision is as high as 0.85. F1-scores are also high with all but one dataset obtaining an F1-score of higher than 0.56 and some datasets achieving scores as high as 0.78. The one dataset that received a lower F1-score, C-plan-0th, is a very small dataset.

Comparing the results between the two work item types, we observe fairly similar prediction performances for both Plan Items and Stories. As expected, prediction performance tends to increase as the prediction stage increases, due to the availability of additional data at these later stages.

One key factor to take into account when comparing results, however, is the balance of dataset denoted by the column NO%. We expect to have better results when the dataset is more balanced since it allows for enough positives in the training set. We see a trend of having a more balanced dataset in the 0th prediction stage for most projects and work item types.

Answer to RQ1: We developed models that were able to accurately predict whether a requirement completed within the planned iteration. We obtained precision scores of up to 0.85 and F1-scores up to 0.78 (see Table III).

B. Cost Sensitive Model Performance (RQ2)

RQ2: Can we optimize the predictive model to maximize precision of predictions, while maintaining an acceptable recall?

Table IV shows the precision, recall and weighted average (WA) of our cost sensitive learning process for each dataset.

TABLE III
COST INSENSITIVE MODEL RESULTS

stage	count	NO%	precision	recall	F1	WA	count	NO%	precision	recall	F1	WA	count	NO%	precision	recall	F1	WA
	Project A - Plan Items						Project B - Plan Items						Project C - Plan Items					
0th	316	.53	.76	.69	.72	.74	231	.30	.69	.47	.56	.63	77	.35	.67	.24	.35	.56
1st	393	.44	.82	.70	.76	.79	224	.54	.63	.67	.65	.64	267	.38	.60	.58	.59	.60
2nd	462	.49	.84	.73	.78	.81	287	.52	.68	.70	.69	.68	280	.39	.67	.56	.61	.64
3rd	468	.49	.84	.73	.78	.81	473	.32	.70	.46	.56	.64	287	.39	.63	.51	.57	.60
	Project A - Stories						Project B - Stories						Project C - Stories					
0th	521	.50	.73	.71	.72	.72	2020	.44	.69	.65	.67	.68	1644	.50	.71	.71	.71	.71
1st	769	.38	.74	.58	.65	.70	2472	.20	.81	.51	.63	.74	1999	.37	.66	.52	.58	.62
2nd	842	.40	.73	.59	.65	.69	2759	.19	.85	.54	.66	.77	2310	.36	.68	.49	.57	.63
3rd	873	.40	.74	.62	.67	.71	2925	.18	.84	.55	.66	.76	2422	.36	.69	.50	.58	.64

TABLE IV
COST SENSITIVE MODEL RESULTS

stage	count	NO%	precision	recall	WA	count	NO%	precision	recall	WA	count	NO%	precision	recall	WA
	Project A - Plan Items					Project B - Plan Items					Project C - Plan Items				
0th	316	.53	.92	.56	.83	231	.30	.94	.24	.77	77	.35	1.0	.04	.76
1st	393	.44	.90	.50	.80	224	.54	.85	.28	.70	267	.38	.76	.28	.64
2nd	462	.49	.93	.56	.83	287	.52	.86	.28	.71	280	.39	.80	.15	.64
3rd	468	.49	.91	.56	.82	473	.32	.94	.11	.73	287	.39	.83	.18	.67
	Project A - Stories					Project B - Stories					Project C - Stories				
0th	521	.50	.88	.34	.74	2020	.44	.85	.30	.71	1644	.50	.85	.34	.72
1st	769	.38	.88	.29	.73	2472	.20	.97	.23	.79	1999	.37	.84	.16	.67
2nd	842	.40	.87	.31	.73	2759	.19	.94	.30	.78	2310	.36	.84	.16	.67
3rd	873	.40	.86	.29	.71	2925	.18	.95	.30	.79	2422	.36	.88	.17	.70

We see similar skewness across the datasets in regards to the NO% as seen in RQ1. We obtain precision vales higher than 0.80 for almost all projects, work item types and stages. In some cases, precision goes as high as 0.97. Of course, recall is lower than seen in RQ1, usually between 0.15 and 0.35. Though, in some cases, recall values are as high as 0.56.

When comparing the weighted average of the cost sensitive (Table IV) and the cost insensitive learning (Table III) process results, we observe that without any exception, WA of the cost sensitive predictions are always equal to or greater than cost insensitive predictions. This shows that the cost sensitive models are overall more accurate while providing the high precision values desired by IBM.

Answer to RQ2: We developed models that achieved higher precision values, up to 0.97 (see Table IV), and obtained higher overall performance compared to RQ1 as measured by the weighted average scores.

C. Feature Importance (RQ3)

RQ3: What are the features that can be used in this prediction and how important are they relatively?

Table V shows detailed information on relative feature importance for the model trained on each dataset and Table VI shows three aggregated rankings over all datasets using three different criteria. When interpreting these results, one should consider that the importance of the features in each dataset can be influenced by:

- The availability rate and cardinality of a feature. Some features might have a high number of missing values in some stages. It is also possible that some features have similar values and a low cardinality and, thus, provide lower information gain.
- Other dominant features. These feature importance values are relative to other features within the same dataset. A change in relative importance does not indicate a change in the absolute importance of that feature.
- The randomness factor of the random forest algorithm in selecting subsets of training samples as well as features when building individual trees [33].

As can be seen in Table VI , feature importance is highly project and prediction stage dependent. The only feature that is globally almost always highly important across all projects is ITERATION_DAYS_REMAINED. This is not surprising as regardless of project, the learner wants to know how much time is left to the end of iteration. CREATOR_IDENTIFIER also tends to be important in most datasets, which is in alignment with the findings of prior work [14]. We also found that DAYS_SINCE_LAST_SUMMARY, ITERATION_CHANGE_COUNT, and DAYS_WITHOUT_OWNER have high importance overall. LARGE_SIZE_CHILD_COUNT_NEW is also important in all projects as it is a good indications of how much work needs to be done.

Answer to RQ3: For the most part, feature importance is highly project and prediction stage dependent. Though, there are some features that are almost consistently ranked higher than others. Tables V and VI show the relative importance of the 29 features included in our models.

TABLE V
RELATIVE VARIABLE IMPORTANCE RANKING PER DATASET

attribute	plan												story											
	A				B				C				A				B				C			
	0th	1st	2nd	3rd	0th	1st	2nd	3rd	0th	1st	2nd	3rd	0th	1st	2nd	3rd	0th	1st	2nd	3rd	0th	1st	2nd	3rd
creator_identifier	1	13	4	3	14	2	5	27	25	28	26	26	2	3	4	1	2	26	1	1	6	17	2	6
creation_month	11	4	27	4	1	18	7	15	28	12	11	8	28	10	19	13	3	18	4	6	23	8	5	26
owner_identifier	6	28	28	2	28	25	28	28	27	7	25	21	29	1	2	20	29	27	15	15	29	26	7	1
subscriber_count	14	3	8	28	18	26	11	23	15	2	9	9	27	24	10	11	5	3	12	25	7	13	13	8
filed_against	2	18	9	18	4	10	25	25	1	8	19	25	21	26	3	29	4	7	29	19	4	15	27	10
iteration_change_cnt	17	5	5	21	6	7	15	11	6	10	21	15	3	15	5	2	19	24	5	20	24	20	19	4
iteration_days_remained	9	1	2	1	25	9	4	3	2	1	1	1	1	19	22	6	1	6	17	23	1	7	18	29
days_since_creation	28	15	18	27	24	21	1	2	12	22	23	20	17	23	29	27	17	17	28	14	5	23	3	16
status	4	26	16	19	27	24	2	9	24	19	5	13	13	11	24	14	11	8	3	13	2	21	29	13
priority	26	21	23	11	26	28	8	14	16	23	13	17	18	21	12	25	23	2	24	10	3	22	12	21
severity	15	8	21	16	3	15	24	24	13	15	16	3	12	5	23	8	28	22	23	29	11	27	8	7
days_without_owner	21	11	6	23	10	5	6	19	10	27	6	14	7	27	27	10	18	1	2	3	28	4	14	22
days_since_last_comment	3	27	24	20	2	27	19	22	18	20	17	5	25	6	9	3	24	13	26	26	15	12	21	27
owner_change_cnt	18	2	1	5	7	8	9	13	7	26	28	19	4	28	26	17	20	14	20	11	25	19	23	11
days_since_last_owner	22	24	7	26	11	13	23	26	3	11	8	4	8	12	21	12	13	4	6	12	18	29	11	20
summary_change_cnt	20	23	13	25	9	16	10	21	9	24	24	24	6	25	7	7	22	19	9	8	27	3	4	19
description_change_cnt	19	17	17	13	8	12	12	1	8	16	15	22	5	7	20	26	21	15	16	16	26	1	16	12
days_since_last_summary	23	19	3	17	12	19	26	5	4	3	3	2	9	22	15	23	14	16	14	4	19	10	1	23
days_since_last_description	24	12	20	24	13	3	16	6	5	14	2	18	10	16	17	24	15	5	21	7	20	14	15	17
days_since_last_type_s_p	7	20	22	12	15	17	14	7	19	17	14	10	14	8	8	18	7	20	19	22	12	9	10	24
days_since_last_type_child	8	25	25	14	16	14	20	8	20	21	20	7	15	14	11	21	8	21	25	27	13	16	20	25
days_since_last_de_comment	5	16	11	7	5	6	18	10	26	18	22	23	23	20	16	16	27	25	27	24	16	18	6	15
component_resolver	10	14	12	8	21	4	17	20	14	13	27	12	24	4	1	9	6	9	7	18	21	11	22	14
creator_team_relationship	16	22	15	15	17	1	3	18	17	25	4	16	22	2	6	28	26	28	11	17	9	6	9	18
comment_count	12	10	10	6	19	11	13	4	21	9	12	27	20	13	14	4	10	29	18	21	17	2	26	9
commenter_count	13	9	19	9	20	23	22	12	22	6	10	28	11	18	28	15	9	23	13	28	14	5	24	5
same_type_child_count_new	18	14	21	22	29	13	9	8	22	19	17	29	19	17	25	22	25	10	8	5	22	24	17	28
large_size_child_count_new	25	7	9	10	22	20	21	16	23	4	18	11	16	9	13	5	16	11	10	2	10	25	28	3
medium_size_child_count_new	27	6	14	22	23	22	29	17	11	5	7	6	26	29	18	19	12	12	22	9	8	28	25	2

TABLE VI
AGGREGATED RELATIVE VARIABLE IMPORTANCE RANKING

Rank by frequency of being among top 10	cnt
iteration_days_remained	17
creator_identifier	15
days_without_owner	12
iteration_change_count	11
filed_against	11
creation_month	11
large_size_child_count_new	10
days_since_last_summary	10
summary_change_count	10
subscriber_count	10
component_resolver	9
comment_count	9
owner_change_count	8
days_since_last_type_s_p	8
creator_team_relationship	8
medium_size_child_count_new	7
commenter_count	7
owner_identifier	7
status	7
days_since_last_owner	7
days_since_last_description	7
severity	7
days_since_last_comment	6
days_since_last_de_comment	6
description_change_count	6
same_type_child_count_new	5
days_since_last_type_child	4
priority	4
days_since_creation	4

Rank by frequency of being among top 5	cnt
creator_identifier	13
iteration_days_remained	12
days_since_last_summary	8
iteration_change_count	7
filed_against	6
creation_month	6
days_without_owner	5
status	5
large_size_child_count_new	4
days_since_last_description	4
days_since_creation	4
days_since_last_comment	4
subscriber_count	4
owner_change_count	4
owner_identifier	4
creator_team_relationship	4
component_resolver	3
comment_count	3
description_change_count	3
days_since_last_owner	3
severity	3
medium_size_child_count_new	2
days_since_last_de_comment	2
priority	2
commenter_count	2
summary_change_count	2
same_type_child_count_new	1
days_since_last_type_s_p	0
days_since_last_type_child	0

Rank by average rank	avg
iteration_days_remained	8.7
creator_identifier	10.2
iteration_change_count	12.5
days_since_last_summary	12.8
creation_month	12.9
component_resolver	13.3
days_without_owner	13.4
subscriber_count	13.5
large_size_child_count_new	13.9
comment_count	14
days_since_last_description	14.1
description_change_count	14.2
days_since_last_owner	14.3
days_since_last_type_s_p	14.4
status	14.6
creator_team_relationship	14.6
filed_against	14.9
owner_change_count	15
summary_change_count	15.6
severity	15.7
commenter_count	16.1
medium_size_child_count_new	16.6
days_since_last_de_comment	16.7
days_since_last_comment	17.1
days_since_last_type_child	17.3
priority	17.5
same_type_child_count_new	18.4
days_since_creation	18
owner_identifier	18.9

VI. Discussion

In this study, we developed and evaluated a machine learning process, filling a gap in software release planning research. Existing research performs bug fix effort estimation and release readiness prediction. We proposed a predictive modeling process including a set of 29 features and machine learning techniques that predicts likelihood of requirement implementation within the planned iteration. This modeling process was validated on three projects with satisfying results. Although all studied projects are within the same company, many of the proposed features are general enough to be applicable to projects from other companies. Besides the features themselves, we grouped the 29 features in nine categories which, due to a higher abstraction level, represent a starting point for other organizations to engineer their own features.

We have also ranked the features according to their relative importance to the trained model. Although we observed that the importance of features is dependent on prediction stage and project even within the same company, we showed that there are certain features that are almost consistently ranked higher than others across the different projects and stages of development. The implication of these results are a set of candidate features to receive further attention by researchers and practitioners.

Another implication of this study is related to the precision-recall trade-off. In general, we would like to have predictive models with both high precision and recall, and there is a general belief that models with low precision or recall are practically useless [36]. However, as Menzies et al. [37] argue, there are certain practical use cases for models with low precision and high recall, such as in the defect prediction domain. In our study, however, our claim for the usefulness of models with high precision and relatively low recall is based on our industrial partner's interest for high precision at the expense of recall. We also showed that machine learning techniques, in particular cost sensitive learning, could be adopted to satisfy such requirements.

VII. Threats to Validity

Like any empirical study, our paper is prone to some threats to validity [38]. We describe them below, together with our mitigation strategies. The first threat, to external validity, is the potential lack of generalizability of our findings due to the close connection with the data and respondents from only one software organization. We addressed this threat by studying three different projects, adopting techniques and many of the features as inspired from prior work. The model feature categories in our model are also intended to be broad enough to be applicable to other organizations and their project data.

The second threat, to construct validity, concerns our potential misinterpretation of the workflows within the IBM ecosystem. This was mitigated by prolonged close connection with IBM practitioners and through iterative and feedback-driven design of our model and machine learning techniques.

The third threat, to internal validity, relates to whether the results really do follow from the data in our study. Specifically, whether the features are meaningful to our prediction outcome and whether the performance estimations are realistic. In our study, we performed manual feature engineering through iterative cycles of conducting interviews and using the domain knowledge of the data and avoided automatic means in feature selection to avoid bias. Moreover, we performed LOOCV as recommended in effort estimation literature [30] to reduce bias and variance of our performance estimations.

The fourth threat to validity concerns the reliability and reproducibility of our findings. We have done our best to provide a clear methodology and specify design decisions, so that as long as other researchers study the same data following the same study settings, same results should be achieved.

VIII. Conclusion and Future Work

In this work, we studied the problem of predicting requirement implementation within the planned iteration using a combination of qualitative and quantitative methods on three large IBM projects to make machine learning based predictions at four meaningful stages of a requirement lifetime. We engineered a set of 29 features and put them in nine logical categories and trained various models and achieved acceptable results. Then, we modified the learning process to maximize precision of predictions according to the business interest of our industrial partner and achieved precision values between 0.8 and 0.97 depending on the project and prediction stage while retaining a recall value between 0.15 and 0.6 depending on the project. We then ranked features based on their relative importance to the trained model and observed that although importance of features is highly project and prediction stage dependent, there are certain features, such as creator of the requirement, time remained to the end of iteration, time past since last requirement summary change and number of times requirement has been replanned for a new iteration, that are overall of high importance to most projects and prediction stages.

Future work of this study includes studying new features and analyzing results when new features are included. One of the techniques that is not incorporated in this study is text analysis on attributes such as summary, description and comments of a work item that has shown successful results in previous work [9], [11]. Although most previous models using text analysis have been solely relied on text attributes, there has been prior work that integrated them with other methods successfully using ensemble techniques [12] or by transforming text attributes into numeric features [39], [8]. This opens an avenue for future work as relying on different sources of data and combining their estimates is likely to produce better estimates compared to each individual one [40]. Other ensemble learning techniques have also been used successfully in effort estimation literature [41], [42], [43] which could be applied on our research settings to improve the results. Finally, analyzing potential influences of the geographical distribution of project stakeholders [14], as well as other factors related to their communication and coordination are worthwhile directions for future work.

ACKNOWLEDGMENT

Special thanks to Fabio Calefato from University of Bari, Alan Yeung from Persistent Systems and the IBM team.

REFERENCES

[1] A. Alam, S. Didar, D. Pfahl, and G. Ruhe, "Release readiness classification: An explorative case study," in *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. ACM, 2016, p. 27.

[2] S. D. Al Alam, M. R. Karim, D. Pfahl, and G. Ruhe, "Comparative analysis of predictive techniques for release readiness classification," in *Realizing Artificial Intelligence Synergies in Software Engineering (RAISE), 2016 IEEE/ACM 5th International Workshop on*. IEEE, 2016, pp. 15–21.

[3] L. D. Panjer, "Predicting eclipse bug lifetimes," in *Proceedings of the Fourth International Workshop on mining software repositories*. IEEE Computer Society, 2007, p. 29.

[4] 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*. ACM, 2010, pp. 52–56.

[5] W. Abdelmoez, M. Kholief, and F. M. Elsalmy, "Bug fix-time prediction model using naïve bayes classifier," in *Computer Theory and Applications (ICCTA), 2012 22nd International Conference on*. IEEE, 2012, pp. 167–172.

[6] 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 Engineering*. ACM, 2011, p. 11.

[7] A. N. Duc, D. S. Cruzes, C. Ayala, and R. Conradi, "Impact of stakeholder type and collaboration on issue resolution time in oss projects," in *IFIP International Conference on Open Source Systems*. Springer, 2011, pp. 1–16.

[8] R. Kikas, M. Dumas, and D. Pfahl, "Using dynamic and contextual features to predict issue lifetime in github projects," in *Proceedings of the 13th International Conference on Mining Software Repositories*. ACM, 2016, pp. 291–302.

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

[10] S. Assar, M. Borg, and D. Pfahl, "Using text clustering to predict defect resolution time: a conceptual replication and an evaluation of prediction accuracy," *Empirical Software Engineering*, vol. 21, no. 4, pp. 1437–1475, 2016.

[11] D. Pfahl, S. Karus, and M. Stavnycha, "Improving expert prediction of issue resolution time," in *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering*. ACM, 2016, p. 42.

[12] A. Dehghan, K. Blincoe, and D. Damian, "A hybrid model for task completion effort estimation," in *Proceedings of the 2nd International Workshop on Software Analytics*. ACM, 2016, pp. 22–28.

[13] L. L. Minku, E. Mendes, and B. Turhan, "Data mining for software engineering and humans in the loop," *Progress in Artificial Intelligence*, vol. 5, no. 4, pp. 307–314, 2016.

[14] P. J. Guo, T. Zimmermann, N. Nagappan, and B. Murphy, "Characterizing and predicting which bugs get fixed: an empirical study of microsoft windows," in *Software Engineering, 2010 ACM/IEEE 32nd International Conference on*, vol. 1. IEEE, 2010, pp. 495–504.

[15] H. Zeng and D. Rine, "Estimation of software defects fix effort using neural networks," in *Computer Software and Applications Conference, 2004. COMPSAC 2004. Proceedings of the 28th Annual International*, vol. 2. IEEE, 2004, pp. 20–21.

[16] U. Raja, "All complaints are not created equal: text analysis of open source software defect reports," *Empirical Software Engineering*, vol. 18, no. 1, pp. 117–138, 2013.

[17] 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*. ACM, 2011, pp. 207–210.

[18] M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim, "Do we need hundreds of classifiers to solve real world classification problems," *J. Mach. Learn. Res*, vol. 15, no. 1, pp. 3133–3181, 2014.

[19] F. Yang, H.-z. Wang, H. Mi, W.-w. Cai *et al.*, "Using random forest for reliable classification and cost-sensitive learning for medical diagnosis," *BMC bioinformatics*, vol. 10, no. 1, p. S22, 2009.

[20] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Transactions on knowledge and data engineering*, vol. 21, no. 9, pp. 1263–1284, 2009.

[21] 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*, vol. 11, no. 1, pp. 10–18, 2009.

[22] K. M. Ting, "An instance-weighting method to induce cost-sensitive trees," *IEEE Transactions on Knowledge and Data Engineering*, vol. 14, no. 3, pp. 659–665, 2002.

[23] R. Klinkenberg, *RapidMiner: Data mining use cases and business analytics applications*. Chapman and Hall/CRC, 2013.

[24] P. Domingos, "A few useful things to know about machine learning," *Communications of the ACM*, vol. 55, no. 10, pp. 78–87, 2012.

[25] A. S. Selya, J. S. Rose, L. C. Dierker, D. Hedeker, and R. J. Mermelstein, "A practical guide to calculating cohens f2, a measure of local effect size, from proc mixed," *Frontiers in psychology*, vol. 3, p. 111, 2012.

[26] H. Liu and R. Setiono, "Chi2: Feature selection and discretization of numeric attributes," in *Tools with artificial intelligence, 1995. proceedings., seventh international conference on*. IEEE, 1995, pp. 388–391.

[27] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen, *Classification and regression trees*. CRC press, 1984.

[28] S. Fortmann-Roe, "Understanding the bias-variance tradeoff," 2012.

[29] B. Efron, "Estimating the error rate of a prediction rule: improvement on cross-validation," *Journal of the American Statistical Association*, vol. 78, no. 382, pp. 316–331, 1983.

[30] E. Kocaguneli and T. Menzies, "Software effort models should be assessed via leave-one-out validation," *Journal of Systems and Software*, vol. 86, no. 7, pp. 1879–1890, 2013.

[31] L. Song, L. L. Minku, and X. Yao, "The impact of parameter tuning on software effort estimation using learning machines," in *Proceedings of the 9th international conference on predictive models in software engineering*. ACM, 2013, p. 9.

[32] B. N. Province, "The effects of parameter tuning on machine learning performance in a software defect prediction context," Ph.D. dissertation, WEST VIRGINIA UNIVERSITY, 2015.

[33] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[34] P. Ramarao, K. Muthukumaran, S. Dash, and N. B. Murthy, "Impact of bug reporter's reputation on bug-fix times," in *Information Systems Engineering (ICISE), 2016 International Conference on*. IEEE, 2016, pp. 57–61.

[35] G. Jeong, S. Kim, and T. Zimmermann, "Improving bug triage with bug tossing graphs," 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. 111–120.

[36] H. Zhang and X. Zhang, "Comments on" data mining static code attributes to learn defect predictors"," *IEEE Transactions on Software Engineering*, vol. 33, no. 9, 2007.

[37] T. Menzies, A. Dekhtyar, J. Distefano, and J. Greenwald, "Problems with precision: A response to" comments on'data mining static code attributes to learn defect predictors'"," *IEEE Transactions on Software Engineering*, vol. 33, no. 9, 2007.

[38] S. Easterbrook, J. Singer, M.-A. Storey, and D. Damian, "Selecting empirical methods for software engineering research," in *Guide to advanced empirical software eng.* Springer, 2008, pp. 285–311.

[39] Y. Zhou, Y. Tong, R. Gu, and H. Gall, "Combining text mining and data mining for bug report classification," *Journal of Software: Evolution and Process*, 2016.

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

[41] 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.

[42] D. Azhar, P. Riddle, E. Mendes, N. Mittas, and L. Angelis, "Using ensembles for web effort estimation," in *Empirical Software Engineering and Measurement, 2013 ACM/IEEE International Symposium on*. IEEE, 2013, pp. 173–182.

[43] A. Idri, M. Hosni, and A. Abran, "Systematic literature review of ensemble effort estimation," *Journal of Systems and Software*, vol. 118, pp. 151–175, 2016.