2018

Developing analytics models for software project management

Morakot Choetkiertikul
University of Wollongong

UNIVERSITY
OF WOLLONGONG
AUSTRALIA

Developing analytics models for software project management

Morakot Choetkiertikul B.Sc.(ICT), M.Sc.(CS)

This thesis is presented as part of the requirements for the conferral of the degree:

Doctor of Philosophy

Supervisor:
Dr. Hoa Khanh Dam

Co-supervisor:
Prof. Aditya Ghose

The University of Wollongong
School of Computing and Information Technology

September 9, 2018

Declaration

I, *Morakot Choetkiertikul B.Sc.(ICT), M.Sc.(CS)*, declare that this thesis submitted in partial fulfilment of the requirements for the conferral of the degree *Doctor of Philosophy*, from the University of Wollongong, is wholly my own work unless otherwise referenced or acknowledged. This document has not been submitted for qualifications at any other academic institution.

Morakot Choetkiertikul B.Sc.(ICT), M.Sc.(CS)

September 9, 2018

Abstract

Schedule and cost overruns constitute a major problem in software projects and have been a source of concern for the software engineering community for a long time. Managing software projects to meet time and cost constraints is highly challenging, due to the dynamic nature of software development. Software project management covers a range of complex activities such as project planning, progress monitoring, and risk management. These tasks require project managers the capability to make correct estimations and foresee future significant events in their projects. However, there has been little work on providing automated support for software project management activities.

Modern software projects mostly follow the iterative development process where software products are incrementally developed and evolved in a series of iterations. Each iteration requires the completion and resolution of a number of issues such as bugs, improvement or new feature requests. Since modern software projects require continuous deliveries in every iteration of software development, it is essential to monitor the execution of iterations and the resolution/completion of issues, and make reliable predictions. There is thus a strong need to provide the project managers, software engineers, and other stakeholders with predictive support at the level of iterations and issues.

This thesis aims to leverage a large amount of data from software projects to generate actionable insights that are valuable for different software project management activities at the level of iterations and issues. Using cutting-edge machine learning technology (including deep learning), we develop a novel suite of data analytics techniques and models for predicting delivery capability of ongoing iterations, predicting issue delays, and estimating the effort for resolving issues. An extensive empirical evaluation using data from over ten large well-known software projects (e.g., Apache, Duraspace, Java.net, JBoss, JIRA, Moodle, and Mulesoft) demonstrates the high effectiveness of our approach.

Acknowledgments

First of all, I would like to express my deepest appreciation to my supervisor, Hoa Khanh Dam, for his patient guidance, encouragement, and excellent opportunities to complete my Ph.D. thesis. At many stages in the course of this research, I encountered many difficult problems and challenges. He has never turned his back on me but patiently guided me through the most difficult times. He always believes in my ability and keeps pushing me forward with his strong determination to achieve our goals. I have learned not only scientific knowledge from him, but also how to be a good scientist. It would never be possible for me to complete this thesis without his incredible support.

I am also very grateful to Prof. Aditya Ghose, my co–supervisor, for his constant support and encouragements. He always gives constructive suggestions which lead to the accomplishment of this thesis. I also would like to thank Dr. Truyen Tran and Trang Pham at Deakin University, Australia for providing the excellent guidance and sharing their expertise which helped me in the first steps of machine learning and deep learning in the development of this research work.

Special thanks go to Prof. Tim Menzies at North Carolina State University, and Prof. John Grundy at Monash University for their valuable colloborations.

This work would not have been possible without the financial support from the Faculty of Information and Communication Technology (ICT), Mahidol University, Thailand. I especially thank to Assoc. Prof. Dr. Jarernsri L. Mitrpanont, Dean of Faculty of Information and Communication Technology, who has been supportive of my academic and career goals, and who always guides me to pursue those goals.

I would like to thank my colleagues and friends in the Decision Systems Lab (DSL) for their friendly companionship and for providing a nice and friendly working environment. Especially, my labmate: Alexis Harper, Yingzhi Gou, and Josh Brown who usually organized the game and movie nights which helped me overcome some depressing moments from rejections. I also thank the administrative and technical support staff at UOW, who helped me in various stages of my research.

I am truly grateful to my parents: Manit and Chuleeporn Choetkiertikul. Without the support of my family, this thesis would not have been possible. I thank them so much for their continuous and unconditional support. They always give me a courage and strong determination to overcome any difficult challenges in my life and keep pushing me forward to pursue my dream.

At last, I wish to thank many other people whose names are not mentioned here but this does not mean that I have forgotten their help.

Publications

Earlier versions of the work in this thesis were published as listed below:

- *Predicting delivery capability in iterative software development.*
  Morakot Choetkiertikul, Hoa Khanh Dam, Truyen Tran, Aditya Ghose, and John Grundy. **IEEE Transactions on Software Engineering (IEEE TSE)**, 2017, DOI: 10.1109/TSE.2017.2693989, IEEE. [Chapter 3]

- *Characterization and prediction of issue-related risks in software projects.*
  Morakot Choetkiertikul, Hoa Khanh Dam, Truyen Tran, and Aditya Ghose. In Proceedings of the 12th **Working Conference on Mining Software Repositories (MSR)**, co-located with the International Conference on Software Engineering (ICSE), 2015, pages 280 – 291, IEEE. [Chapter 4]
  Received the ACM SIGSOFT Distinguished Paper Award

- *Predicting the delay of issues with due dates in software projects.*
  Morakot Choetkiertikul, Hoa Khanh Dam, Truyen Tran, and Aditya Ghose.
  Empirical Software Engineering journal (EMSE), Volume 22, Issue 3, pages 1223-1263, 2017, Springer. [Chapter 4]

- *Predicting delays in software projects using networked classification.*
  Morakot Choetkiertikul, Hoa Khanh Dam, Truyen Tran, and Aditya Ghose. In Proceedings of the 30th **IEEE/ACM International Conference on Automated Software Engineering (ASE)**, 2015, pages 353 - 364, IEEE. [Chapter 5]

- *A deep learning model for estimating story points.*
  Morakot Choetkiertikul, Hoa Khanh Dam, Truyen Tran, Trang Pham, Aditya Ghose, and Tim Menzies. **IEEE Transactions on Software Engineering (IEEE TSE)**, 2018, DOI: 10.1109/TSE.2018.2792473, IEEE. [Chapter 6]

The following publications are not directly related to the work that is presented in this thesis. They were produced in parallel to the research performed in this thesis.

- *Poster: Predicting components for issue reports using deep learning with information retrieval.*
  Morakot Choetkiertikul, Hoa Khanh Dam, Trang Pham, Truyen Tran, and Aditya Ghose. In Proceedings of the 40th **International Conference on Software Engineering (ICSE)**, 2018.

- *A CMMI-based automated risk assessment framework.*
  Morakot Choetkiertikul, Hoa Khanh Dam,Aditya Ghose, and Thanwadee T. Sunetnanta. In Proceedings of the **International Workshop on Quantitative Approaches to Software Quality**, co-located with the Asia-Pacific Software Engineering Conference (APSEC), 2014, volume 2, pages 63 - 68, IEEE.

- *Threshold-based risk identification in software projects.*
  Morakot Choetkiertikul, Hoa Khanh Dam, and Aditya Ghose. In Proceedings of the 24th **Australasian Software Engineering Conference (ASWEC)**, 2015, pages 81 - 85, ACM.

- *Who will answer my question on Stack Overflow?*
  Morakot Choetkiertikul, Daniel Avery, Hoa Khanh Dam, Truyen Tran, and Aditya Ghose. In Proceedings of the 24th **Australasian Software Engineering Conference (ASWEC)**, 2015, pages 155 - 164, IEEE.

- *Predicting issues to be resolved for the next release.*
  Shien Wee Ng, Hoa Khanh Dam, Morakot Choetkiertikul, and Aditya Ghose. In Proceedings of the Sixth **Australasian Symposium on Service Research and Innovation (ASSRI)**, 2017.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

LATE delivery and cost overruns have been a common problem in software projects for many years. A recent study by McKinsey and the University of Oxford in 2012 of 5,400 large scale IT projects found that on average 66% of IT projects were over budget and 33% went over the scheduled time [1]. Managing software projects to meet the expected cost and time constraints is highly challenging, due to the inherent dynamic nature of software development (e.g. constants changes to software requirements). Current support for software project managers is however limited, especially at the fine-grained levels of iterations and tasks. This thesis proposes a novel suite of data analytics techniques and models to better support project managers, team leaders, and other decision makers in software development. We harvest valuable insights buried under the huge amount of software development data to build prediction models for delays, delivery capability, and effort estimation.

Today's software development is trending toward agility and continuous delivery. Even large software systems, e.g. Microsoft Windows [2], are moving from major releases to a stream of continuous updates. This is a shift from a model where all functionalities are shipped together in a single delivery to a model involving a series of incremental, continuous deliveries of a small number of functionalities. Modern software development projects are mostly based on the incremental and iterative process where a software is designed, developed, and tested in repeated cycles (i.e. the so-called "iterations") [3]. The iterative development allows software developers to gain benefits from what were learned during the development of earlier parts or versions of the software. An iteration is usually limited to a certain short time length. At the end of an iteration, deliverable software packages which may contain new features and/or bug fixes are expected to be delivered [4].

Project management is still critical to the success of today's software projects regardless of which development process is employed. It typically involves many complex activities ranging from planning, estimating to progress monitoring of iterations and releases, and dealing with risks. These require project managers the capability to make correct estimation and foresee if their team can deliver on time, within budget and with all planned features. There has been a large body of work on building various prediction models to support software development. For example, existing effort estimation models (e.g. [5], [6], [7], and [8]) was built to predict the effort required for developing a whole software. Other existing empirical work has however considered how prediction was done at the project level (e.g. software project risk prediction [9]–[12]).

A substantial amount of work has also been proposed in many other aspects of software engineering (e.g. bug localization [13]–[17], defect prediction [18], [19], software vulnerability prediction [20], and bug severity prediction [21]–[23]). However, there has been very little work on providing automated support in software project management activities. It would be valuable for project managers and decision makers to be provided with insightful and actionable information at the level of iterations and tasks (rather than just only at the project level). For example, being able to predict that an iteration is at risk of not delivering what has been planned allows project managers the opportunity adapt a current plan earlier, e.g. moving some features from the current iteration to the next one.

This thesis aims to fill this gap. The overall objective of this research is providing decision makers in modern software projects with actionable insights by leveraging a huge amount of data from software projects. We specifically focus on data generated from fixing bugs, and implementing new features and improvements which usually recorded as *issues* in an issue tracking system (e.g. JIRA software[a]). We leverage state-of-the-art machine learning techniques (including deep learning) to develop a suite of analytics models which makes use of both structured data and unstructured data (e.g. textual description of an issue). Our predictive models are capable of offering accurate predictions at the fine-grained levels of iterations and issues.

1.1 Research questions

Iterative software development has become widely practiced in industry. Since modern software projects require fast, incremental delivery for every iteration of software devel-

[a]https://www.atlassian.com/software/jira

opment, it is essential to monitor the execution of an iteration and foresee whether teams can deliver quality products as the iteration progresses. Hence, our focus here is on predicting the delivery capability of each iteration at a time. The first research question we would like to address is:

- **Research question 1**: How to leverage a history of project iterations and their associated issues to provide automated support for project managers and other decision makers in predicting delivery of an ongoing iteration?

In practice, to facilitate project planning and maintaining the progression of a project, an issue (also referred as a project task) usually have a certain deadline. A deadline can be imposed on an issue by either explicitly assigning a due date to it, or implicitly assigning it to a release and having it inherit the release's deadline. It is important for project managers to ensure as many issues be completed in time against their respective due date as possible to avoid adverse implications (e.g. delayed issue effects) to the overall progress of a project. Our second research question thus focuses on providing this kind of support.

- **Research question 2a**: How to analyze the historical data associated with a software project to predict whether a current issue is at risk of being delayed?

An issue may have relationships with other issues. For example, an issue needs to be resolved before the resolution of another issue can be started. Those relationships usually determine the order in which issues should be resolved. This information provides valuable information for predicting delays. For example, if an issue blocks another issue and the former is delayed, then the latter is also at risk of getting delayed. This example demonstrates a common delay propagation phenomenon in software projects. Thus, the third research question we would like to address is:

- **Research question 2b**: How to leverage not only features specific to individual issues (i.e. local data) – as address in Research question 2a – but also their relationships (i.e. networked data) to improve the predictive performance of the delay prediction model?

To ensure software projects complete in time and within budget, effort estimation is a critical part of project planning [24]–[26]. Incorrect estimates may have adverse impact on the project outcomes (e.g. project delays) [24], [27]–[29]. Although there has been substantial research in software analytics for effort estimation in traditional software projects, little work has been done for estimation in agile projects, especially estimating the effort required for completing user stories or issues. Story points are the most common unit of measure used for estimating the effort involved in completing a user story or resolving an issue. Currently, most agile teams heavily rely on experts' subjective assessment (e.g. planning poker, analogy, and expert judgment) to arrive at an estimate. This may lead to inaccuracy and more importantly inconsistencies between estimates [30]. Hence, our last research question aims to facilitate effort estimation at the issue level:

- **Research question 3**: How to develop a highly accurate estimation model which is able to recommend a story-point estimate of an issue using the team's past estimation?

1.2 Research outcomes and main contributions

Our general framework makes use of historical data that is stored in issue tracking systems such as JIRA software[b] (see Figure 1.1). We have collected iteration and issue reports from over ten well-known large open source repositories (e.g. Apache, Appcelerator, DuraSpace, Atlassian, Moodle, MongoDB, and JBoss). We have analyzed the collected data and studied their characteristics (e.g. structure) to answer each research question. We have then performed data pre-processing (e.g. data cleansing and data labeling) to build these datasets for our studies. The next step involves developing predictive models which focuses on feature engineering and model building. Feature engineering involves exploring, analyzing, and extracting the datasets to create a set of features. Machine learning techniques are employed in order to build a predictive model. The features are then used for training the model. Finally, an extensive evaluation is performed to measure the performance of the predictive models. In the evaluation, we compare the predictive performance of our models against alternative methods in order to demonstrate the effectiveness of the approach over the existing models and techniques.

This thesis makes several contributions to state of the art as follows. Firstly, we propose a novel, data-driven approach to providing automated support for project managers and other decision makers in predicting delivery capability for an ongoing iteration. Our approach leverages a history of project iterations and associated issues, and in particular, we have extracted characteristics of previous iterations and their associated issues in the form of features. In addition, our approach characterizes an iteration using a novel combination of techniques including feature aggregation statistics, automatic feature learning using the Bag-of-Words approach, and graph-based complexity measures.

[b]`https://www.atlassian.com/software/jira`

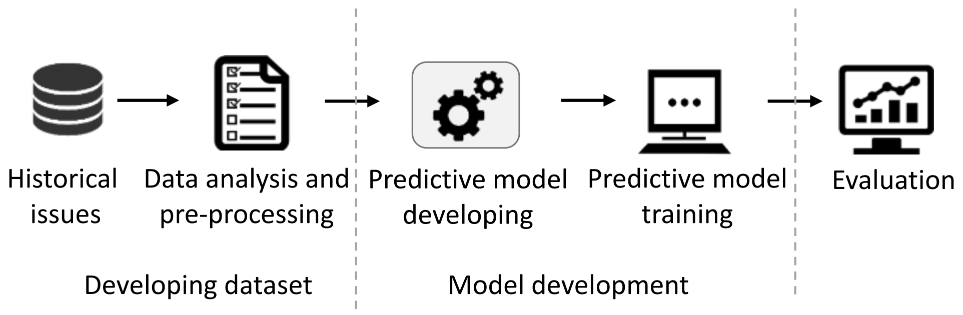

Historical issues
Data analysis and pre-processing
Predictive model developing
Predictive model training
Evaluation
Developing dataset
Model development

Figure 1.1: An overview of the research framework

We have then employed the state-of-the-art randomized ensemble methods (e.g. Random Forests) to build the predictive models. An extensive evaluation of the technique on five large open source projects (Apache, JBoss, JIRA, MongoDB, and Spring) demonstrates that our predictive models outperform three common baseline methods in Normalized Mean Absolute Error and are highly accurate in predicting the outcome of an ongoing iteration **(Research question 1)**.

Secondly, we have developed highly accurate models which are capable of predicting whether an issue is at risk of being delayed against its deadline. We have extracted the comprehensive set of features characterized delayed issues (e.g. discussion time, elapsed time from when the issue is created until prediction time, elapsed time from prediction time until the deadline, percentage of delayed issues that a developer involved with, and developer's workload) from 8 open source projects (Apache, Duraspace, Java.net, JBoss, JIRA, Moodle, Mulesoft, and WSO2). We have employed feature selection techniques to derive a set of features with good discriminative power to build predictive models. Our predictive models can predict both the impact of the delay and the likelihood of the delay occurrence. The evaluation results demonstrate the strong effectiveness of our predictive models in predicting if an ongoing issue has a delay risk **(Research question 2a)**.

Thirdly, the model we discussed earlier relies only on individual issues to make delay predictions. The dependencies among issues however exist and provide a good source of information for predicting delay. Thus, another contribution of this thesis is the enhancement of the delay prediction model by leveraging both the features of individual issues and their relationships (e.g. issue links). We have extracted various relationships between issues to build an issue network. Issue relationships can be explicit (those that are explicitly specified in the issue reports) or implicit (those that need to be inferred from other issue information). Explicit relations usually determine the order of issues, while implicit relations reflect other aspects such as issues assigned to the same developer, issues affecting the same software component or similar issues. We have developed the delay prediction models that make use of those issue relations. Our evaluation results show a significant improvement over traditional approaches which make the prediction based on each issue independently **(Research question 2b)**.

Fourthly, we have developed a highly accurate predictive models for estimating story points based on a novel combination of two powerful deep learning architectures: long short-term memory (LSTM) and recurrent highway network (RHN). LSTM allows us to model the long-term context in the textual description of an issue, while RHN provides us with a deep representation of that model. Our prediction system is end-to-end trainable from raw input data (i.e. textual description of issues) to prediction outcomes without any manual feature engineering. Our model learns from the team's previous estimations to recommend a story point of a new issue. An extensive evaluation demonstrates that our approach consistently outperforms common baselines in effort estimation and alternative techniques **(Research question 3)**.

Finally, we have developed a number of comprehensive datasets for our studies. The datasets consist of the delivery capability of iterations, the delay of issues, and story point estimation. Some of these datasets are the first dataset of its kind (e.g. the delivery capability dataset and issue delay dataset). In total, our datasets consist of over 200,000 issue reports and over 3,000 iteration reports. We have made our datasets publicly available[c], which we believe a significant contribution to the software engineering research community.

1.3 Thesis organization

This section provides a brief overview of the thesis and the general structure in each chapter. The remainder of this thesis is structured as follow:

- Chapter 2 provides background material and existing work related to this thesis. This chapter outlines the landscape of our research in software engineering analytics. It includes the background on machine learning and deep learning. This chapter also presents the characteristics of issue driven project management and issue's life-cycle, and also an overview of software analytics and its applications in software engineering.
- Chapter 3 describes the predictive model for predicting delivery capability in iteration software development in detail. This chapter presents our novel feature aggre-

[c] `https://github.com/SEAnalytics/datasets`

gation techniques to build the predictive model. This chapter is structured according to the steps in the framework discussed in Section 1.2. We first discuss the challenges and the problems we are going to tackle. It also elaborates the needs of the automated prediction model and emphasizes the main contributions of the chapter. We present the proposed approach and how to build the predictive models. We then discuss about the dataset used in this study that includes the data pre-processing step and the statistical information the dataset, before we elaborate how we run the extensive evaluations and report the results. Chapter 4 – 6 also follow the same structure.

- Chapter 4 presents our issue-level delay prediction model. This chapter provides a description of our approach to mine historical data, extracts features of an issue, and build the predictive model.
- Chapter 5 describes how we leverage the relationships of issues to enhance the delay prediction model. It also includes how we extract explicit and implicit relationships to construct an issue network and how we adopt the network classification model to learn those networked data.
- Chapter 6 presents our story point estimation model using deep learning techniques. It explains how we combine LSTM and RHN to build the prediction model and discusses how the model is trained using textual descriptions of a huge amount of issues.
- Chapter 7 summarizes this thesis's contributions and discusses future work.

Chapter 2

Background

THE main purpose of this chapter is to briefly provide the landscape of software analytics and give an overview of our research context. The first part of this chapter provides some background on machine learning techniques in Section 2.1 including the description of the techniques that we use in our research (e.g. deep learning). The second part of this chapter (Section 2.2) gives an overview of issue-driven project management including the characteristics of an issue and iteration. In addition, we explain the process of issue resolution and how issues are used in iterative development. The final section of this chapter (Section 2.3) briefly describes software analytics and its application in software engineering.

2.1 Machine learning

In this section, we briefly present the machine learning background that is necessary to this thesis. We briefly describe two important machine learning techniques: supervised and unsupervised learning in Section 2.1.1. We then focus on two classes of supervised learning: classification and regression problems in Section 2.1.2. We also briefly describe the common and well-known machine learning algorithms that have been widely used in software engineering (Section 2.1.3). We introduce the basic concepts of deep learning and the well-known deep learning architectures in Section 2.1.4.

Machine learning aims to enable computers to learn new behavior (e.g. classification) based on empirical data [31]. The goal of machine learning field is to design algorithms that allow the computer to solve a problem based on learning from human experience (e.g. historical data), rather than rely on human instructions. This learning process occurs through a learning algorithm. The different learning algorithms are designed to be as general purpose as possible. It can thus be applied to a broad class of learning problems. Machine learning has been utilized in many domains such as Internet security [32], [33], image recognition [34], [35], and on-line marketing [36], [37].

2.1.1 Supervised vs unsupervised learning

Machine learning algorithms can be classified into two major groups: supervised learning and unsupervised learning [38]. The supervised learning algorithms learn from many data points (also called instances and examples) contained in a dataset. Each data point is described by a number of input variables and an output variable. The input variables are also known as features that represent characteristics of a data point. The output variable (e.g. label) is the predicting target (e.g. category). In the learning process, the learning model infers a function based on a learning algorithm that can map input variables to an output variable. The supervised learning model that is supplied with labeled data points can then classify unlabeled data points based on their features. Unsupervised learning, in contrast, requires only a set of input variables without a target label. The unsupervised learning algorithms are able to find the structure or relationships between different input variables. There are two major unsupervised learning algorithms: clustering the data into groups based on similarity measures (e.g. k-means clustering) and reducing dimensionality to compress the data (e.g. Principal Component Analysis). Since we have mostly employed supervised learning algorithms to build prediction models, our discussion focuses on supervised learning.

To illustrate how supervised learning work, we examine the problem of deciding whether we should go hiking based on three features: weather condition, number of people, and trail difficulty (see Table 2.1). The features can be a categorical value (e.g. weather condition) or a numerical value (e.g. the number of people). Row 1 to 7 are labeled data points because these are the past decisions (also called ground-truth), while row 8 to 10 are unlabeled data points as these are our future hiking events. We would like to build a model that can support our decision making based on the past experienced (i.e. historical data). In supervised learning, the model approximates the relationship f between the three features X and the corresponding label y, defined as:

$$y = f(X)$$

Table 2.1: Example of a supervised learning problem

No.	weather condition	number of people	trail difficulty	go hiking?
1	good	3	hard	yes
2	good	1	hard	no
3	bad	4	easy	yes
4	good	4	easy	yes
5	bad	3	hard	no
6	good	3	hard	yes
7	bad	3	easy	no
8	good	2	easy	?
9	bad	1	hard	?
10	good	3	hard	?

One possibility for making the recommendation is to create a rule-based model to identify the relationships between the three features and the target output. For example, if the weather condition is `good`, the answer is `go`. However, from the past decisions, the data shows that we should not go hiking alone in a `hard` trail, even the weather condition is `good` (e.g. the second row). We thus need a number of rules to capture all possible combinations. Maintaining the consistency among those rules is difficult. The supervised learning solves this problem by identifying patterns in the data (i.e. features and labels) through a learning algorithm to derive f. The model f can predict y as long as the features X are available. Basically, to build a predictive model, labeled data points are used in training and testing of the model since the ground truths (i.e. known-label data) are required in order to measure the predictive performance. A dataset is usually separated into two mutually exclusive sets of data: training set and test set. The training set is used for learning (i.e. derive f). The trained model is then evaluated on the test set.

2.1.2 Classification vs regression

The supervised learning's goal is to predict y as accurately as possible, given a set of features X. The supervised learning problems can further be grouped into two tasks: regression and classification tasks [38]. A problem falls into the classification task when the output y is a categorical value (i.e. class label). In other words, the classification aims to assign a set of features X to a predefined class based on a trained classification model. For example, our example on deciding for hiking problem falls into the classification task because the output has two classes: `yes/no` (i.e. binary classification). We can also have multi-class classification task where the output has more than 2 classes (e.g. go hiking, do not go hiking, and postpone).

On the other hand, the regression task is when the output variable is a continuous numerical value. For example, we can change our problem in the example from classification to regression by changing the output from `yes/no` to the continuous score ranged from `0` to `10` representing the satisfaction level for the hike where the higher score, the more satisfaction. Most supervised learning algorithms (e.g. Decision Trees, Random Forests, Neural Network) can be employed in both classification and regression tasks. In the next section, we provide a brief discussion on common supervised learning algorithms that have been used in this thesis.

2.1.3 Learning algorithms

We briefly describe the common supervised learning algorithms as follows:

Decision Tree

Decision Tree is a supervised learning algorithm that can be represented as a tree graph model consisted of decision nodes and leaf nodes represented the target output (e.g. category). A decision node is a rule that is derived based on a feature that can split most data into their actual category [39]. The classification of an unknown data is achieved by passing it through the tree starting at the top and moving down until a leaf node is reached. The value at that leaf node gives the predicted output for the data. At each node, the branch is selected based on the value of the corresponding feature (see Figure 2.1). The significant benefit of decision tree classifier is that it offers an explainable model. Thus, most of the work that focuses on interpretable models (e.g.,[40], [41]) selected this technique. C4.5 is a well-known decision tree algorithm which can perform tree pruning to reduce the classification error [39].

Random Forests (RF)

Random Forests (RF) is a significant improvement of the decision tree approach by generating many classification trees, each of which is built on a random resampling of the data, with a random subset of variables at each node split. A prediction are then made by voting from many decision trees [42]. Random Forests comprises many weak models

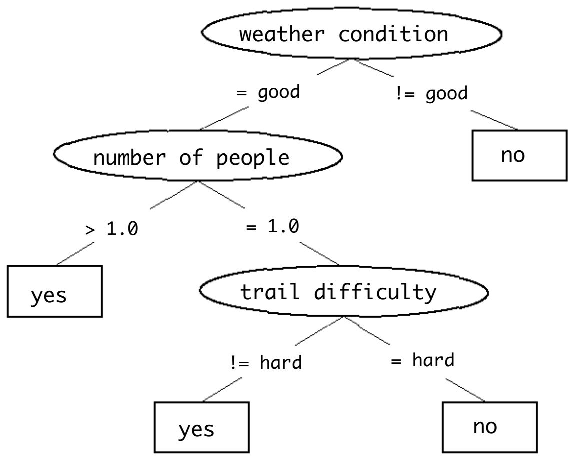


Figure 2.1: Example of a decision tree

(i.e. decision tree) that only learn a small portion of the data. This technique is also called an ensemble method. Previous research, [43] ,[44] has shown that RF is one of the best for a wide range of problems.

Gradient Boosting Machines (GBMs)

Gradient Boosting Machines (GBMs) are an ensemble method that combines multiple weak learners in an additive manner [45], [46]. At each iteration, a weak learner is trained to approximate the functional gradient of the loss function. A weight is then selected for the weak learner in the ensemble. Unlike Random Forests, a weak learner can be any function approximator (e.g. SVM and regression tree).

Support Vector Machines (SVM)

Support Vector Machines (SVMs) are successful in statistical learning theory. SVM represents data as points in space and uses a kernel (e.g. linear kernel, polynomial kernel, radial kernel) to learn the hyperplane that can separate the data points. The learned hyperplane is used to predict an output of the new (unknown) point. Originally, an SVM has been designed for binary classification (i.e. two classes) [47]. Multi-class SVM is an extension of SVMs that aims to extend binary classifiers to support multiclass by building binary classifiers between every pair of classes. The instance is classified to the highest score class [47].

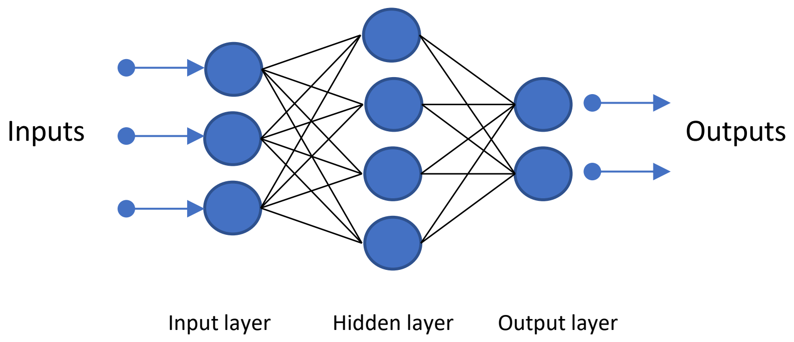


Figure 2.2: Artificial Neural Networks (aNN)

Naive Bayes (NB)

Naive Bayes (NB) is a probabilistic-based classifier. The NB algorithm builds a probabilistic model based on the Bayes's rule by learning the condition probabilities of each input feature given a possible value taken by the output [48]. Bayes's rule is defined as:

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}$$

where $P(A \mid B)$ is the probability of observing A given that B occurs, and $P(B \mid A)$ is the probability of observing B given A, and $P(A)$ and $P(B)$ is the individual probabilities of A and B [49]. The model bases on the assumption that features, when the outcome is known, are conditionally independent. Despite of this naive assumption, NB has been found to be effective as a classifier [48], [50]. NB offers a natural way to estimate the probability distributions. Thus, a study (e.g. [50], [51]) that aims to measure the degree of uncertainty usually employ NB to build models.

NBTree

NBtree is a hybrid algorithm which combines Naive Bayes classifier and C4.5 Decision Tree classifier together [52]. The decision tree consists of the root, the splitting, and the leave node. The root node denotes the starting point of the classification. The splitting is condition to separate data into two clusters. The leave nodes give the final results of the classification. At the leave nodes, NBTree uses the information on the frequency of classified instances to estimate probability using Naive Bayes.

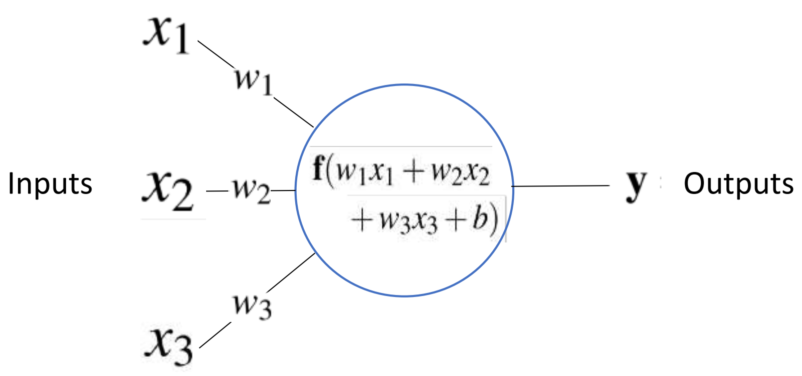


Figure 2.3: A single node in Artificial Neural Networks

Artificial Neural Networks (aNN)

Artificial Neural Networks (aNN) [53] is a model of feedforward recognition mimicking biological neurons and synapses. It provides nonlinear function approximation that maps an input vector into an output. aNN consists of neurons (nodes) arranged in layers. A node has connections between nodes from adjacent layers (see Figure 2.2). All these connections have weights associated with them. The input nodes at the input layer provide the data to the network. There is no computation at the input node. The information transformation is performed at the hidden node in the hidden layer. The output node in the output layer responses for transform information from the hidden nodes to give the output.

A hidden node and output node receive inputs from other nodes and computes an output. Each input has an associated weight (w). In the information transformation, the node applies an activation function **f** to the weighted sum of its inputs. Figure 2.3 shows an example of a single neuron node where the node takes inputs x_1, x_2, and x_3 and has weights w_1, w_2, and w_3 associated with those inputs. Note that b is a bias that is a trainable constant value.

The activation function **f** performs a mathematical operation on the weighted sum **x**. Several activation functions have been proposed [54]. The following activation functions are commonly used in practices.

- Sigmoid (*sigma*): an output is between 0 and 1.

$$\sigma(\mathbf{x}) = \frac{1}{(1 + exp(-\mathbf{x}))}$$

- tanh (*tanh*): an output is between -1 and 1.

$$tanh(\mathbf{x}) = 2\sigma(2\mathbf{x}) - 1$$

- Rectified Linear Unit (*ReLU*): This activation function replaces negative values with zero.

$$ReLU(\mathbf{x}) = max(0, \mathbf{x})$$

In training, the total error at the output layer is calculated. The error is then propagated back through the network (i.e. back propagation) [53]. All weights in the network are adjusted to reduce the error at the output layer. aNN is widely used in pattern recognition because of their flexibility as an universal function approximator and powerful generalization. In software engineering, there has been work applying aNNs, e.g., [55] ,[56] ,[57], which yields good results.

Deep Neural Networks with Dropouts (Deep Nets)

Deep Neural Networks with Dropouts (Deep Nets) is traditional artificial neural networks with multiple hidden layers. Recent advances include dropout [58], a simple yet powerful technique that allows hidden units and features to be randomly removed for each data point at each parameter update step which allows many networks are trained simultaneously. This creates an implicit ensemble of exponentially many networks without explicit storing of these networks. This is highly desirable because ensemble methods have been known to improve prediction generalizability.

2.1.4 Deep learning

Deep learning is a type of machine learning in which the term *deep* refers to a number of layers in the network. Deep learning can have hundreds of layers, while traditional neural networks contain only three layers: an input layer, one hidden layer, and an output layer [59]. The basic deep learning architecture can be represented as a deep neural network that combines multiple non-linear processing layers (see Figure 2.4). Deep learning architectures can consist of an input layer, multiple hidden layers, and an output layer. The layers are interconnected via nodes, or neurons, with each hidden layer using the output of the previous layer as its input. The model still focuses on learning a function f to map input X to output y. This learning process occurs through the back propagation technique

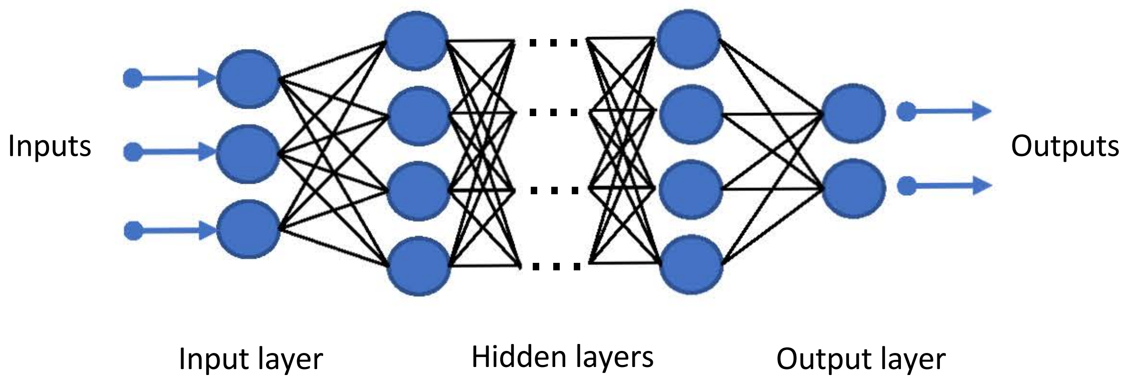


Figure 2.4: Deep neural network

where the parameters of the networks are adjusted to minimized loss on the training data [60].

Several deep learning architectures have been proposed to handle different problems. We can broadly categorize most of them into three major classes [61]:

1. **Deep networks for unsupervised learning** aim to capture correlation and pattern of data when no information about target class (i.e. label) is available. There are several deep learning architectures that fall into this class such as restricted Boltzmann machine (RBM), deep belief networks (DBN), deep Boltzmann machines (DBM), and recurrent neural networks (RNN). For example, DBN is the deep networks that stack multiple layers of RBM. Each RBM layer learns a feature representation processed from previous RBM layers. The output at the final layer is the feature representation of the data processed from multi RBM layers. In training, since it is unsupervised learning, the model aims to minimize loss on decoding the derived feature representation back to the original input. These deep networks are mostly adapted in speech recognition and natural language processing where deep and dynamic structure representation of data can be extracted [61].

2. **Deep networks for supervised learning** intend to provide high discriminative power for supervised learning problems (i.e. target labels are available). The convolutional neural network (CNN) and the supervised learning version of RNN are well-known supervised deep learning architectures. Note that the previous mechanism of RNN was classified to unsupervised learning, but it has limited success [53]. CNN consists of convolutional layers and a pooling layers that are stacked up to form a deep model. While the convolutional layer shares many parameters, the pooling layer subsamples the output of the convolutional layer and reduces the data rate from the previous convolutional layer (see Figure 2.5) [62]. CNN performs

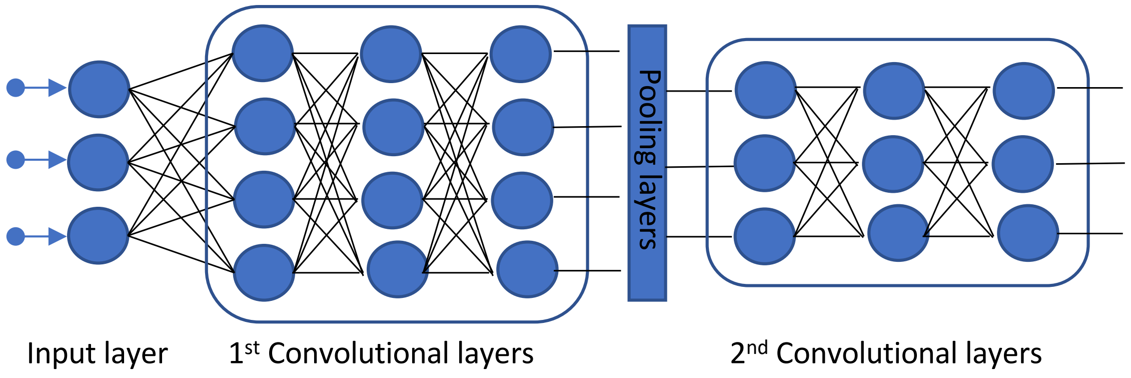


Figure 2.5: Convolutional Neural Network (CNN)

best and is commonly used in computer vision and image recognition [63]. RNN recently can be also used as a discriminative model (i.e. supervised learning) where the target output (i.e. label) associates to the input data sequence (e.g. predicting a next word in a sentence) [64]. RNN uses the internal memory cell (i.e. many trainable parameters) to process sequences of inputs. This technique allows RNN to learn unsegmented and connected data such as text, speech, handwriting. In our research, we have adapted the well-known variation of RNN called Long Short-Term Memory (LSTM) which we provide the discussion in the next section.

3. **Hybrid deep networks** combine (1) and (2) to perform a task. This deep learning architecture makes use of the two classes of deep networks: unsupervised learning and supervised learning. While, the former is excellent in capturing patterns and structures from data, the latter effectively performs a discrimination task. For example, the work in [65] improves the speech recognition and language translation by combining the two deep networks. The first unsupervised learning networks learn features from raw speech signal. The supervised learning networks then determine a word string that corresponds to the speech in different languages.

2.1.5 Long Short Term Memory

Long Short-Term Memory (LSTM) [66], [67] is a special variant of RNN [64]. While a feedforward neural network maps an input vector into an output vector, an LSTM network uses a loop in a network that allows information to persist and it can map a sequence into a sequence (see Figure 2.6).

Let $w_1, ..., w_n$ be the input sequence (e.g. words in a sentence) and $y_1, ..., y_n$ be the sequence of corresponding labels (e.g. the next words). At time step t, an LSTM unit

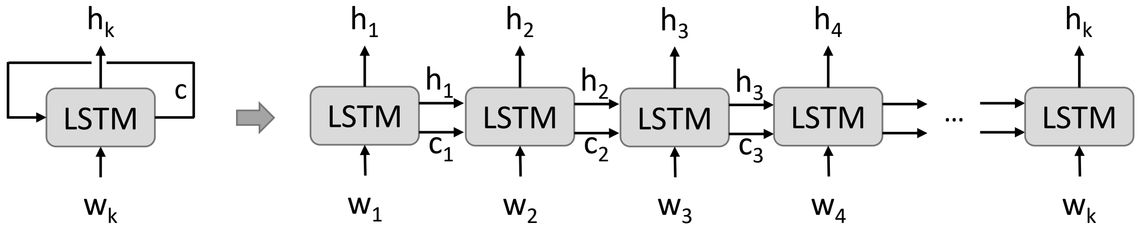


Figure 2.6: An LSTM network

reads the input w_t, the previous hidden state h_{t-1}, and the previous memory c_{t-1} in order to compute the hidden state h_t. The hidden state is used to produce an output at each step t. For example, the output of predicting the next word k in a sentence would be a vector of probabilities across our vocabulary, i.e. $softmax(V_k h_t)$ where V_k is a row in the output parameter matrix W_{out}.

The most important element of LSTM is a short-term *memory cell* – a vector that stores accumulated information over time (see Figure 2.7). The information stored in the memory is refreshed at each time step through partially forgetting old, irrelevant information and accepting fresh new input. An LSTM unit uses the forget gate f_t to control how much information from the memory of previous context (i.e. c_{t-1}) should be removed from the memory cell. The forget gate looks at the the previous output state h_{t-1} and the current word w_t, and outputs a number between 0 and 1. A value of 1 indicates that all the past memory is preserved, while a value of 0 means "completely forget everything". The next step is updating the memory with new information obtained from the current word w_t. The input gate i_t is used to control which new information will be stored in the memory. Information stored in the memory cell will be used to produce an output h_t. The output gate o_t looks at the current code token w_t and the previous hidden state h_{t-1}, and determines which parts of the memory should be output.

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

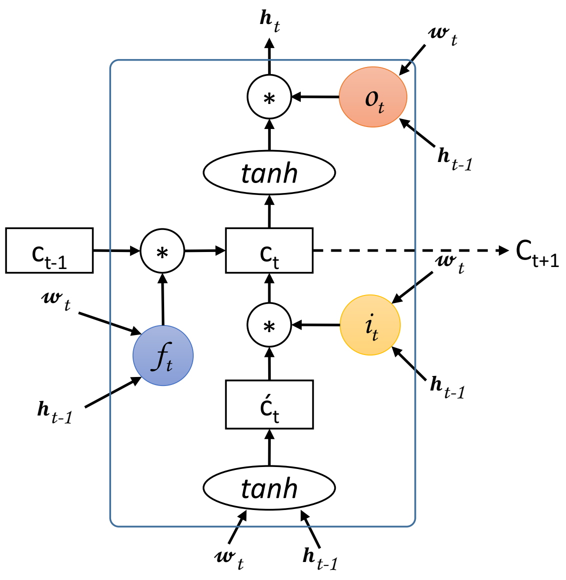


Figure 2.7: The internal structure of an LSTM unit

sequences. LSTM has demonstrated ground-breaking results in many applications such as language models [68], speech recognition [69] and video analysis [70].

2.1.6 Predictive performance measures

After training, the prediction model is evaluated on a test set. There are a number of performance measures (e.g. Precision, Recall, and Absolute Error) for assessing the predictive performance. In this section, we briefly discuss some common predictive performance measures used in our studies.

Classification metrics

A classification model predicts class labels for a given input data. The confusion matrix is commonly used to store the correct and incorrect decisions made by a classifier. For example, in our hiking problem, if an input is classified as "`yes`" when it was truly "`yes`", the classification is a true positive (tp). If the input is classified as "`yes`" when actually it was "`no`", then the classification is a false positive (fp). If the input is classified as "`no`" when it was in fact "`yes`", then the classification is a false negative (fn). Finally, if the input is classified as "`no`" and it was in fact "`no`", then the classification is true negative (tn). In binary classification, "`yes/no`" outputs are commonly referred as positive/negative samples (i.e. positive and negative classes).

The values stored in the confusion matrix are then used to compute the widely-used Precision, Recall, and F-measure to evaluate the performance of the predictive models:

- Precision (Prec): The ratio of correctly predicted samples over all the samples predicted as positive class. It is calculated as:

$$pr = \frac{tp}{tp + fp}$$

- Recall (Re): The ratio of correctly predicted positive samples over all of the actually positive samples. It is calculated as:

$$re = \frac{tp}{tp + fn}$$

- F-measure: Measures the weighted harmonic mean of the precision and recall. It is calculated as:

$$F - measure = \frac{2 * pr * re}{pr + re}$$

- Area Under the ROC Curve (AUC) is used to evaluate the degree of discrimination achieved by the model. The value of AUC is ranged from 0 to 1 and random prediction has AUC of 0.5. The advantage of AUC is that it is insensitive to decision threshold like precision and recall. The higher AUC indicates a better predictor.

Regression metrics

Since an output from a regression model is a continuous number, regression metrics aim to measure error between predicted target values and correct target values (i.e. ground truth). We briefly describe here some common performance measures for regression problems as follows:

- Absolute Error (AE): The difference between the predicted value ($\hat{y}$) and the correct value (y). It is calculated as:

$$AE = |\hat{y} - y|$$

- Mean Absolute Error (MAE): The average of all absolute errors. MAE can reflect the magnitude of the error. It is calculated as:

$$MAE = \frac{1}{N}\sum_{i=1}^{N} |\hat{y}_i - y_i|$$

- Root Mean Squared Error (RMSE): The standard deviation of the differences between predicted values and correct values. RMSE assumes that the error follows a normal distribution. It is calculated as:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N} (\hat{y}_i - y_i)^2}$$

- Mean of Magnitude of Relative Error (MRE): The average of magnitude of the relative error. It is calculated as:

$$MRE = \frac{1}{N}\sum_{i=1}^{N} |\hat{y}_i - y_i| / y_i$$

  where N is the number of samples in test set, $\hat{y}_i$ is the predicted value, and y_i is the correct value, for sample i.

2.2 Issue-driven software project management

In the previous section, we provided the background of machine learning, discussed the common machine learning algorithms that have been used our research, and provided the basic concepts and the well-known architectures of deep learning. In this section, we give an overview of issue-driven project management. It includes the basic information of an issue recorded in an issue tracking system in Section 2.2.1. In Section 2.2.2, we then discuss the issue's life cycle where it involves the process of issue resolution. We also briefly discuss how the issue tracking system supports the managing of an iteration in Section 2.2.3.

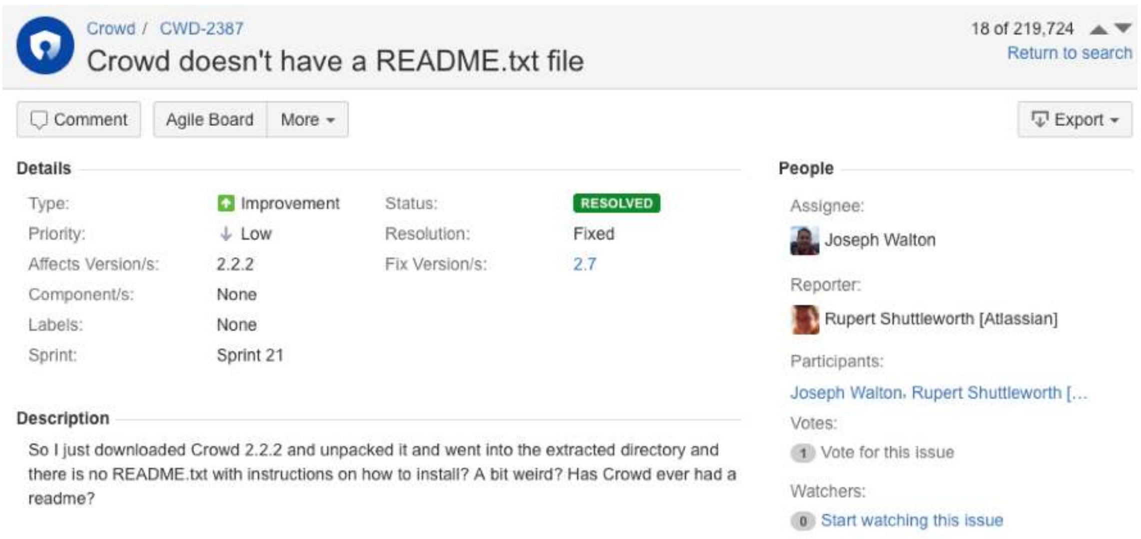
The issue-driven approach aims to promote good practices in project management. The common recommendations in project management guidance (e.g. [71]–[74]) mostly state that the progress of a project should be visible to all team members and they should know what tasks that they are working on and what to do next. Currently, software development teams usually use an issue tracking system (e.g. JIRA software[a]) that can provide a central place for team members to manage issues in which a task can be specified by an issue. For example, a task of implementing a new feature would involve with issues in the new feature request type. An issue tracking system records all actions performed on an issue (e.g. team's discussion is recorded in a form of comments). The team members can track and review their progress on a single issue. Hence, these mechanisms increase accountability of the projects as recommended in the guidance.

2.2.1 Characteristics of an issue

An *Issue* can be broadly referred to new feature request, bug report, development task, or enhancement [75] where a type of an issue can be specified as one of the issue's attributes. Issues can be created by stakeholders who involve in a project (e.g. users, developers, code reviewer, or tester). Currently, there are several software tools called *issue tracking system* that support software teams in managing issues (e.g. monitoring issue's status, assigning issues to a developer, and assigning issues to a planned release). For example, JIRA software and Bugzilla[b] are an issue tracking systems from Atlassian and Mozilla, respectively. These systems are applied in several applications, e.g., bugs/change requests tracking, help desk/customer service, project management, task tracking, requirement management, and workflow/process management. It can link issue reports with other tools that support different development activities such as linking issues with commits in GitHub[c] for source code version controlling, and grouping issues for a release with Bamboo[d] for integration and release management. Large open source projects (e.g. Apache, JBoss, and Moodle) have a large number of issues reported every day. For example, on average, around 70 issue were reports created every day in the Eclipse project [76]. In addition, a recent study [77] shows that an average of 168 new issues are reported per day in the Mozilla's projects.

Issues recorded in those systems mostly have the same primitive attributes to describe the characteristics of each issue. It has a unique identifier for issue referencing. Those importantly common issue's attributes are a project, type, summary and description, priority, status, assignee, fix/affect version, due date, and resolution. Note that we

[a]`https://www.atlassian.com/software/jira`
[b]`https://www.bugzilla.org/`
[c]`https://github.com/`
[d]`https://www.atlassian.com/software/bamboo`



Crowd / CWD-2387

Crowd doesn't have a README.txt file

18 of 219,724

Return to search

Comment | Agile Board | More | Export

Details

Type:	Improvement	Status:	RESOLVED
Priority:	Low	Resolution:	Fixed
Affects Version/s:	2.2.2	Fix Version/s:	2.7
Component/s:	None		
Labels:	None		
Sprint:	Sprint 21		

Description

So I just downloaded Crowd 2.2.2 and unpacked it and went into the extracted directory and there is no README.txt with instructions on how to install? A bit weird? Has Crowd ever had a readme?

People

Assignee: Joseph Walton

Reporter: Rupert Shuttleworth [Atlassian]

Participants: Joseph Walton, Rupert Shuttleworth [...

Votes: 1 Vote for this issue

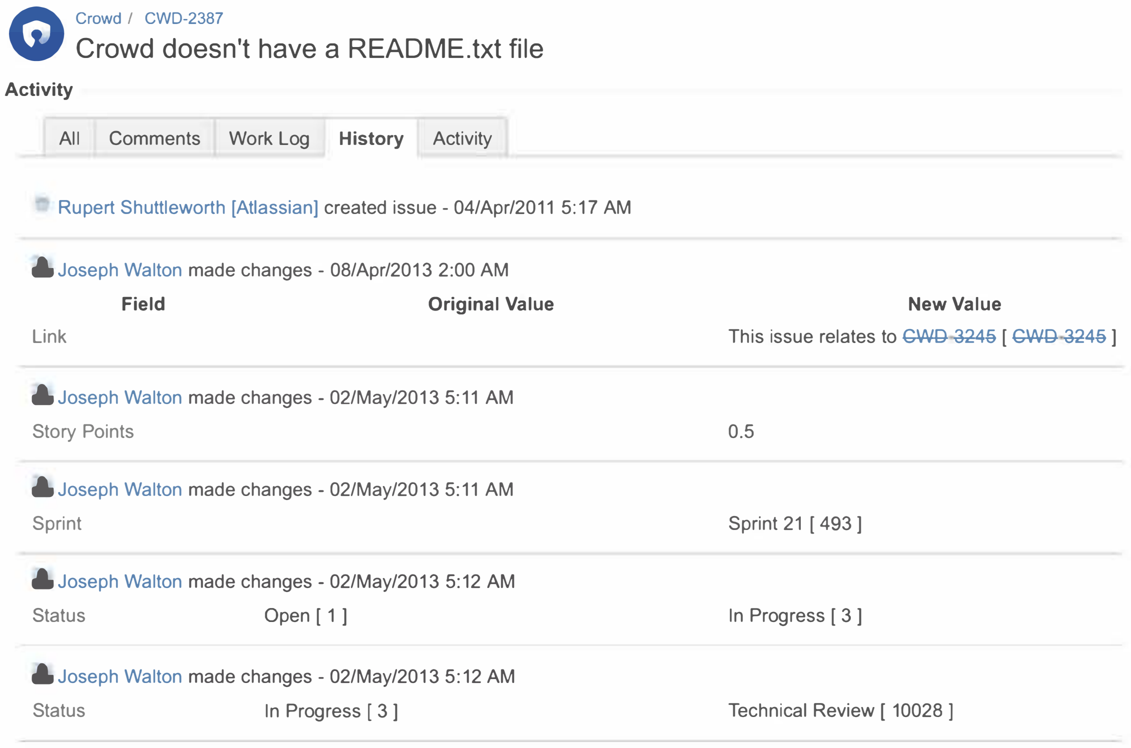
Watchers: 0 Start watching this issue

Figure 2.8: Example of an issue in JIRA software

focus on the characteristics of a JIRA issue since our studies perform on the issues from their platform.

The *issue's type* describes a type of an issue. For example, the *Bug* type describes that an issue is a problem that causes software malfunction or the *Epic* type is a large user story that can be broken down into a number of user stories which can involves in multiple iterations in the iterative software development (e.g. Agile) and versions. The *summary and description* is a brief (one-line) summary and a detailed description of an issue, respectively. The *priority* indicates the importance of an issue compared to other issues (e.g. highest, high, medium, low, or lowest). The *status* indicates the current state of an issue in issue's life cycle (see Section 2.2.2). The *assignee* is a person (e.g. developer) who an issue is currently assigned to. The *fix/affect versions* are two attributes indicated project versions in which an issue will be fixed and project versions in which an issue has been found. The *due date* is the date which an issue is planned to be resolved. The *resolution* is a record of an issue's resolution. An issue could have been completed in many ways, for example, *Done* means an issue has been resolved while *Duplicate* indicates that this is an issue duplicated issue and the work has been tracked in other issues. Figure 2.8 shows an example of issue ID *CWD-2387*[e] with its attributes recorded in the JIRA's issue tracking system. All changes occurred on issue's attributes are recorded in the issue change log. A team can trace back their actions and review the progress in resolving the issue. Figure 2.9 shows an example of issue's change log of issue ID *CWD-2387*.

[e]`https://jira.atlassian.com/browse/CWD-2387`


Crowd / CWD-2387

Crowd doesn't have a README.txt file

Activity

All | Comments | Work Log | History | Activity

Rupert Shuttleworth [Atlassian] created issue - 04/Apr/2011 5:17 AM

Joseph Walton made changes - 08/Apr/2013 2:00 AM

Field	Original Value	New Value
Link		This issue relates to ~~CWD-3245~~ [~~CWD-3245~~]

Joseph Walton made changes - 02/May/2013 5:11 AM

Story Points		0.5

Joseph Walton made changes - 02/May/2013 5:11 AM

Sprint		Sprint 21 [493]

Joseph Walton made changes - 02/May/2013 5:12 AM

Status	Open [1]	In Progress [3]

Joseph Walton made changes - 02/May/2013 5:12 AM

Status	In Progress [3]	Technical Review [10028]

Figure 2.9: Example of an issue's change log

2.2.2 Lifecycle of an issue

An issue lifecycle describes the process of issue resolution. When an issue has been created, it must be passed through the process of triaging corresponding to the defined issue lifecycle (e.g. workflow), for example, identifying the priority, identifying the related projects or components, and assigning an issue to developers [78], [79]. This workflow can be varied from projects to projects. Typically, workflows represent software development process within an organization and a team project. The issue lifecycle is a set of states and transitions (i.e. the *state* attribute) that an issue goes through during the issue resolution process. Different issue types can have different issue lifecycles.

For example, Figure 2.10 shows the issue lifecycle from the Moodle[f] project (E-learning platform). They collect the issues in their JIRA software repository[g]. There are ten possible states of issues (e.g. Open/Reopened, Development in progress, or Waiting for review) involved different roles in project teams (e.g. Developers, Integration reviewers, and Testers). When a new issue has been created, the developer team performs the issue triaging process that involves the new issue investigation to confirm whether the

[f]`https://moodle.com/`

[g]`https://tracker.moodle.org/`

issue report is correctly recorded. All the important issue's attributes are then identified. The issue's state is set to *Development in progress* when the assigned developer starts working on the issue. After the assigned developer finishes his/her work (e.g. coding), the issue is then passed to the integration reviewers for the code-level review. The state of the issue is then changed to *Waiting for review* and *Review in progress* when the reviewer starts the reviewing process. If the new code (e.g. bug fixing, new function) pass the review, the code is then integrated into their version control system (GitHub repository) for further testing. The issue's state is changed to *Testing in progress* when the tester starts the testing process. In testing, if the testers find problems, the integrated code must be removed from the repository, and the issue is pushed back to the developer for further work. The final state is *Closed* which represents that the issue has been resolved. How the issue is closed is then specified in the *Resolution* attribute (e.g. fixed, duplicated, and can not replicate).

2.2.3 Characteristics of an iteration

Incremental and iterative development are essential parts of many popular software development methodologies such as (Rational) Unified Process, Extreme Programming, Scrum and other agile software development methods [3] where software products are developed in sequences of fixed-length iterations. Typically, it is one or two-week length. This process helps teams respond to the unpredictable changes of developing software through incremental and iterative work. Figure 2.11 shows an overview of Scrum Agile methodology. Scrum is one of the Agile software development frameworks. In Scrum, an iteration is called *sprint*. In each sprint, a team works on the highest priority user stories selected from the product backlog which is a prioritized list of requirements.

Several issue tracking systems also support the iterative development (e.g., Scrum and Kanban). These tools allow teams to manage issues following the practices of Agile development. The prioritized issues can be considered as the product backlog. The team can then create sprints from the list of prioritized issues (e.g. backlog). Figure 2.12 shows an example of *Sprint 4*[h] in JIRA Scrum dashboard of the Source Tree project recorded in the JIRA software. The *sprint 4* consists of 8 issues which are separated into 3 states: *To Do*, *In Progress*, and *Done*, following the practices in the progress monitoring of Scrum. These three states correspond to the states of an issue in its life cycle. The issues in *To Do* are those that their state is *opened*. When the state of issues are changed to *Development in progress*, those issues are moved from *To Do* to *In Progress* in the

[h]`https://jira.atlassian.com/secure/RapidBoard.jspa?rapidView=1301`

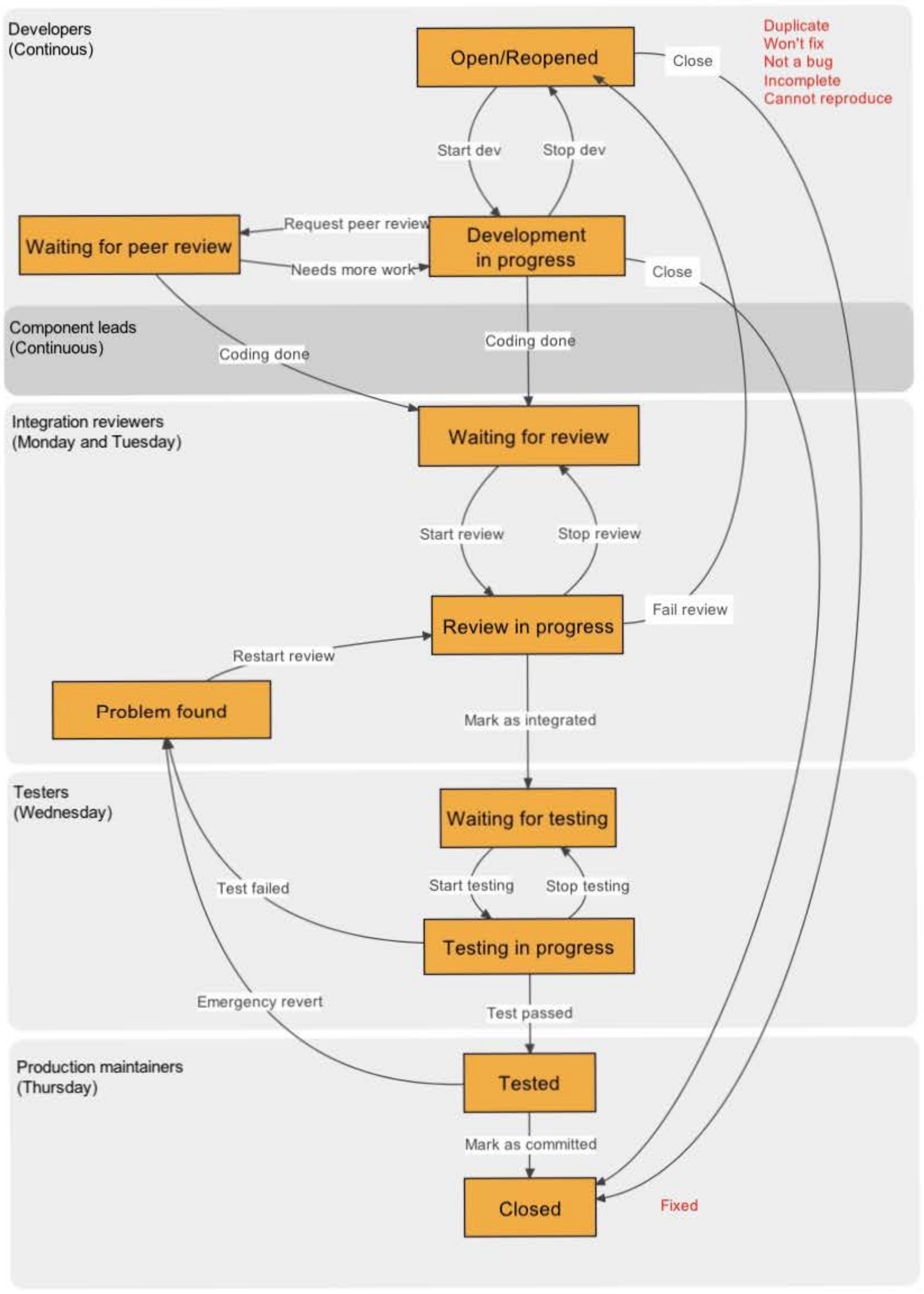


Figure 2.10: Example of an issue workflow. This figure is adopted from `https://docs.moodle.org/dev/Process`

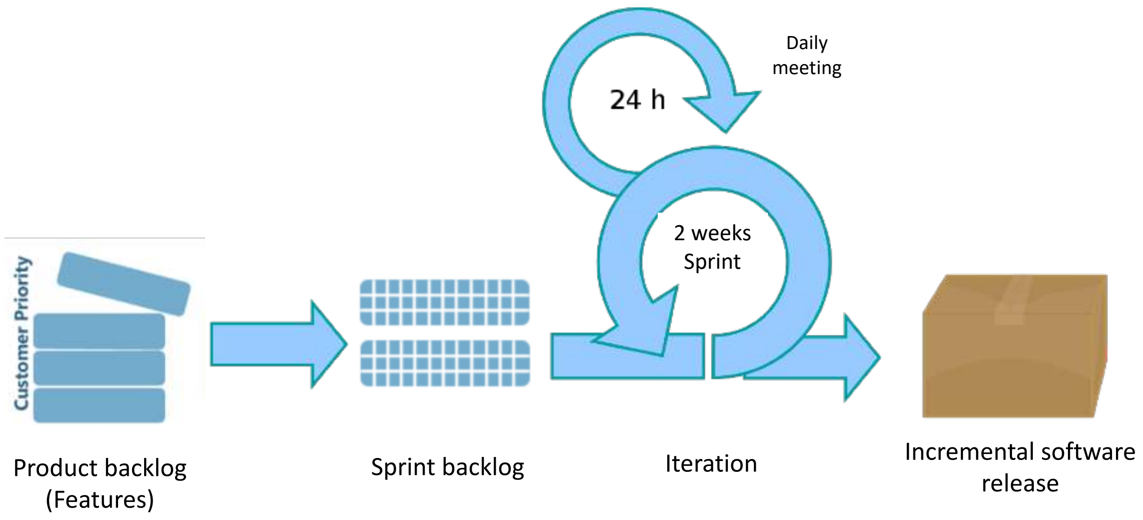
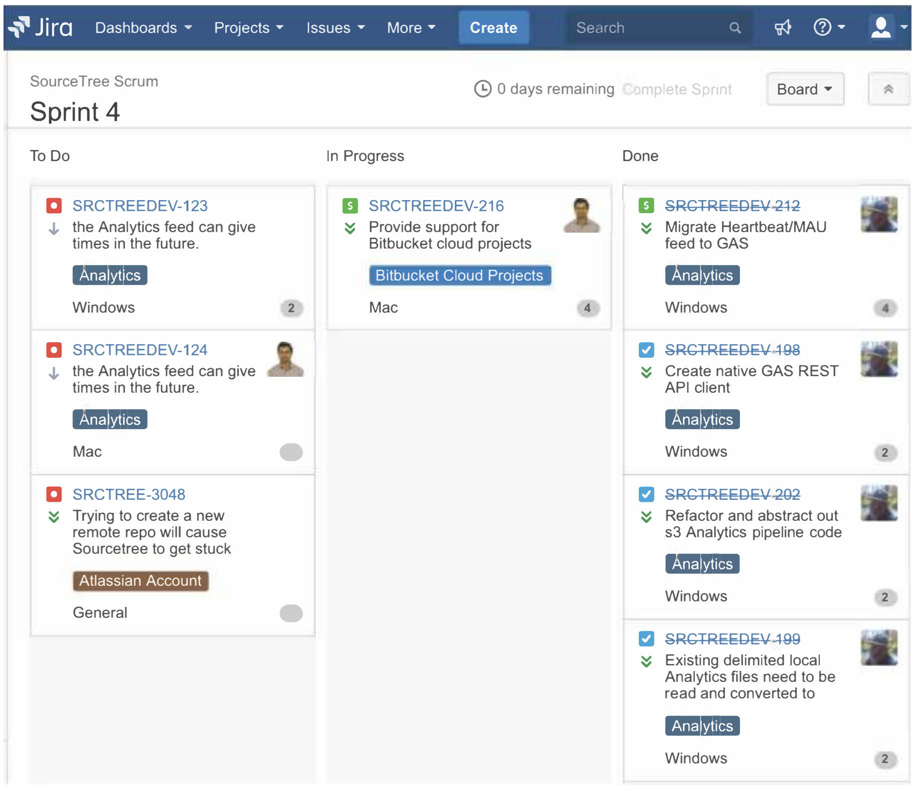


Figure 2.11: The Scrum Agile methodology



SourceTree Scrum

Sprint 4

0 days remaining Complete Sprint

To Do

SRCTREEDEV-123
the Analytics feed can give times in the future.
Analytics
Windows 2

SRCTREEDEV-124
the Analytics feed can give times in the future.
Analytics
Mac

SRCTREE-3048
Trying to create a new remote repo will cause Sourcetree to get stuck
Atlassian Account
General

In Progress

SRCTREEDEV-216
Provide support for Bitbucket cloud projects
Bitbucket Cloud Projects
Mac 4

Done

~~SRCTREEDEV-212~~
Migrate Heartbeat/MAU feed to GAS
Analytics
Windows 4

~~SRCTREEDEV-198~~
Create native GAS REST API client
Analytics
Windows 2

~~SRCTREEDEV-202~~
Refactor and abstract out s3 Analytics pipeline code
Analytics
Windows 2

~~SRCTREEDEV-199~~
Existing delimited local Analytics files need to be read and converted to
Analytics
Windows 2

Figure 2.12: Example of a sprint in JIRA software

dashboard. The *Done* state thus contains the issues having *closed* as issue's state. At the end of the sprint, the report recorded the completed, in-completed, and removed issues is generated. The Scrum team can thus keep track their productivity from past sprints to maintain the consistency of their delivery capability.

2.3 Software engineering analytics

In this section, we present a broad background on software engineering analytics and its applications related to several software engineering tasks. Note that for the work that is directly relevant to our research questions, we discuss them in the related work section in each chapter.

Software engineering has been known as the systematic methods of the activities for the development of software [29]. Typically, the activities involved in software development can be divided into several sub-disciplines, e.g., requirement elicitation, software design, software construction, and software maintenance. Software engineering also includes software project management which aims to assure that the development and maintenance of software are systematic, disciplined, and quantified. The technical-related activities (e.g. coding, testing) focus on developing quality software products, while the project management activities (e.g. planning, monitoring) focuses on controlling of a project direction [80].

Nowadays, software engineering becomes data-rich activities [81]. A huge amount and many types of software development data from real-world settings are publicly available. For example, more than 800,000 bug reports have been recorded in the Mozilla Firefox's bug repository, 324,000 projects have been hosted by Soruceforge.net, and more than 11.2 million projects have been hosted by GitHub [81]. The PROMISE repository [82] has provided the effort estimation data from over 100 projects. Those open source projects provide not only the technical-related data (e.g. source code and bug report), but also provide non technical-related data. For example, the Moodle[i] project has published their current software release plan and the reports of previous releases that include 25 major releases and over 50 minor releases.

The well-known definition of software engineering analytics was defined in 2012 by R. Buse and T. Zimmermann as "applying analytic approaches on software data for managers and software engineers with the aim of empowering software development in-

[i]`https://moodle.com/`

dividuals and teams to gain and share insight from their data to make better decisions ” [83]. This approach leverages historical data generated from software engineering activities by applying techniques in statistics, data mining, pattern recognition, and machine learning which also called *data-driven* software engineering [81], [84]–[88]. The goal is to support software practitioners (e.g. project managers, risk mitigation planners, team leaders, and developers) in extracting important, insightful, and actionable information for completing various tasks in software engineering [83].

Machine learning techniques have been adapted in many aspects (e.g. building a predictive model) to improve performance and increase the productivity of software practitioners. The applying of machine learning involves the formulation of a problem to make it conforms with machine learning algorithms [31]. For example, Darwish et al. [89] have formulated the problem of selecting requirement elicitation techniques into a supervised learning problem. They have employed Artificial Neural Network (aNN) to build a recommendation model that can recommend suitable elicitation techniques. The model learns from the past selections and a set of features representing past requirement elicitation scenarios (e.g. elicitor, number of users, project complexity). To give a recommendation, the model recommends a set of techniques from 14 elicitation techniques (e.g. interviews, questionnaires, observation) based on the current requirement elicitation scenario. Grieco et al. [90] have also formulated the security problem as the supervised learning problem. The model learns from past vulnerability (e.g. exploitable) and features extracted from source code. The developer can input their new source code to the model to predict the vulnerability.

2.3.1 Applications of software engineering analytics

Software engineering analytics covers a broad spectrum of software development process. In this section, we discuss the applications of this approach. We however note that this does not intend to be a comprehensive list of existing work in software engineering analytics, since it involves with different areas related to software development (e.g. mining software repositories, software process management, and software recommendation system). We thus emphasize on providing a wide range of its applications.

Predicting and Identifying Bugs

Identifying bugs from a million lines of source code remains a challenge for most software projects. Applying data analytics to predict and identify bugs can increase developer's productivity. These techniques help developers and testers focus on those risky code. Lam et al. [15] recently proposed an automated bug localization approach using the combination between deep learning and information retrieval (e.g. measuring textual similarity) techniques. Their model overcomes the problem lexical mismatch between the same texts in bug reports and technical terms in source code. Bug prediction or bug localization approaches mostly aim for leveraging data from bug tracking systems and source code version control systems (e.g. [13], [14], [17], [19], [91], [92]). Generally, those approaches consider the relationships between words appeared in bug reports and code tokens appeared in source code. Dam et al. [93] presented a deep tree-based model that can take the Abstract Syntax Tree (AST) representation of source code into account. By taking AST as an input, the model learns not only the code syntax but also the different levels of semantics in source code.

Estimating effort

Effort estimation is the process related to estimating the effort necessary to complete a software task or a whole software project (e.g. the total hours of a developer to complete a task) [94]. High accurate effort estimation is one of the key success factors for software project [24], [27]–[29], [72], [94]. Several data analytic techniques have been adopted to build effort estimation models, for example, Kanmani et al. [95] employed Neural Network to build an estimation model for Object-Oriented systems using Class point data, while Kanmani [96] employed Fuzzy logic on the same data. Panda et al. [97] also used Neural Network applied on Agile development data (e.g. total number of story points, velocity of projects) to predict effort at the project level. Sentas et al. [98], [99] proposed regression-based methods for the confidence interval estimation. Kocaguneli et al. [8] explored the benefits of ensemble effort estimation techniques applied on process data to predict overall project effort.

In the community of software effort estimation research, a number of publicly available datasets (e.g. China, Desharnais, Finnish, Maxwell, and Miyazaki datasets in the PROMISE repository [82]) have become valuable assets for many research projects in software effort estimation in the recent years. Those datasets are suitable for estimating effort at the project level (i.e. estimating effort for developing a complete software system). For example, the recent work from Sarro et al. [7] applied multi-objective evolutionary approaches to five real-world datasets from the PROMISE repository namely China, Desharnais, Finnish, Miyazaki, and Maxwell involving 724 different software projects in total.

Managing risks

Software risk management consists two main tasks: risk assessment and risk control [100]. Risk assessment focuses on identifying and prioritizing risks, while risk control involves risk resolution planning and monitoring. The central of risk management is a capability to forecast those uncertainty situations whether it will take place [100]. Numerous data analytics approaches have been proposed to promote risk management tasks. Pika et al. [101] presented the event log data analysis approach for process delay prediction. They apply statistical techniques (e.g. identifying outliers) on the historical log to make the prediction. Neumann [56] proposed a risk analysis model using Neural Network. The model aims to support the risk prioritization (e.g. high impact or low impact) by analyzing the correlation between software metrics (e.g. McCabe's measurement) and source code (e.g. number of characters, number of comments, and lines of code). Xu et al. [10] applied genetic algorithm and decision tree to identify the best set of features for identifying risks of software projects. Chang [102] mined software repositories to acquire risks form past software projects. They collected the data at each milestone review, for example, review date, number of uncompleted tasks, number of requirement changes, and identified risks. They then employed association rule mining techniques to identify the processes that affect cost and schedule of projects.

Supporting bug triaging

In bug triaging process, making a correct assignment of all bug's attributes is non-trivial (e.g. components, priority, and severity). To support the process of bug triaging, a number of recommendation systems using data analytics has been proposed. For example, Kochhar et al. [103] proposed an automated bug classification techniques based on a similarity between bug's attributes. The work in [104]–[106] also proposes the techniques to classify bugs into software components based on the information from bug's descriptions and its related source code. In addition, textual description of the bugs has been leveraged for severity/priority predicting (e.g. [22], [23], [79], [107], [108]). Assigning a bug to a suitable developer who can fix it is also important in bug triaging. Xuan et al. [109], Robbes et al. [110], Anvik et al. [78], and Xia et al. [92] leveraged the historical data in bug tracking systems, especially, the past developer assignments to recommend a developer who has a potential to fix a new bug. A fixed bug can be reopened if it does not fixed properly. Providing sufficient and correct information can prevent the bug from reopening [111]. However, there are many reasons that cause reopened bugs (e.g. a bug fixed in the previous version can re-appears in the current version). The work in [111]–[113] leverages the data from bug reports (e.g. textual description and comment) to predict whether a bug will be reopened.

Program comprehension

Program comprehension focuses on studying the process involving in software maintenance. It aims to support software practitioners (e.g. developer) in understanding of existing software systems [114]. Software engineering analytics has been used to support team members in understanding software systems. Software systems are usually described by a number of documents (e.g. software design document). However, the documents for large software systems are often out of date and do not conform to the actual implemented systems [115]. It is a challenge for a new member to understand large software systems.

A recent study [116] presents an approach to leverage historical data in project repositories such as emails and bug reports to help in understanding software systems. The approach aims to identify relationships among data recorded in different repositories. For example, a developer can see the related discussions on the code being viewed in the code editor. Hassan et al. [117] mined the historical changes recorded in source control systems to build the static dependency graph of a software system. The constructed dependency graph can assist developers in understanding the architecture of large software systems.

Understanding development teams

Large software development teams usually communicate through different channels e.g. e-mails, social networks, and instant messaging. The discussions recoded in those systems are valuable data since those cover many important informations (e.g. code review, project plan, and project policy) [115]. To make use of those data, Bird et al. [118] proposed a statistical analysis approach to improve the communication and coordination among team members. Their approach mines the correlation among team members from the email data to construct social networks representing team's communication patterns. Rigby et al. [119] applied the sentiment analysis technique on the mailing list discussions to study the attitude changing of a development team before and after software release.

Other applications

There are many threads of research applying data analytics to support different aspects involved in software projects. For example, in the requirement gathering, Harman et al. [120] and Mojica et al. [121] proposes techniques to extract new requirements from user's reviews recorded the App store. In software production line management, Sayyad et al. [122] and Palma et al. [123] make use of the software configuration data from past software versions to recommend a suitable configuration for the next version. Hindle [124] analyzes operating system logs to predict power consumption of a software.

2.4 Chapter summary

In this chapter, we have given an overview of machine learning by briefly describing its concepts, discussing the difference between supervised and unsupervised learning and the difference between classification and regression. We have also explained the well-known machine learning algorithms that have been used in our research: Decision Tree (C4.5), Random Forests (RF), Gradient Boosting Machines (GBMs), Support Vector Machines (SVM), Naive Bayes (NB), NBTree, Artificial Neural Networks (aNN), Deep Neural Networks with Dropouts (Deep Nets). We have also given a background of deep learning including the description of three architecture types: deep networks for unsupervised and supervised learning and hybrid deep networks. We have then focused on Long Short-Term Memory (LSTM). The second part of this chapter provided a brief description of issue-driven project management which includes the description of an issue, its lifecycle, and an iteration. The final part of the chapter started with the introduction of software engineering analytics. We have also discussed how machine learning techniques have been adopted in software analytics. Finally, we have discussed the number of applications of software analytics in software engineering. In the next chapter, we begin describing our work starting with the delivery capability prediction.

Chapter 3

Delivery capability prediction

MODERN software development is mostly based on an incremental and iterative approach in which software is developed through repeated cycles (iterative) and in smaller parts at a time (incremental), allowing software developers to benefit from what was learned during development of earlier portions or versions of the software. Incremental and iterative development are essential parts of many popular software development methodologies such as (Rational) Unified Process, Extreme Programming, Scrum and other agile software development methods [3]. This is achieved by moving from a model where all software packages are delivered together (in a single delivery) to a model involving a series of incremental deliveries, and working in small iterations. Uncertainties, however, exist in software projects due to their inherent dynamic nature (e.g. constant changes to software requirements) regardless of which development process is employed. Therefore, an effective planning, progress monitoring, and predicting are still critical for iterative software development, especially given the need for rapid delivery [125]. Our focus here is on predicting the delivery capability of a *single iteration at a time*, rather than the whole software lifecyle as in traditional waterfall-like software development processes.

Existing work on building prediction models to support software development mostly focuses on the prediction at a whole software project, not a single iteration at a time (e.g. [5]–[8]), while providing with insightful and actionable information at the level of iterations would be valuable for project managers and decision makers. Our work in this chapter aims to fill this gap. We focus on predicting delivery capability as to whether the target amount of work will be delivered at the end of an iteration. Our proposal, with its ability to learn from prior iterations to predict the performance of future iterations, represents an important advance in our ability to effectively use the incremental model of software development. To do so, we have developed a dataset of 3,834 iterations in five large open source projects, namely Apache, JBoss, JIRA, MongoDB, and Spring. Each iteration requires the completion of a number of work items (commonly referred to as *issues*), thus 56,687 issues were collected from those iterations.

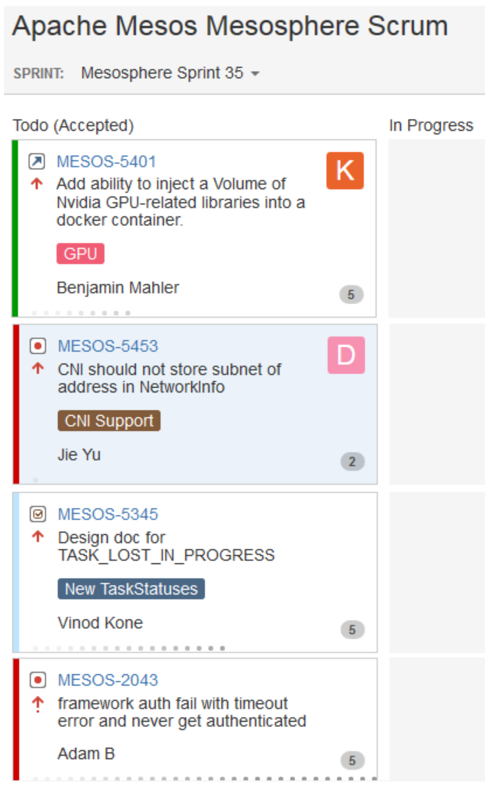
To build a feature vector representing an iteration, we extracted fifteen attributes associated with an iteration (e.g. its duration, the number of participants, etc.). In addition, an iteration is also characterized by its issues, thus we also extracted twelve attributes associated with an issue (e.g. type, priority, number of comments, etc.) and a graph describing the dependency between these issues (e.g. one issue blocks another issue). The complexity descriptors of the dependency graph (e.g. number of nodes, edges, fan in and fan out) form a group of features for an iteration. The attributes of the set of issues in an iteration are also combined, using either statistic aggregation or bag-of-words method, to form another group of features for an iteration. Statistical aggregation looks for simple set statistics for each issue attribute such as max, mean or standard deviation (e.g. the maximum number of comments in all issues in a iteration). On the other hand, the bag-of-words technique clusters all the issues in a given project, then finds the closest prototype (known as “word”) for each issue in an iteration to build a feature vector. The bag-of-words method therefore obviates the need for manual feature engineering. Our novel approach of employing multiple layers of features and automatic feature learning is similar to the notion of increasingly popular deep learning methods.

Those features allow us to develop accurate models that can predict delivery capability of an iteration (i.e. how much work was actually completed in an iteration against the target). Our predictive models are built based on three different state-of-the-art *randomized ensemble methods*: Random Forests, Stochastic Gradient Boosting Machines, and Deep Neural Networks with Dropouts. An extensive evaluation was performed across five projects (with over three-thousand iterations), and our evaluation results demonstrate that our approach outperforms three common baselines and performs well across all case studies.

The remainder of this chapter is organized as follows. Section 3.1 presents an overview of our approach. Section 3.2 presents a comprehensive set of features and describes how these features are aggregated. Section 3.3 presents the predictive models we have developed. We explain how we collect the data for our empirical study in Section 3.4. Section 3.5 reports on the experimental evaluation of our approach. Related work is discussed in Section 3.6 before we summarize the chapter in Section 3.7.

3.1 Approach

In iterative, agile software development, a project has a number of *iterations* (which are referred to as *sprints* in Scrum [126]). An iteration is usually a short (usually 2–4 weeks) period in which the development team designs, implements, tests and delivers a distinct *product increment*, e.g. a working milestone version or a working release. Each iteration requires the resolution/completion of a number of *issues*. For example, iteration *Mesos-sphere Sprint 35*[a] in the Apache project (see Figure 3.1) requires the completion of four issues: *MESOS-5401*, *MESOS-5453*, *MESOS-5445*, and *MESOS-2043*. At the beginning of the iteration (i.e. May 14, 2016), all of these issues were placed in the Todo list (or also referred to as the iteration or sprint backlog). The iteration was scheduled to finish on May 30, 2016.


Apache Mesos Mesosphere Scrum

SPRINT: Mesosphere Sprint 35

Todo (Accepted) | In Progress

MESOS-5401
Add ability to inject a Volume of Nvidia GPU-related libraries into a docker container.
GPU
Benjamin Mahler
K
5

MESOS-5453
CNI should not store subnet of address in NetworkInfo
CNI Support
Jie Yu
D
2

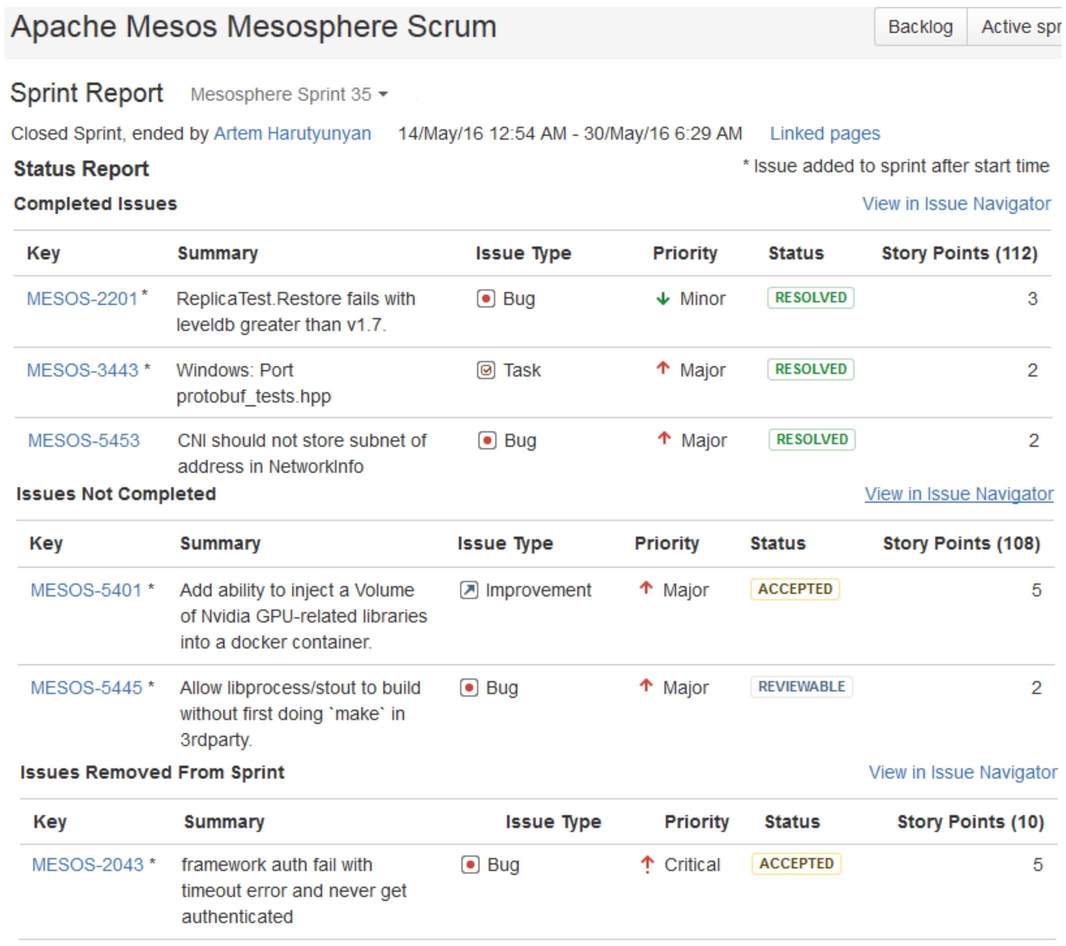
MESOS-5345
Design doc for TASK_LOST_IN_PROGRESS
New TaskStatuses
Vinod Kone
5

MESOS-2043
framework auth fail with timeout error and never get authenticated
Adam B
5

Figure 3.1: An example of an iteration (at the beginning)

Planning is done before an iteration starts and focuses on determining its starting time, completion time and the issues to be resolved in this iteration. Agile approaches recommend that an iteration be *time-boxed* (i.e. have a fixed duration) [125]. During an iteration, issues can be added and removed from the iteration. At the end of an iteration, a

[a]`https://issues.apache.org/jira/secure/RapidBoard.jspa?rapidView=62&view=reporting&chart=sprintRetrospective&sprint=236`



Apache Mesos Mesosphere Scrum

Sprint Report Mesosphere Sprint 35

Closed Sprint, ended by Artem Harutyunyan 14/May/16 12:54 AM - 30/May/16 6:29 AM Linked pages

Status Report * Issue added to sprint after start time

Completed Issues View in Issue Navigator

Key	Summary	Issue Type	Priority	Status	Story Points (112)
MESOS-2201 *	ReplicaTest.Restore fails with leveldb greater than v1.7.	Bug	Minor	RESOLVED	3
MESOS-3443 *	Windows: Port protobuf_tests.hpp	Task	Major	RESOLVED	2
MESOS-5453	CNI should not store subnet of address in NetworkInfo	Bug	Major	RESOLVED	2

Issues Not Completed View in Issue Navigator

Key	Summary	Issue Type	Priority	Status	Story Points (108)
MESOS-5401 *	Add ability to inject a Volume of Nvidia GPU-related libraries into a docker container.	Improvement	Major	ACCEPTED	5
MESOS-5445 *	Allow libprocess/stout to build without first doing `make` in 3rdparty.	Bug	Major	REVIEWABLE	2

Issues Removed From Sprint View in Issue Navigator

Key	Summary	Issue Type	Priority	Status	Story Points (10)
MESOS-2043 *	framework auth fail with timeout error and never get authenticated	Bug	Critical	ACCEPTED	5

Figure 3.2: An example of a closed iteration report

number of issues are completed and there may also be a number of issues assigned to the iteration that remain incomplete/unresolved. These incomplete/unresolved issues may be assigned to future iterations.

Let t_{pred} refer to the time at which a prediction is being made (e.g. the third day of a 17-day iteration). Given time t_{pred} during an iteration, **we would like to predict the amount of work delivered at the end of an iteration (i.e. the number of issues resolved), relative to the amount of work which the team has originally committed to.** More specifically, let `Committed` be the set of issues that the team commits to achieve in the current iteration before time t_{pred}. Let `Delivered` be the set of issues actually delivered at the end of the iteration, and `NonDelivered` be the set of issues that the team committed to delivering but failed to deliver in this iteration. Note that `NonDelivered` includes issues from the `Committed` set but are removed from the iteration after prediction time t_{pred} and/or issues that are not completed when the iteration ends.

We illustrate and motivate this using the example in Figure 3.1. At the beginning, a team planned to deliver 4 issues which are *MESOS-5401*, *MESOS-5453*, *MESOS-5445*, and *MESOS-2043* from iteration *Mesossphere Sprint 35* which is a 17–day iteration. Assume that, on the second day, the team added 2 new issues to the iteration: *MESOS-2201* and *MESOS-3443*. Let assume that three days after the iteration started we would like to make a prediction; i.e. t_{pred} at day 3. The number of issues in `Committed` is then 6 issues (*MESOS-5401*, *MESOS-5453*, *MESOS-5445*, *MESOS-2043*, *MESOS-2201*, and *MESOS-3443*) at the time t_{pred} when the prediction is made. After the third day, issue *MESOS-2043* has been removed from the iteration, and at the end of the iteration, a team completed only 3 issues which are *MESOS-2201*, *MESOS-3443*, and *MESOS-5453*, while the remaining issues (*MESOS-5401* and *MESOS-5445*) were not resolved. Thus, the issues in `Delivered` are *MESOS-2201*, *MESOS-3443*, and *MESOS-5453* and the issues in `NonDelivered` are *MESOS-5401* and *MESOS-5445* which were not completed and *MESOS-2043* which was removed from the iteration (see Figure 3.2). We note that an issue's status (e.g. resolved or accepted) also corresponds to the associated iteration's report, e.g. *MESOS-2201* was resolved and was placed in the completed list while issue *MESOS-5445* was in the reviewing process.

Predicting which issues would belong to the `Delivered` sets is difficult, and in some cases is impossible, e.g. some new issues in the `Delivered` set could be added after prediction time t_{pred}. **Hence, we propose to quantify the amount of work done in an iteration and use this as the basis for our prediction.** This approach reflects common practice in agile software development. For example, several agile methods (e.g. Scrum) suggest the use of *story points* to represent the effort, complexity, uncertainty, and risks involving resolving an issue [125]. Specifically, a story point is assigned to each issue in an iteration. The amount of work done in an iteration is then represented as a *velocity*, which is the total story points completed in the iteration. Velocity reflects how much work a team gets done in an iteration.

Definition 1 (Velocity) *Velocity of a set of issues I is the sum story points of all the issues in I:*

$$velocity(I) = \sum_{i \in I} sp(i)$$

where $sp(i)$ is the story point assigned to issue i.

For example, as can be seen from Figure 3.2, there are six issues that were committed to be delivered from iteration *Mesossphere Sprint 35* in the Apache project at the prediction time t_{pred} (e.g. issue *MESOS-2201* has 3 story points). Thus, the committed velocity is 19, i.e. $velocity(\texttt{Committed}) = 19$. However, there are only three issues had been resolved in iteration *Mesossphere Sprint 35*. Thus, the velocity delivered from this iteration is 7, i.e. $velocity(\texttt{Delivered}) = 7$ (two issues have the story points of 2 and one of them has the story point of 3).

We thus would like to predict the delivery capability in an iteration. Our approach is able to predict, given the current state of the project at time t_{pred}, what is the difference between the actual delivered velocity against the committed (target) velocity, defined as $velocity(\texttt{Difference})$:

$$velocity(\texttt{Difference}) = velocity(\texttt{Delivered}) - velocity(\texttt{Committed})$$

For example, the difference between the actual delivered velocity against the committed velocity of iteration *Mesossphere Sprint 35* was -12, i.e. $velocity(\texttt{Difference}) = -12$, because $velocity(\texttt{Delivered})$ was 7 and $velocity(\texttt{Committed})$ at t_{pred} was 19. This iteration delivered below the target, i.e. $velocity(\texttt{Committed}) > velocity\ (\texttt{Delivered})$. Note that $velocity(\texttt{Difference}) = 0$ does not necessarily imply that an iteration has delivered on all its commitments (in terms of the specific issues that were to be resolved) but instead it assesses the quantum of work performed.

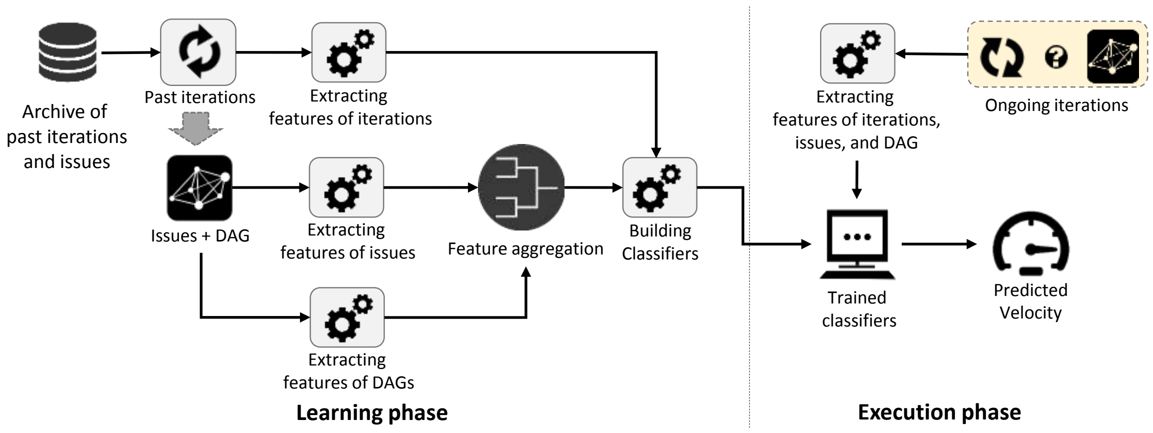

Archive of past iterations and issues
Past iterations
Extracting features of iterations
Issues + DAG
Extracting features of issues
Feature aggregation
Building Classifiers
Extracting features of DAGs
Learning phase
Extracting features of iterations, issues, and DAG
Ongoing iterations
Trained classifiers
Predicted Velocity
Execution phase

Figure 3.3: An overview of our approach

Our approach consists of two phases: the *learning phase* and the *execution phase* (see Figure 3.3). The learning phase involves using historical iterations to build a predictive model (using machine learning techniques), which is then used to predict outcomes, i.e. $velocity(\texttt{Difference})$, of new and ongoing iterations in the execution phase. To apply machine learning techniques, we need to engineer features for the iteration. An iteration has a number of attributes (e.g. its duration, the participants, etc.) and a set of issues whose dependencies are described as a dependency graph. Each issue has its own attributes and derived features (e.g. from its textual description). Our approach separates the iteration-level features into three components: (i) iteration attributes, (ii) complexity descriptors of the dependency graph (e.g. the number of nodes, edges, fan-in, fan-out, etc.), and (iii) aggregated features from the set of issues that belong to the iteration. A more sophisticated approach would involve embedding all available information into an Euclidean space, but we leave this for future work.

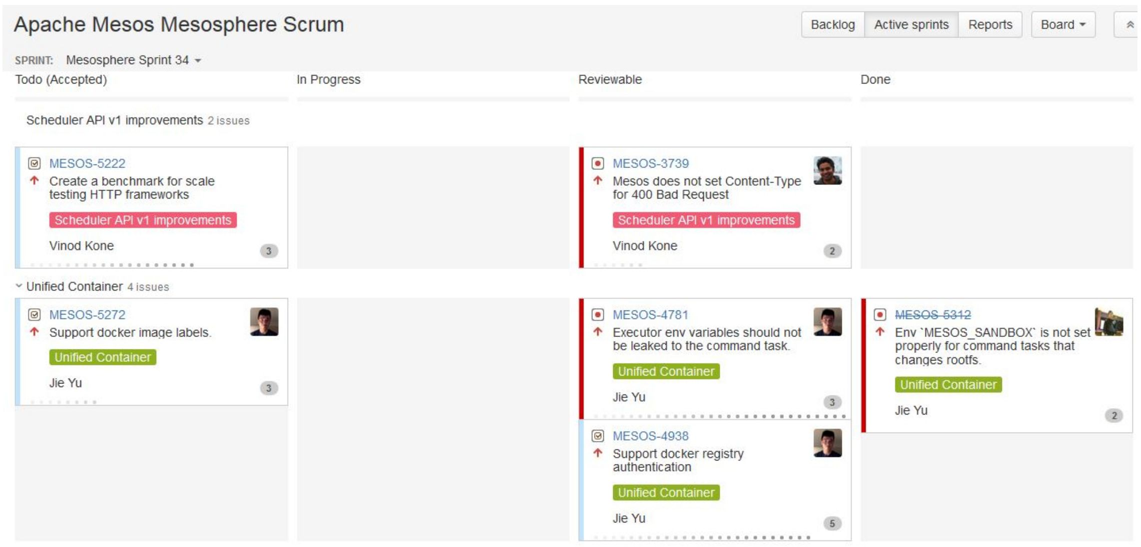
Formally, issue-level features are vectors located in the same Euclidean space (i.e. the issue space). The aggregation is then a map of a set points in the issue space onto a point in the iteration space. The main challenge here is to handle sets which are unordered and variable in size (e.g. the number of issues is different from iteration to iteration). We propose two methods: statistical aggregation and bag-of-words (BoW). Statistical aggregation looks for simple set statistics for each dimension of the points in the set, such as maximum, mean or standard deviation. For example, the minimum, maximum, mean, and standard deviation of the number of comments of all issues in an iteration are part of the new features derived for the iteration. This statistical aggregation technique relies on manual feature engineering. On the other hand, the bag-of-words method automatically clusters all the points in the issue-space and finds the closest prototype (known as "word") for each new point to form a new set of features (known as bag-of-words, similarly to a typical representation of a document) representing an iteration. This technique provides a powerful, automatic way of learning features for an iteration from the set of issues in the layer below it (similar to the notions of deep learning).

For prediction models, we employ three state-of-the-art *randomized ensemble methods*: Random Forests, Stochastic Gradient Boosting Machines, and Deep Neural Networks (DNNs) with Dropouts to build the predictive models. Our approach is able to make a prediction regarding the delivery capability in an iteration (i.e. the difference between the actual delivered velocity against the committed/target velocity). Next we describe our approach in more detail.

3.2 Feature extraction and aggregation

In this section, we describe our feature extraction from iteration and issue reports. Since an iteration has a set of issues to be resolved, we extract not only features of an iteration, but also features of an issue and the issue graph (i.e. dependency of issues) in an iteration. Most modern issue tracking systems (e.g. JIRA-Agile[b]) support an iterative, agile development which enables teams to use agile practices for their development to plan,

[b]https://www.atlassian.com/software/jira/agile



Apache Mesos Mesosphere Scrum

SPRINT: Mesosphere Sprint 34

Todo (Accepted) | In Progress | Reviewable | Done

Scheduler API v1 improvements 2 issues

MESOS-5222 Create a benchmark for scale testing HTTP frameworks — Scheduler API v1 improvements — Vinod Kone — 3

MESOS-3739 Mesos does not set Content-Type for 400 Bad Request — Scheduler API v1 improvements — Vinod Kone — 2

Unified Container 4 issues

MESOS-5272 Support docker image labels. — Unified Container — Jie Yu — 3

MESOS-4781 Executor env variables should not be leaked to the command task. — Unified Container — Jie Yu — 3

MESOS-4938 Support docker registry authentication — Unified Container — Jie Yu — 5

~~MESOS-5312~~ Env `MESOS_SANDBOX` is not set properly for command tasks that changes rootfs. — Unified Container — Jie Yu — 2

Figure 3.4: An example of an on-going iteration report

collaborate, monitor and organize iterations and issues. Dependencies between issues are also explicitly recorded (i.e. issue links) in the issues reports which both iterative and issue reports can be easily extracted from there. We then employ a number of distinct techniques (feature aggregation using statistics, Bag-of-Words, and graph measures) to characterize an iteration using both features of an iteration and features of a number of issues associated to an iteration. These details will be discussed in this section.

3.2.1 Features of an iteration

Table 3.1 summarizes a range of features that are used to characterize an iteration in our study. The features cover three important areas of an iteration: the elapsed time (e.g. the planned duration), the amount of work, and the team. Prediction time (i.e. t_{pred}) is used as a reference point to compute a number of features reflecting the amount of work. These include the set of issues assigned to an iteration when it begins (i.e. start time), and the set of issues added or removed from the iteration between the start time and prediction time. In addition, we also leverage a number of features reflecting the current progress of the team up until prediction time: the set of issues which have been completed, the set of work-in-progress issues (i.e. issues that have been acted upon but not yet completed), and the set of issues on which the team has not started working yet (i.e. to-do issues). Agile practices also suggest using this approach for monitoring the progress of an iteration [126].

Table 3.1: Features of an iteration

Feature	Description
Iteration duration	The number of days from the start date to planned completion date
No. of issues at start time	The number of issues assigned to an iteration at the beginning
Velocity at start time	The sum of story points of issues assigned to an iteration at the beginning
No. of issues added	The number of issues added during an iteration (between start time and prediction time)
Added velocity	The sum of story points of issues added during an iteration (between start time and prediction time)
No. of issues removed	The number of issues removed during an iteration (between start time and prediction time)
Removed velocity	The sum of story points of issues removed during an iteration (between start time and prediction time)
No. of to-do issues	The number of to-do issues in an iteration by prediction time
To-do velocity	The sum of story points of to-do issues by prediction time
No. of in-progress issues	The number of in-progress issues in an iteration by prediction time
In-progress velocity	The sum of story points of in-progress issues by prediction time
No. of done issues	The number of done issues in an iteration by prediction time
Done velocity	The sum of story points of done issues by prediction time
Scrum master	The number of Scrum masters
Scrum team members	The number of team members working on an iteration

Definition 2 (Prediction time) *A prediction time refers to the time that a prediction is made. We use it as a reference point when extracting the values of the features. A prediction time can be any time from the start time to finish time of data points (i.e. iteration or issue). The value of the features of each iteration and each issue in a dataset must be extracted at the same prediction time.*

Figure 3.4 shows an example of an on-going iteration report (recorded in JIRA-Agile) of *Mesosphere Sprint 34*[c] in the Apache project. This iteration started from April 27, 2016 to May 11, 2016. This iteration has two issues in the Todo state (*MESOS-5272* and *MESOS-5222*), three issues in the In-progress state – those are all in the reviewing process (*MESOS-3739*, *MESOS-4781*, and *MESOS-4938*), and one issue has been resolved (*MESOS-5312*). These issues have story points assigned to them. For each of those sets of issues, we compute the set cardinality and velocity, and use each of them

[c]`https://issues.apache.org/jira/secure/RapidBoard.jspa?rapidView=62`

as a feature. From our investigation, among the under-achieved iterations across all case studies, i.e. $velocity(\texttt{Difference}) < 0$, 30% of them have new issues added after passing 80% of their planned duration (e.g. after 8^{th} day of a ten-day iterations), while those iterations deliver zero-issue. Specifically, teams added more $velocity(\texttt{Committed})$ while $velocity(\texttt{Delivered})$ was still zero. This reflects that adding and removing issues affects the deliverable capability of an on-going iteration. This can be a good indicator to determine the outcome of an iteration.

These features were extracted by examining the list of complete/incomplete issues of an iteration and the change log of an issue, e.g. which iteration an issue was added or removed on which date, and its status (e.g. an issue is in the set of work-in-progress issues or in the set of to-do issues) at prediction time. Figure 3.5 shows an example of an iteration report for the iteration named *Twitter Aurora Q2' 15 Sprint 3* in the Apache project. The report provides a list of completed issues (e.g. *AURORA-274*) and uncompleted issues (e.g. *AURORA-698*), a list of added and removed issues during an iteration (e.g. *AURORA-1267*), and iteration details (e.g. state, start date, and planned end date). We can also identify when those issues were added or removed from the iteration by examining their change logs. Figure 3.6 shows an example of a change log of issue *AURORA-1267* which records that this issue has been added to *Twitter Aurora Q2' 15 Sprint 3* on May 16, 2015 while this iteration was started one day earlier (i.e. May 15, 2015).

There are a number of features reflecting the team involved in an iteration. These include the number of team leads (e.g. Scrum masters) and the size of the team. Note that the team structure information is not explicitly recorded in most issue tracking systems. We thus conjecture that the number of team members is the number of developers assigned to issues in an iteration. The number of Scrum masters is the number of authorized developers who can manage issues (e.g. add, remove) in an iteration. Future work could look at other characteristics of a team including the expertise and reputation of each team member and the team structure.

3.2.2 Features of an issue

The issues assigned to an iteration also play an important part in characterizing the iteration. Figure 3.7 shows an example of an issue report of issue *AURORA-716* in the Apache project which the details of an issue are provided such as type, priority, description, and comments including a story points and an assigned iteration. Hence, we also extract a

```
{"contents":{
 "completedIssues":[
   {"id": 12702203,
    "key": "AURORA-274"},
   {"id": 12724064,
    "key": "AURORA-556"},
   {"id": 12769421,
    "key": "AURORA-1047"},...],
 "incompletedIssues":[
   {"id": 12740639,
    "key": "AURORA-698"},...],
 "puntedIssues": [], ...
 "issueKeysAddedDuringSprint": {
   "AURORA-1267": true,
   "AURORA-1321": true,...}},
 "sprint": {
   "id": 127,
   "sequence": 127,
   "name": "Twitter Aurora Q2'15 Sprint 3",
   "state": "CLOSED",
   "startDate": "12/May/15 6:59 AM",
   "endDate": "26/May/15 4:00 AM",...}}
```

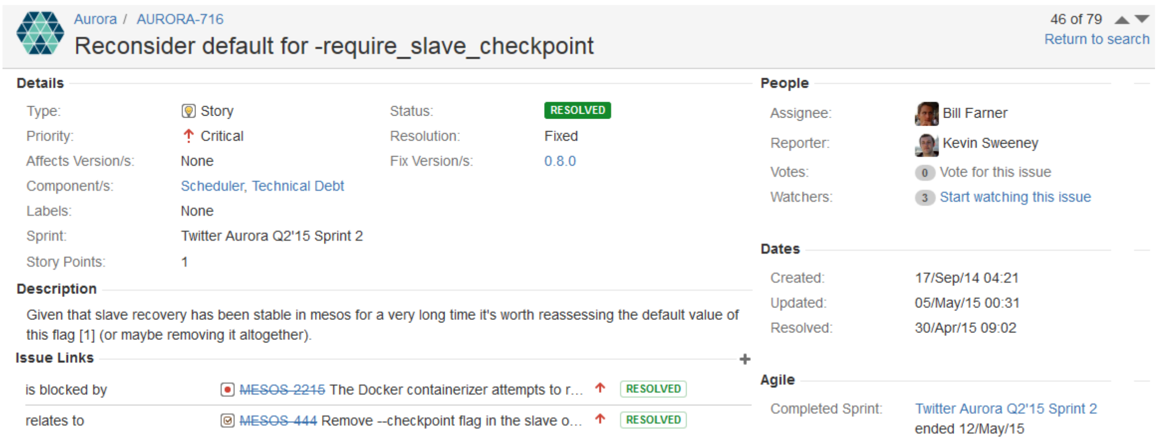
Figure 3.5: Example of an iteration report in JSON format of the iteration named *"Twitter Aurora Q2'15 Sprint 3"* in the Apache project

broad range of features representing an issue (see Table 3.2). The features cover different aspects of an issue including primitive attributes of an issue, issue dependency, changing of issue attributes, and textual features of an issue's description. Some of the features of an issue (e.g. number of issue links) were also adopted from our previous work [127].

It is important to note that we used the time when a prediction is made (prediction time t_{pred}) as the reference point when extracting the values of *all* the features. By processing an issue's change log during both training and testing phases we collected the value which a feature had just before the prediction time. For example, if the final number of comments on an issue is 10, and there were no comments at the time when the prediction was made, then the value of this feature is 0. The underlying principle here is that: when making a prediction, we try to use only information available just *before* the prediction time. The purpose of doing this is to prevent using "future" data when making a prediction – a phenomenon commonly referred in machine learning as information leakage [128].

```
{"key": "AURORA-1267",
   "changelog": { "histories": [...
  {"id": "14758778",
   "created": "2015-05-16T00:31:55.018+0000",
   "items": [{
     "field": "Sprint",
     "fieldtype": "custom",
     "from": null,
     "fromString": null,
     "to": "127",
     "toString":"Twitter Aurora Q2'15 Sprint 3"}
   ]},...]}}
```

Figure 3.6: Example of a change log of an issue in JSON format of issue *AURORA-1267*


Aurora / AURORA-716

Reconsider default for -require_slave_checkpoint

46 of 79

Return to search

Details

Type:	Story	Status:	RESOLVED
Priority:	Critical	Resolution:	Fixed
Affects Version/s:	None	Fix Version/s:	0.8.0
Component/s:	Scheduler, Technical Debt		
Labels:	None		
Sprint:	Twitter Aurora Q2'15 Sprint 2		
Story Points:	1		

Description

Given that slave recovery has been stable in mesos for a very long time it's worth reassessing the default value of this flag [1] (or maybe removing it altogether).

Issue Links

is blocked by	~~MESOS-2215~~ The Docker containerizer attempts to r...	RESOLVED
relates to	~~MESOS-444~~ Remove --checkpoint flag in the slave o...	RESOLVED

People

Assignee:	Bill Farner
Reporter:	Kevin Sweeney
Votes:	0 Vote for this issue
Watchers:	3 Start watching this issue

Dates

Created:	17/Sep/14 04:21
Updated:	05/May/15 00:31
Resolved:	30/Apr/15 09:02

Agile

Completed Sprint:	Twitter Aurora Q2'15 Sprint 2 ended 12/May/15

Figure 3.7: An example of an issue report of issue *AURORA-716* in the Apache project

Primitive attributes of an issue

These features are extracted directly from the issue's attributes, which include type, priority, number of comments, number of affect versions, and number of fix versions. Each issue will be assigned a type (e.g task, bug, new feature, improvement, and story) and a priority (e.g. minor, major, and critical). These indicate the nature of the task associated with resolving the issue (e.g. new feature implementation or bug fixing) and the order in which an issue should be attended with respect to other issues (e.g. a team should concerns critical priority issues more than issues with major and minor priority). These attributes might affect the delivery of an iteration. For example, in the Apache project approximately 10% of the under-achieved iterations have at least one *critical* priority issue.

Table 3.2: Features of an issue

Feature	Description
Type	Issue type
Priority	Issue priority
No. of comments	The number of comments
No. of affect versions	The number of versions for which an issue has been found
No. of fix versions	The number of versions for which an issue was or will be fixed
Issue links	The number of dependencies of an issues
No. of blocking issues	The number of issues that block this issue for being resolved
No. of blocked issues	The number of issues that are blocked by this issue
Changing of fix versions	The number of times in which a fix version was changed
Changing of priority	The number of times an issue's priority was changed
Changing of description	The number of times in which an issue description was changed
Complexity of description	The read ability index (Gunning Fog [129]) indicates the complexity level of a description which is encoded to easy and hard

Previous studies (e.g. [130]) have found that the number of comments on an issue indicates the degree of team collaboration, and thus may affect its resolving time. The "affect version" attribute of an issue specifies versions in which an issue (e.g. bug) has been found, while the "fix version" attribute indicates the release version(s) for which the issue was (or will be) fixed. One issue can be found in many versions. An issue with a high number of affect versions and fix versions needs more attention (e.g. an intensive reviewing process) to ensure that the issue is actually resolved in each affect version and does not cause new problems for each fix version. For example, in Apache, we found that 80% of the over-achieved iterations have issues assigned to only one fix version. We also found that 75.68% of the issues were assigned at least one affect version and fix version.

Dependency of an issue

We extract the dependency between the issues in a term of the number of issue links. Issue linking allows teams to create an association between issues (e.g. an issue resolution may depend on another issue). Figure 3.7 also shows an example of issue links of issue *AURORA-716* in the Apache project for which it has been blocked by issue *AURORA-MESOS-2215*. There are several relationship types for issue links (e.g. relate to, depend on). In our approach we consider all types of issue links and use the number of those links as features. Moreover, blocker is one of the issue linking types that indicates the complexity of resolving issue since these blocker issues block other issues from being completed (i.e. all blocker issues need to be fixed beforehand). As such they directly affects the progress and time allocated to solve other issues [131]–[133]. The blocker relationship is thus treated separately as two features: number of issues that are blocked by this issue and number of issues that block this issue. We found that there are more than 30% of issues have at least one relationship. We also note that when counting the number of links between issues, we count each link type separately. For example, if there are three different types of links between issues A and B, the number of links counted would be 3.

Changing of issue attributes

Previous research, e.g., [133], [134], has shown that changing of an issue's attribute (e.g. priority) may increase the issue resolving time and decrease the deliverable capability which it could be a cause of delays in software project. In our study, there are three features reflecting changing of issue's attributes which are: the number of times an issue priority was reassigned, the number of times in which a fix version was changed, and the number of times in which an issue description was changed. The changing of an issue's priority may indicate the shifting of its complexity. In addition, the changing of the fix version(s) reflects some changes in the release planning for which it affects directly the planning of on-going iterations. In particular, changing the description of an issue could indicate that the issue is not stable and could also create misunderstanding. These may consequently have an impact on the issue resolution time.

Textual features of an issue's description

An issue's description text can provide good features since it explains the nature of an issue. A good description helps the participant of an issue understand its nature and complexity. We have a employed readability measure to derive textual features from the textual description of an issue. We used Gunning Fox [129] to measure the complexity level of the description in terms of a readability score (i.e. the lower score, the easier to read). Previous studies (e.g. [135]) have found that issues with high readability scores were resolved more quickly. We acknowledge that there are more advanced techniques with which to derive features from textual data (e.g. word2vec), use of which we leave for our future work.

3.2.3 Feature aggregation

As previously stated, to characterize an iteration, we extracted both the features of an iteration and the features of issues assigned to it (i.e. one iteration associates with a number of issues). Feature aggregation derives a new set of features from the issues (i.e. a number of issues) for an iteration by aggregating those features of the issues assigned to the iteration. We discuss here two distinct feature aggregation techniques (i.e. statistical aggregation and Bag-of-Words) we use to characterize an iteration using features of the issues assigned to it. The complexity descriptors of a graph describing the dependencies among those issues also form a set of features representing the iteration. These aggregation approaches aim to capture the characteristics of issues associated to an iteration in different aspects which each of them could reflects the situation of an iteration. The features of an iteration and its aggregated features of the issues are then fed into a classifier to build a predictive model that we discuss in Section 3.3.

Statistical aggregation

Statistical aggregation aggregates the features of issues using a number of statistical measures (e.g. max, min, and mean) which aims to capture the statistical characteristics of the issues assigned to an iteration. For each feature k in the set of features of an issue (see Table 3.2), we have a set of values $V_k = \{x_1^k, x_2^k, ..., x_n^k\}$ for this feature where x_i^k ($i \in [1..n]$) is the value of feature k of issue x_i in an iteration, and n is the number of issues assigned to this iteration. Applying different statistics over this set V_k (e.g. min, max, mean, median, standard deviation, variance, and frequency) gives different aggregated features. Table 3.3 shows eight basic statistics that we used for our study. Note that the categorical features (e.g. type and priority) are aggregated by summing over the occurrences of each category. For example, minimum, maximum, mean, and standard deviation of number of comments, and frequency of each type (e.g. number of issues in an iteration that are "bug" type) are the features among the aggregated features that characterize an iteration.

Feature aggregation using Bag-of-Words

The above-mentioned approach require us to *manually* engineer and apply a range of statistics over the set of issues in an iteration in order to derive new features characterizing the iteration. Our work also leverages a new feature learning approach known as Bag-of-Words, which has been widely used in computer vision for image classification (e.g.

Table 3.3: Statistical aggregated features for an issue's feature k

Function	Description
min	The minimum value in V_k
max	The maximum value in V_k
mean	The average value across V_k
median	The median value in V_k
std	The standard deviation of V_k
var	The variance of V_k (measures how far a set of numbers is spread out)
range	The difference between the lowest and highest values in V_k
frequency	The summation of the frequency of each categorical value

[136]), to obviate the need for manual feature engineering. Here, new features for a project's iteration can be *automatically* learned from all issues in the project. We employ unsupervised K-means clustering to learn features for an iteration. Specifically, we apply K-means clustering to all the issues extracted from a given project, which gives us k issue clusters whose centers are in $\{C_1, C_2, ..., C_k\}$. The index of the closest center to each issue in an iteration forms a word in a bag-of-words representing the iteration. The occurrence count of a word is the number of issues closest to the center associated with the word. For example, assume that an iteration has three issues X, Y and Z, and the closest cluster center to X is C_1 while the closest center to Y and Z is C_2. The bag-of-words representing this iteration is a vector of occurrence counts of the cluster centers, which in this case is 1 for C_1, 2 for C_2 and 0 for the remaining clusters' centers. Note that in our study we derived 100 issue clusters ($k = 100$) using *kmeans*[d] package from Matlab. Future work would involve evaluation using a different number of issue clusters.

This technique provides a powerful abstraction over a large number of issues in a project. The intuition here is that the number of issues could be large (hundreds of thousands to millions) but the number of issue types (i.e. the issue clusters) can be small. Hence, an iteration can be characterized by the types of the issues assigned to it. This approach offers an efficient and effective way to learn new features for an iteration from the set of issues assigned to it. It can also be used in combination with the statistical aggregation approach to provide an in-depth set of features for an iteration.

[d]`http://au.mathworks.com/help/stats/kmeans.html`

Graph descriptors

Dependencies often exist between issues in an iteration. These dependency of issues are explicitly recorded in the form of issue links (e.g. relate to, depend on, and blocking). Blocking is a common type of dependency that is recorded in issue tracking systems. For example, blocking issues are those that prevent other issues from being resolved. Such dependencies form a directed acyclic graph (DAG) which depicts how the work on resolving issues in an iteration should be scheduled (similar to the popular activity precedence network in project management). Figure 3.8 shows an example of a DAG constructed from nine issues assigned to the iteration *Usergrid 20*[e] in the Apache project. However note that we consider only the relationships among issues in the same iteration.

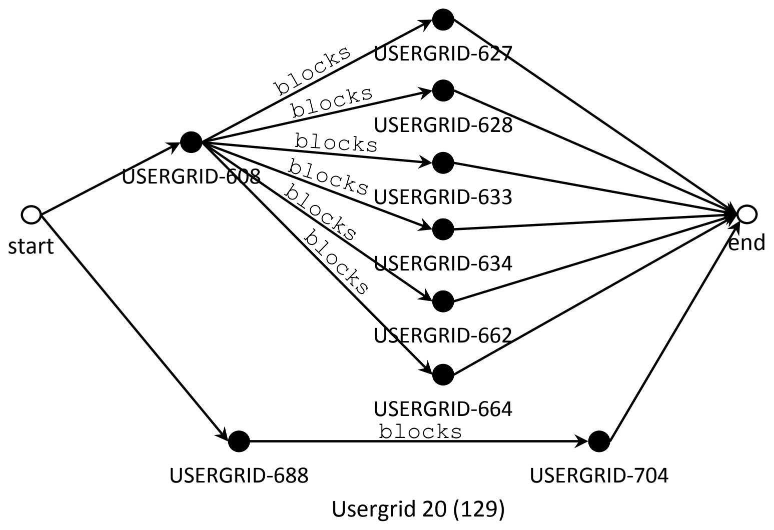


Figure 3.8: Example of a DAG of issues in the iteration “Usergrid 20” in the Apache project

The descriptors of complexity of such a DAG of issues provide us with a rich set of features characterizing an iteration. These include basic graph measures such as the number of nodes in a graph, the number of edges, the total number of incoming edges, and so on (see [137] for a comprehensive list). Table 3.4 lists a set of graph-based features that we currently use. For example, from Figure 3.8, among the aggregated features using graph-based feature aggregation for the iteration *Usergrid 20*, the number of nodes equals 9 and the number of edges equals 7. We acknowledge that the dependencies between issues in an iteration may not exist (e.g. no issue link between issues in an iteration) however the aggregated features from this approach can be combined with the features from our other techniques to characterize an iteration in terms of dependencies between

[e]`https://issues.apache.org/jira/secure/RapidBoard.jspa?rapidView=23&view=reporting&chart=sprintRetrospective&sprint=129`

Table 3.4: Features of a DAG of issues in an iteration

Graph measure	Description
number of nodes	The number of issues in DAG
number of edges	The number of links between issues in DAG
sum of fan in	The total number of incoming links of issues in DAG
sum of fan out	The total number of outgoing links of issues in DAG
max of fan in	The maximum number of incoming links of issues in DAG
max of fan out	The maximum number of outgoing links of issues in DAG
mean of fan in	The average of numbers of incoming links across all issues in DAG
mean of fan out	The average of numbers of outgoing links across all issues in DAG
mode of fan in	The number of incoming links that appear most often in DAG
mode of fan out	The number of outgoing links that appear most often in DAG
avg. node degree	The degree distribution of nodes and edges of DAG

issues assigned to it. Future work would involve exploring some other measures such as graph assortativity coefficient [138].

3.3 Predictive model

Our predictive models can predict the difference between the actual delivered velocity against the committed (target) velocity for an iteration, i.e. $velocity(\texttt{Difference})$. To do so, we employ regression methods (supervised learning) where the outputs reflect the deliverable capability in an iteration e.g., the predicting of $velocity(\texttt{Difference})$ will be equal to 12. The extracted features of the historical iterations (i.e. training set) are used to build the predictive models. Specifically, a feature vector of an iteration and an aggregated feature vector of issues assigned to the iteration are concatenated and fed into a regressor.

We apply the currently most successful class of machine learning methods, namely *randomized ensemble methods* [139]–[141]. Ensemble methods refer to the use of many regressors to make their prediction [141]. Randomized methods create regressors by randomizing data, features, or internal model components [142]. Randomizations are powerful regularization techniques which reduce prediction variance, prevent overfitting, are robust against noisy data, and improve the overall predictive accuracy [143], [144].

We use the following high performing regressors that have frequently won recent data science competitions (e.g. Kaggle[f]): *Random Forests* (RFs) [145], *Stochastic Gra-*

[f]`https://www.kaggle.com`

dient Boosting Machines (GBMs) [46], [146] and *Deep Neural Networks with Dropouts* (DNNs) [58]. All of them are ensemble methods that use a divide-and-conquer approach to improve performance. The key principle behind ensemble methods is that a group of "weak learners" (e.g. classification and regression decision trees) can together form a "strong learner".

3.3.1 Random Forests

Random forests (RFs) [145] uses decision trees as weak learners and typically works as follows. First, a subset of the full dataset is randomly sampled (i.e. 200 out of 1,000 iterations) to train a decision tree. At each node of the decision tree, we normally search for the best feature across *all* the features (predictor variables) to split the training data. Instead of doing so, at each node of a decision tree, RFs randomly selects a subset of candidate predictors and then find the best splitting predictor for the node. For example, if there are 200 features, we might select a random set of 20 in each node, and then split using the best feature among the 20 available, instead of the best among the full 200 features. Hence, RFs introduces randomness not just into the training samples but also into the actual trees growing.

This process is repeated: a different random sample of data is selected to train a second decision tree. The predictions made by this second tree is typically different from those of the first tree. RFs continues generating more trees, each of which is built on a slightly different sample and producing slightly different predictions each time. We could continue this process indefinitely, but in practice 100 to 500 trees are usually generated. To make predictions for a new data, RFs combines all separate predictions made by each of the generated decision tree typically by averaging the outputs across all trees.

3.3.2 Stochastic Gradient Boosting Machines

RFs grows independent decision trees (which thus can be done in parallel) and simply takes the average of the predictions produced by those trees as the final prediction. On the other hand, gradient boosting machines (GBMs) [46], [146] generates trees and adds them to the ensemble in a *sequential* manner. The first tree is generated in the same way as done in RFs. The key difference here is the generation of the second tree which aims at minimizing the prediction errors produced by the first tree (thus the trees are not independent to each other as in RFs). Both the first and second trees are added to the ensemble but different weights are assigned to each of them. This process is repeated multiple times: at each step, a new tree is trained with respect to the error of the whole ensemble learnt so far and is then added to the ensemble. The final ensemble is used as a model for predicting the outcome of new inputs.

Unlike RFs, a weak learner in GBMs can be not just only regression trees but also any other regression learning algorithms such as neural networks or linear regression. In our implementation, regression trees are used as weak learners and 100 trees were generated.

3.3.3 Deep Neural Networks with Dropouts

Neural networks have long been used in many prediction tasks where the input data has many variables and noises. The neural network is organized in a series of layers: the bottom layer accepting the input, which is projected to a hidden layer, which in turn projects to an output layer. Each layer consists of a number of computation units, each of which is connected to other units in the next layer. Deep neural networks (DNNs) are traditional artificial neural networks with multiple hidden layers, which make them very expressive models that are capable of learning highly complicated relationships between their inputs and outputs. Limited training data may however cause overfitting problem, where many complicated relationships exist in the training data but does not occur in the real test data.

Building an ensemble of many differently trained neural networks would alleviate the overfitting problem (as seen in RFs and GBMs). To achieve the best performance, each individual neural network in the ensemble should be different from each other in terms of either the architecture or the data used for training. *Dropout* [58] is a simple but scalable technique which, given a main network, builds an ensemble from many variants of this network. These variant networks are generated by temporarily removing one random unit (and its incoming and outgoing connections) at a time from the main network. Hence, a neural network with n units can be used to generate an ensemble of 2^n networks. After being trained, those 2^n networks can be combined into a single neural network to make predictions for the new inputs.

3.4 Dataset

In this section, we describe how data were collected for our study and experiments.

3.4.1 Data collecting and preprocessing

We collected the data of past iterations (also referred to as sprints in those projects) and issues from five large open source projects which follow the agile Scrum methodology: Apache, JBoss, JIRA, MongoDB, and Spring. The project descriptions and their agile adoptions have been reported in Table 3.5. All five projects use *JIRA-Agile* for their development which is a well-known issue and project tracking tool that supports agile practices. We use the Representational State Transfer (REST) API provided by JIRA to query and collect iteration and issue reports in JavaScript Object Notation (JSON) format. Note that the JIRA Agile plug-in supports both the Scrum and Kanban practices, but we collected only the iterations following the Scrum practice. From the JIRA API, we were also able to obtain issues' change log and the list of complete/incomplete issues of each iteration.

We initially collected 4,301 iterations from the five projects and 65,144 issues involved with those iterations from *February 28, 2012* to *June 24, 2015*. The former date is the date when the first iteration (among all the case studies) was created and the later is the date when we finished collecting the data. Hence, this window ensures that we did not miss any data up to the time of our collection. The data was preprocessed by removing duplicate and demonstration iterations as well as future and ongoing iterations. We also removed iterations that have zero resolved issues where those iterations are ignored by teams (e.g. no activity recorded in the issue report, all issues have been removed).

In total, we performed our study on 3,834 iterations from the five projects, which consist of 56,687 issues. Table 3.6 shows the number of iterations and issues in our datasets and summarizes the characteristics of the five projects in terms of the iteration length, number of issues per iteration, number of team members per iteration in terms of the minimum, maximum, mean, median, and standard deviations (STD). The iteration length across the five projects tends to be in the range of 2 to 4 weeks. All projects have almost the same team size, while the number of issues per iteration varies. For example, the mean number of issues per iteration in MongoDB is 15 issues, while it is only 5.5 issues in JIRA.

Table 3.5: Project description

Project	Brief description	Agile adoption	Year of adoption
Apache	A web server originally designed for Unix environments. There are more than fifty sub-projects under the Apache community umbrella (i.e. Aurora, Apache Helix, and Slider).	Almost 50% of the issues in the sub-projects that are applied the Scrum method (e.g. Aurora) have been assigned to at least one iteration.	2006
JBoss	An application server program which supports a general enterprise software development framework.	JBoss community has described their iterative development process in the developer guide.$^+$ There are more than ten sub-projects that are applied agile methods.	2004*
JIRA	A project and issue tracking system including the agile plug-in that provides tools to manage iterations following agile approaches (e.g. Scrum, Kanban) provided by Atlassian.	Recently, Atlassian reported their success in applying agile methods to improve their development process.^{x}	N/A
MongoDB	A cross platform document-oriented database (NoSQL database) provided by MongoDB corporation and is published as free and open-source software.	More than 70% of the issues in the main sub-projects (e.g. Core MongoDB, MongoDB tool) have been assigned to at least one iteration	2009
Spring	An application development framework that contains several sub-projects in their repository i.e. Spring XD and Spring Data JSP.	Almost 70% of the issues in the core sub-projects (e.g. Spring XD) have been assigned to at least one iteration.	N/A

$^+$:`http://docs.jboss.org/process-guide/en/html/`,
*: identified from the developer guided published date,
^{x}:`http://www.businessinsider.com.au/atlassian-2016-software-developer-survey-results-2016-3`

Table 3.6: Descriptive statistics of the iterations of the projects in our datasets

Project	# iterations	# issues	# days/iteration				# issues/iteration				# team members /iteration			
			min/max	mean	median	SD	min/max	mean	median	SD	min/max	mean	median	SD
Apache	348	5,826	3/21	12.02	14	10.12	3/128	15.39	7	8.24	3/21	4.6	4	3.4
JBoss	372	4,984	3/49	12.52	13	7.4	4/122	7.14	5	8.54	2/20	2.6	2	3.11
JIRA	1,873	10,852	4/60	10.5	9	6.03	3/88	5.5	5	6.2	2/12	3.71	2	2.07
MongoDB	765	17,528	3/43	20	18	36	3/180	15	9	21.4	2/30	4.12	3	5.02
Spring	476	17,497	3/49	14.28	14	7.21	3/161	20.71	17	20.11	3/15	5.4	4	4.37
Total	3,834	56,687												

iterations: number of iterations, # issues: number of issues, # days/iteration: number of days per iteration,
issues/iteration: number of issues per iteration, # team members/iteration: number of team members per iteration

3.5 Evaluation

This section discusses an extensive evaluation that we have carried out of our approach. We describe the experimental setting, discuss the performance measures, and report our results. Our empirical evaluation aims to answer the following research questions:

RQ1 *Do the feature aggregation approaches improve the predictive performance?*

We build predictive models using the three feature aggregation approaches and compare them against a predictive model using only the information extracted directly from the attributes of an iteration (i.e. features of an iteration). This is to evaluate whether the aggregated features of issues offer significant improvement. We also investigate which combinations of the feature aggregation approaches are the best performer (e.g. combining the aggregated features from statistical feature aggregation and Bag-of-words aggregation approach).

RQ2 *Do randomized ensemble methods improve the predictive performance compared to a traditional regression model?*

We employ Support Vector Machine (SVM) as a representative for traditional regression models. SVM is the most important (deterministic) classifier in the year 2000s [147]. Its predictive power has been challenged, only recently, by randomized and ensemble techniques (e.g, see the recent comparative study proposed by Fernández-Delgado et al [148]). SVM has been widely used in software analytics, such as defect prediction, effort estimation, and bug localization. Its regression version, Support Vector Regression (SVR), is also known for being highly effective for regression problems. Thus, we employ SVR to build predictive models to evaluate whether our randomized ensemble methods perform better than the traditional regression model. We also find the best randomized ensemble method in predicting the difference between actual achieved and target velocity in an iteration.

RQ3 *Are the purposed randomized ensemble method and the feature aggregation suitable for predicting the difference between the actual delivered velocity against the target velocity of an iteration?*

This is our sanity check as it requires us to compare our purposed prediction model with three common baseline benchmarks used in the context of effort estimation: Random Guessing, Mean Effort, and Median Effort. Random guessing performs random sampling (with equal probability) over the set of iterations with known difference (between target and actual achieved velocity), chooses randomly one iteration from the sample, and uses the target vs. actual difference velocity of that iteration as the prediction of the new iteration. Random guessing does not use any information associated with the new iteration. Thus any useful prediction model should outperform random guessing. Mean and Median Effort predictions are commonly used as baseline benchmarks for effort estimation. They use the mean or median target vs. actual difference of the past iterations to predict the difference of the new iterations.

RQ4 *Does the time of making a prediction (t_{pred}) affect the predictive performance?*

We want to evaluate the predictive performance from the different prediction times (t_{pred}) to confirm our hypothesis that the later we predict, the more accuracy we gain. Specifically, we evaluate the predictive performance from four different prediction times: at the beginning of an iteration, and when it progresses to 30%, 50%, and 80% of its planned duration. We acknowledge that making a prediction as late as at 80% of an iteration duration may not be particularly useful in practice. However, for the sake of completeness we cover this prediction time to sufficiently test our hypothesis. Note that our experiments in RQ1-RQ3 were done at the prediction time when an iteration has progressed to 30% of its planned duration (e.g. make a prediction at the third day of a 10-day iteration).

RQ5 *Can the output from the predictive model (i.e. the difference between the actual delivered velocity against the target velocity) be used for classifying the outcomes of an iteration (e.g. an under-achieved iteration)?*

Rather than the difference between the actual achieved and the target velocity, the outcomes of iterations can also be classified into three classes: below the target – *under achieved*, i.e. $velocity(\texttt{Committed}) > velocity(\texttt{Delivered})$, or above the target – *over achieved*, i.e. $velocity(\texttt{Committed}) < velocity\ (\texttt{Delivered})$, or the same as the target – *achieved*, i.e. $velocity(\texttt{Committed}) = velocity\ (\texttt{Delivered})$.

We want to evaluate whether the output from the predictive models (i.e. the difference velocity) can be used to classify the outcome of an iteration in terms of the three classes: negative outputs, i.e. $velocity(\texttt{Difference}) < 0$, are in the *under-achieved* class, positive outputs, i.e. $velocity(\texttt{Difference}) > 0$, are in the *over-achieved* class, and zero, i.e. $velocity(\texttt{Difference}) = 0$, is in the *achieved* class. This method can also accommodate a tolerance margin e.g. outputs from -1 to 1 are considered in the achieved class. A finer-grained taxonomy (e.g. different levels of over achieved or under achieved) for classifying iterations can also be

Table 3.7: Descriptive statistics of the difference between the actual delivered velocity against the target velocity in each project

Project	$velocity$(Difference)						
	min	max	mean	median	mode	SD	IQR
Apache	-81	49	-9.05	-5	0	17.16	15.00
JBoss	-96	30	-6.03	-1	0	14.83	4.00
JIRA	-83	117	-2.82	0	0	11.37	3.00
MongoDB	-67	50	-0.99	0	0	11.54	5.00
Spring	-135	320	23.77	8.50	4	51.66	32.00

accommodated to reflect the degree of difference between the target and the actual delivered velocity.

3.5.1 Experimental setting

All iterations and issues collected in each of the five case studies were used in our evaluations. As discussed in Section 3.1, we would like to predict the difference between the actual delivered velocity against the target velocity. For example, if the output of our model is -5, it predicts that the team will deliver 5 story points below the target. Table 4.3 shows the statistical descriptions of the difference between the actual delivered against the target velocity of the five projects in terms of the minimum, maximum, mean, median, mode, standard deviations (SD), and interquartile range (IQR).

We used ten-fold cross validation in which all iterations were sorted based on their start date. After that, an iteration i^{th} in every ten iterations is included in fold i^{th}. With larger data sets, we could choose the sliding window setting, which mimics a real deployment scenario, to ensure that prediction on a current iteration is made by using knowledge from the past iterations. We leave for future work since we believe that our findings in the current experimental setting still hold [149]. The time when the prediction is made may affect its accuracy and usefulness. The later we predict, the more accurate our prediction gets (since more information has become available) but the less useful it is (since the outcome may become obvious or it is too late to change the outcome). We later address the varying of the prediction time in RQ4.

3.5.2 Performance measures

There are a range of measures used in evaluating the accuracy of a predictive model in teams of effort estimation. Most of them are based on the Absolute Error, (i.e. $|ActualDiff-$

$EstimatedDiff|$) where $AcutalDiff$ is the real $velocity$(`Difference`) and $EstimatedDiff$ is the predicted $velocity$(`Difference`) given by a predictive model. Mean of Magnitude of Relative Error (MRE) and Prediction at level l [150], i.e. Pred(l), have also been used in effort estimation. However, a number of studies [151]–[154] have found that those measures bias towards underestimation and are not stable when comparing the models. Thus, the *Mean Absolute Error (MAE)* has recently been recommended to compare the performance of effort estimation models [7]. However, different projects have different $velocity$(`Difference`) ranges (see Table 4.3). Thus, we needed to normalize the MAE (by dividing it with the interquartile range) to allow for comparisons of the MAE across the studied project. Similarly to the work in [155], we refer to this measure as Normalized Mean Absolute Error (NMAE).

To compare the performance of two predictive models, we also applied statistical significance testing on the absolute errors predicted by the two models using the Wilcoxon Signed Rank Test [156] and employed the Vargha and Delaney's $\hat{A}_{12}$ statistic [157] to measure whether the effect size is interesting. Later in RQ5, we used a number of traditional metrics: precision, recall, F-measure, Area Under the ROC Curve (AUC) to measure the performance in classifying the outcome of an iteration. We also used another metric called Macro-averaged Mean Absolute Error (MMAE) [158] to assess the distance between actual and predicted classes since the classes is ordinal, we can order them, e.g. *over achieved* is better than *achieved*. The traditional class-based measures (Precision/Recall) do not take into account the ordering between classes, and Matthews Correlation Coefficient (MCC) [159], which performs well on imbalanced data.

Normalized Mean Absolute Error (NMAE)

The Normalized Mean Absolute Error (NMAE) [155] is defined as:

$$NMAE = \frac{\frac{1}{N}\sum_{i=1}^{N} |ActualDiff_i - EstimatedDiff_i|}{IQR}$$

where N is the number of iterations used for evaluating the performance (i.e. test set), $ActualDiff_i$ is the actual $velocity$(`Difference`), $EstimatedDiff_i$ is the predicted $velocity$(`Difference`) for the iteration i, and IQR is the distance between the $velocity$(`Difference`) at 75th and 25th percentile (i.e. $75^{th} percentile - 25^{th} percentile$).

Statistical comparison

We assess the $velocity(\texttt{Difference})$ produced by the predictive models using NMAE. To compare the performance of two predictive models, we tested the statistical significance of the absolute errors achieved with the two models using the Wilcoxon Signed Rank Test [156]. The Wilcoxon test is a safe test since it makes no assumptions about underlying data distributions. The null hypothesis here is: "the absolute errors provided by a predictive model are significantly less that those provided by another predictive model". We set the confidence limit at 0.05 (i.e. $p < 0.05$). In addition, we also employed a non-parametric effect size measure, the Vargha and Delaney's $\hat{A}_{12}$ statistic [157] to assess whether the effect size is interesting. The $\hat{A}_{12}$ measure is chosen since it is agnostic to the underlying distribution of the data, and is suitable for assessing randomized algorithms in software engineering generally [160] and effort estimation in particular [7]. Specifically, given a performance measure (e.g. the Absolute Error from each prediction in our case), the $\hat{A}_{12}$ measures the probability that predictive model M achieves better results (with respect to the performance measure) than predictive model N using the following formula: $\hat{A}_{12} = (r_1/m - (m+1)/2)/n$ where r_1 is the rank sum of observations where M achieving better than N, and m and n are respectively the number of observations in the samples derived from M and N. If the performance of the two models are equivalent, then $\hat{A}_{12} = 0.5$. If M perform better than N, then $\hat{A}_{12} > 0.5$ and vice versa. All the measures we have used here are commonly used in evaluating effort estimation models [7], [160].

Precision/Recall/F-measures/AUC

A confusion matrix is used to store the correct and incorrect classifications for each individual class made by a predictive model. For example, the confusion matrix for class *under achieved* in predicting the target velocity against the actual velocity delivered is constructed as follows. If an iteration is classified as *under achieved* when it truly delivered below than the target, the classification is a true positive (tp). If the iteration is classified as *under achieved* when it is actually *over achieved* or *achieved*, then the classification is a false positive (fp). If the iteration is classified as not *under achieved* when it in fact deliver below than the target, then the classification is a false negative (fn). Finally, if the iteration is classified as not *under achieved* and it in fact is did not deliver below the target, then the classification is true negative (tn). The values of each individual class classification stored in the confusion matrix are used to compute the widely used Precision, Recall, and F-measure [161]. In addition, we used another measure, Area Under the

ROC Curve (AUC), to evaluate the degree of discrimination achieved by the model. We have described these performance measures in details in Section 2.1.6 of Chapter 2.

Matthews correlation coefficient (MCC)

To compare the performance between two cases, using only F-measure can mislead the interpretation (e.g. very high precision but very poor recall), especially in cases of class imbalance. We thus also used Matthews correlation coefficient (MCC) [162]. MCC takes into account all true and false positives and negatives values (tp, tn, fp, and fn) and summarizes into a single value. It is also generally known as a balanced measure which can be used even if the classes are very different sizes [163]. MCC is defined as:

$$MCC = \frac{tp \times tn - fp \times fn}{\sqrt{(tp+fp) \times (tp+fn) \times (tn+fp) \times (tn+fn)}}$$

MCC has a range of -1 to 1 where -1 indicates a completely wrong classifier while 1 indicates a completely correct classifier, and 0 is expected for a prediction that no better than random.

Macro-averaged Mean Absolute Error (MMAE)

Since the classes in each prediction task could be considered as ordinal, we can order them, e.g. *over achieved* is better than *achieved*, and *achieved* is better than *under achieved*. However, the traditional class-based measures (e.g. Precision and Recall) do not take into account the ordering between classes. Hence, we also used another metric called Macro-averaged Mean Absolute Error (MMAE) [158] to assess the distance between actual and predicted classes. MMAE is suitable for ordered classes and insensitive to class imbalance. For example, as we discussed that the ordering of our classes is based on the performance of an outcome.

Let y^i be the true class and $\hat{y}^i$ be the predicted class of iteration i in the task. Let n_k be the number of true cases with class k where $k \in \{1,2,3\}$ – there are 3 classes in our classification – i.e., $n_k = \sum_{i=1}^{n} \delta\left[y^i = k\right]$ and $n = n_1 + n_2 + n_3$. The Macro-averaged Mean Absolute Error is computed as follows.

$$\text{MMAE} = \frac{1}{3}\sum_{k=1}^{3}\frac{1}{n_k}\sum_{i=1}^{n}\left|\hat{y}^i - k\right| \delta\left[y^i = k\right]$$

For example, if the actual class is *over achieved* ($k = 1$), and the predicted class is *under achieved* ($k = 3$), then an error of 2 has occurred. Here, we assume that the predicted class is the one with the highest probability, but we acknowledge that other strategies can be used in practice. We however note that the ordering of the classes can be changed based on the project settings. For example, the achieved class may be more preferred than the over achieved class since over-achieving is not suggested over staying within budget.

3.5.3 Results

We report here our evaluation results in answering our research questions RQs 1—5.

Benefits of the feature aggregations of issues (RQ1)

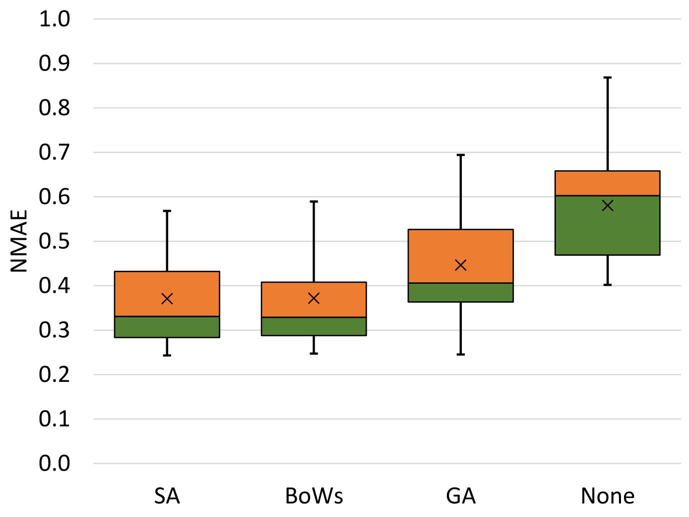


Figure 3.9: Evaluation results on the three aggregated features and the features of iterations (None). X is a mean of NMAE averaging across all projects and all regression models (the lower the better).

We compared the predictive performance (using NMAE) achieved for the three feature aggregation approaches: statistical aggregation (SA), Bag-of-Words (BoWs), and graph-based aggregation (GA) against the predictive model using only the features of an iteration (None). Figure 3.9 shows the NMAE achieved for each of the aggregated features, averaging across all the projects and all the regression models. **The analysis of NMAE suggests that the predictive performance achieved from using the feature aggregation approaches (i.e. SA, BoWs, and GA) consistently outperforms the predictive model using only the features of iterations (i.e. None).** The predictive models

Table 3.8: Comparison of the predictive models between with and without the aggregated features using Wilcoxon test and A^{12} effect size (in brackets)

Project	With agg. vs	Without agg.	
Apache	SA	<0.001	[0.61]
	BoWs	<0.001	[0.60]
	GA	<0.001	[0.56]
JBoss	SA	<0.001	[0.59]
	BoWs	<0.001	[0.60]
	GA	<0.001	[0.54]
JIRA	SA	<0.001	[0.62]
	BoWs	<0.001	[0.56]
	GA	<0.001	[0.52]
MongoDB	SA	<0.001	[0.61]
	BoWs	<0.001	[0.60]
	GA	<0.001	[0.53]
Spring	SA	<0.001	[0.60]
	BoWs	<0.001	[0.61]
	GA	<0.001	[0.54]

using statistical aggregation, Bag-of-Words, and graph-based aggregation achieve an accuracy of 0.371, 0.372, and 0.446 NMAE respectively, while the predictive models using only the features of iterations achieve only 0.581 NMAE.

Table 3.8 shows the results of the Wilcoxon test (together with the corresponding A^{12} effect size) to measure the statistical significance and effect size (in brackets) of the improved accuracy achieved by the aggregated features over the features of iterations. In *all* cases, the predictive models using the feature aggregation approaches significantly outperform the predictive models using only the features of iterations ($p < 0.001$) with effect size greater than 0.5.

We also performed a range of experiments to explore the best combination of aggregated features. There are four possible combinations: SA+GA, BoWs+GA, SA+BoWs, and the combination of all of them (All). For example, SA+GA combines a feature vector of an iteration, a feature vector of issues obtained from statistical aggregation (SA), and a feature vector of issues obtained from graph-based aggregation (GA). As can be seen in Figure 3.10, in most cases the combination of two or more feature aggregation approaches produced better performance than a single aggregation approach. **However, the best performer varies between projects.** For example, SA+GA outperforms the others in JBoss – it achieves 0.555 NMAE while the others achieve 0.558 - 0.598 NMAE (averaging across all regression models), while SA+BoWs is the best performer in JIRA – it achieves 0.265 NMAE while the others achieve 0.285 - 0.366 NMAE (averaging across all

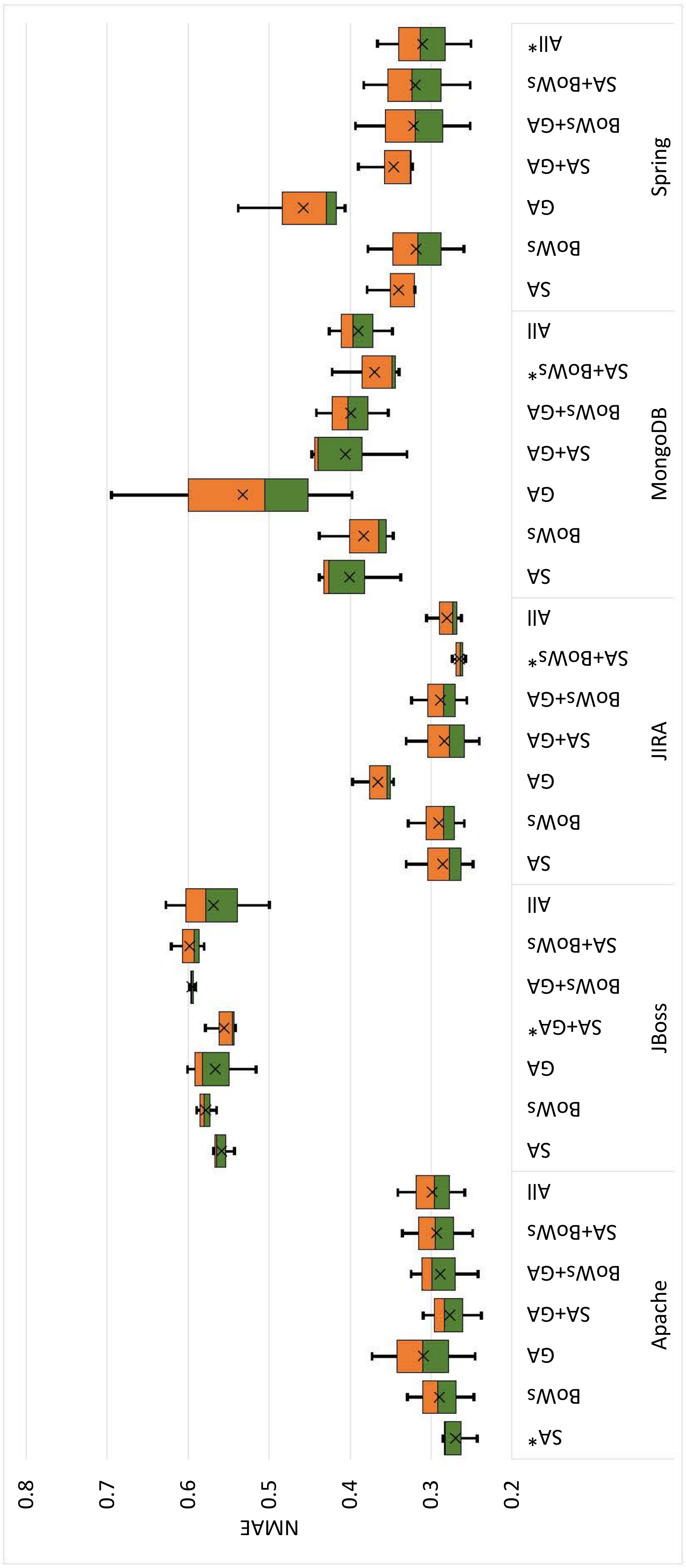


Figure 3.10: Evaluation results on all the combinations of the aggregated features of issues. In each project, the best performer is marked with *. X is a mean of NMAE averaging across all regression models (the lower the better).

regression models). These results suggest that the three approaches are distinct and complementary to each other: the statistical aggregation covers the details, the Bag-of-Words technique addresses the abstraction, while the graph-based approach reflects the network nature of issues in an iteration as also shown in our previous work [132]. We rely not just on features coming directly from the attributes of an object (i.e. an iteration), like most of existing software analytics approaches, but also the features from the parts composing the object (i.e. the issues). The latter features are powerful since they are automatically learned and aggregated, and capture the graphical structure of the object.

Answer to RQ1: Feature aggregation offers significant improvement in predictive performance.

Benefits of the randomized ensemble methods (RQ2)

To answer RQ2, we focus on the predictive performance achieved from different regression models. For a fair comparison we used only one combination of aggregated features that performs best in most cases, SA+BoWs, as reported in RQ1. Figure 3.11 shows the NMAE achieved by the three randomized ensemble methods: Random Forests (RF), Deep Neural Networks with Dropouts (Deep Net.), and Stochastic Gradient Boosting Machines (GBMs), and the traditional Support Vector Regression (SVR), averaging across all projects.

Overall, all the three ensemble methods that we have employed performed well, producing much better predictive performance than the traditional SVR. They achieve 0.392 NMAE averaging across the three ensemble methods, while SVR achieves 0.621 NMAE. The results for the Wilcoxon test to compare the ensemble methods against the tradition regression model is shown in Table 3.9. The improvement of the ensemble methods over the traditional regression model is significant ($p < 0.001$) with the effect size greater than 0.5 all cases. **The best performer is Stochastic Gradient Boosting Machines (GBMs).** GBMs achieved 0.369 NMAE averaging across all projects as confirmed by the results of the Wilcoxon test with the corresponding A^{12} effect size in Table 3.10: GBMs perfomed significantly better than RF and Deep Nets. ($p < 0.001$) with effect size greater than 0.5 in all cases.

Answer to RQ2: Randomized ensemble methods significantly outperform traditional methods like SVR in predicting delivery capability.

Table 3.9: Comparison of the three randomized ensemble methods against the traditional SVR using Wilcoxon test and A^{12} effect size (in brackets)

Project	Method vs	SVR	
Apache	RF	<0.001	[0.61]
	Deep Nets.	<0.001	[0.61]
	GBMs	<0.001	[0.67]
JBoss	RF	<0.001	[0.58]
	Deep Nets.	<0.001	[0.59]
	GBMs	<0.001	[0.66]
JIRA	RF	<0.001	[0.63]
	Deep Nets.	<0.001	[0.63]
	GBMs	<0.001	[0.71]
MongoDB	RF	<0.001	[0.66]
	Deep Nets.	<0.001	[0.70]
	GBMs	<0.001	[0.82]
Spring	RF	<0.001	[0.64]
	Deep Nets.	<0.001	[0.66]
	GBMs	<0.001	[0.73]

Table 3.10: Comparison of GBMs against RF and Deep nets. using Wilcoxon test and A^{12} effect size (in brackets)

Project	Method vs	RF		Deep Nets.	
Apache	GBMs	<0.001	[0.62]	<0.001	[0.60]
JBoss	GBMs	<0.001	[0.60]	<0.001	[0.60]
JIRA	GBMs	<0.001	[0.59]	<0.001	[0.62]
MongoDB	GBMs	<0.001	[0.75]	<0.001	[0.71]
Spring	GBMs	<0.001	[0.67]	<0.001	[0.65]

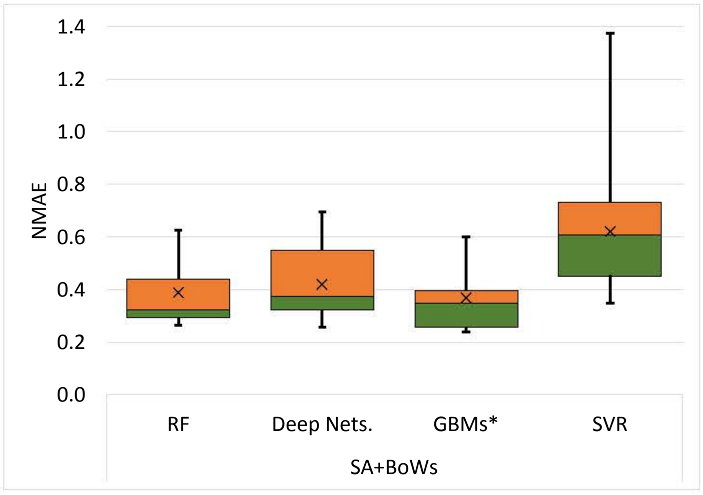


Figure 3.11: Evaluation results on different regression models with SA+BoWs. The best performer is marked with *. X is a mean of NMAE averaging across all projects (the lower the better).

Table 3.11: Comparison on the predictive performance of our approach against the baseline benchmarks using Wilcoxon test and A^{12} effect size (in brackets)

Project	Model vs	Random		Mean		Median	
Apache	GBMs (SA)	<0.001	[0.82]	<0.001	[0.81]	<0.001	[0.77]
JBoss	GBMs (SA+GA)	<0.001	[0.86]	<0.001	[0.86]	<0.001	[0.66]
JIRA	GBMs (SA+BoWs)	<0.001	[0.88]	<0.001	[0.86]	<0.001	[0.67]
MongoDB	GBMs (SA+BoWs)	<0.001	[0.86]	<0.001	[0.85]	<0.001	[0.80]
Spring	GBMs (All)	<0.001	[0.83]	<0.001	[0.83]	<0.001	[0.79]

Sanity check (RQ3)

To answer RQ3, we employed the best randomized ensemble method: GBMs (identified by RQ2), and the best combination of the aggregated features in each project: SA for Apache, SA+GA for JBoss, SA+BoWs for JIRA and MongoDB, and the combined of all (All) for Spring (identified by RQ1). Figure 3.12 shows the predictive performance achieved from GBMs with the best aggregated features and the three baseline methods: Random, Mean, and Median in each project. **Our analysis of this evaluation result suggests that the predictive performance obtained with our approach is better than those achieved by using Random, Mean, and Median in all projects.** Averaging across all the project, our approach (GBMs with the best aggregated features in each project) achieves an accuracy of 0.349 NMAE, while the base of the baselines achieve only 0.702 NMAE. The NMAE produced by our model is higher for JBoss than that for other projects. JBoss is also the project that the baseline methods (Random, Mean and Median) struggled with the most. There are a few reasons which explain this phenomenon. Firstly, JBoss has the smallest number of issues among the five studied projects.

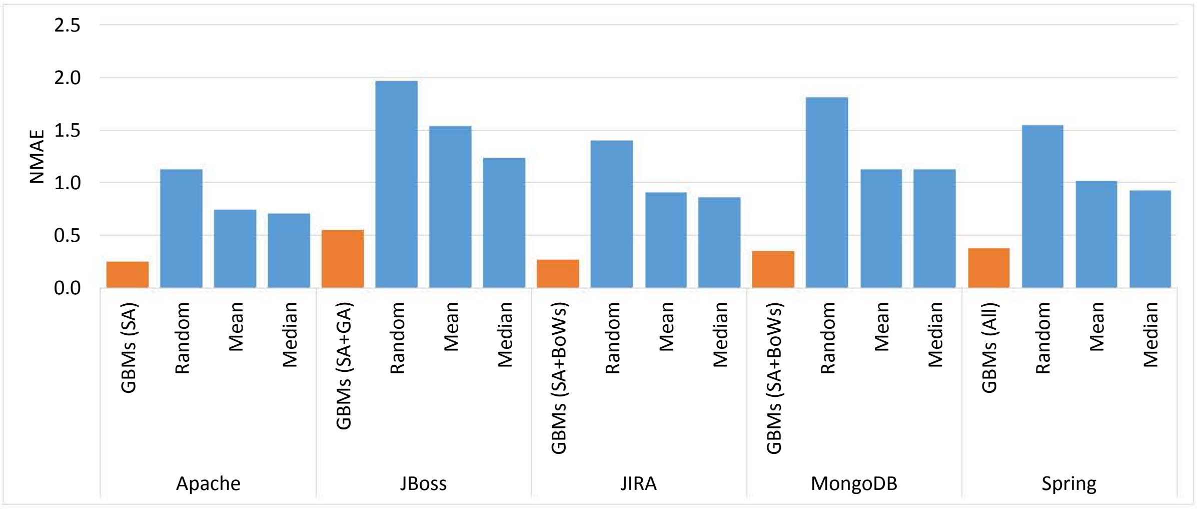


Figure 3.12: Evaluation result of the GBMs with the best aggregated features in each project and the three baseline benchmarks

Small training data may affect the predictive power of a model. Secondly, JBoss has the largest range of story points assigned to issues – it has a standard deviation of 3.27, comparing to 1.71–2.34 standard deviation in the other projects. The high standard deviation indicates that the issue story points in JBoss are spread out over a large range of values. This could make all the models struggle since issue story points directly affect the velocity of an iteration.

Table 3.11 shows the results of the Wilcoxon test (together with the corresponding A^{12} effect size) to measure the statistical significance and effect size (in brackets) of the improved accuracy achieved by our approach over the baselines: Random Guessing, Mean, and Median. In *all* cases, our approach significantly outperforms the baselines ($p < 0.001$) with (large) effect sizes greater than 0.65.

Answer to RQ3: our approach significantly outperforms the baselines, thus passing the sanity check required by RQ3.

The impact of prediction time (RQ4)

We also varied the prediction time (at the beginning of the iteration, and when it progresses to 30%, 50% and 80% of its planned duration) to observe its impact on predictive performance. Note that the extracted features also correspond to the prediction time (e.g. the number of comments of an issue when an iteration progresses to 80% may greater than that at the beginning of the iteration). Moreover, the difference between the target velocity and actual delivered velocity could be dynamic at different prediction times.

Table 3.12: The descriptive statistics of the difference between actual delivered velocity against the target velocity from the different prediction time

Project	Prediction Time(%)	*velocity*(`Difference`)				
		min	max	mean	median	STD
Apache	0	-81	57	-7.25	-4	17.56
	30	-81	49	-9.05	-5	17.16
	50	-81	40	-10.00	-5	16.97
	80	-81	30	-10.86	-6	16.69
JBoss	0	-89	30	-5.12	-1	14.78
	30	-96	30	-6.03	-1	14.83
	50	-96	30	-6.20	-2	15.05
	80	-106	10	-6.35	-2	13.97
JIRA	0	-74	147	-1.61	0	12.19
	30	-83	117	-2.82	0	11.37
	50	-83	96	-3.28	0	11.09
	80	-86	72	-4.07	0	10.68
MongoDB	0	-67	88	1.61	0	12.78
	30	-67	50	-0.99	0	11.54
	50	-67	48	-2.27	-1	11.16
	80	-85	44	-3.65	-1	11.29
Spring	0	-131	332	34.72	15	63.57
	30	-135	320	23.77	8.5	51.66
	50	-135	316	18.79	4	47.92
	80	-147	312	11.07	1	42.13

For example, the actual delivered velocity of the iteration named *Mesosphere Sprint 13* in the Apache project is 72, i.e. *velocity*(`Delivered`) = 72. At the beginning of this iteration, team planned to deliver 15 velocity, i.e. *velocity*(`Committed`) = 15. The difference between actual delivered velocity and target velocity of this iteration when the prediction is done at the beginning of the iteration (0%) is 57, i.e. *velocity*(`Difference`) = 57. When this iteration progressed to 80% of its planned duration, 33 velocity were added and planned to deliver in this iteration, i.e. *velocity*(`Committed`) = 48. Thus, *velocity*(`Difference`) of this iteration when the prediction is done at 80% of the planned duration is 24. Table 3.12 shows the statistical description of *velocity*(`Difference`) in the different prediction time. The decreasing of STD of the different velocity in the later prediction time in all cases shows that teams may adjust the target velocity of ongoing iterations corresponding to the remaining time of iterations rather than extend the duration.

We again used the best performer (i.e. GBMs with SA+BoWs) in most cases to perform this experiment. Figure 3.13 shows NMAE achieved in predicting *velocity*

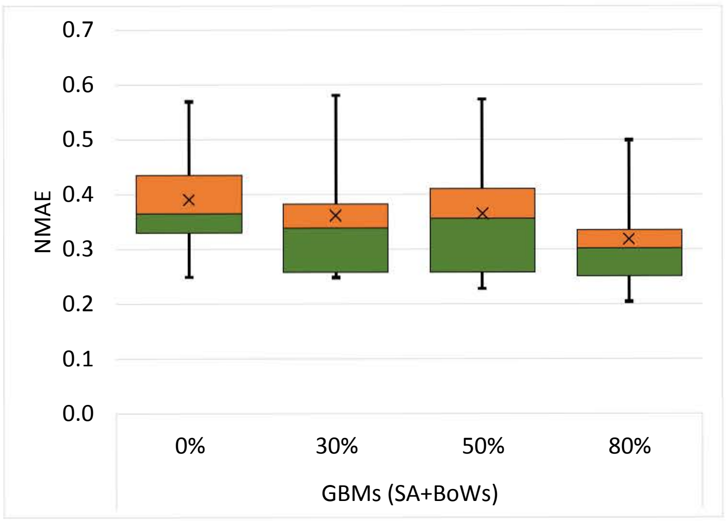


Figure 3.13: Evaluation results on varying prediction time. X is a mean of NMAE averaging across all projects (the lower the better).

(`Difference`) from the four different prediction times (i.e. at the beginning of the iteration, and when it progresses to 30%, 50%, and 80% of its planned duration) obtained from running GBMs using the combination of statistical aggregation and Bag-of-Words aggregation approaches averaging across all projects. **We observe that the predictive performance achieved from the predictions made at a later time in an iteration is better than those that the predictions made at the beginning of the iterations (0%)** – the prediction at 30%, 50%, and 80% of the iteration's duration achieves an average of 0.348 NMAE while the prediction at the beginning of the iteration achieves 0.390 NMAE, averaging across all projects. This confirms our earlier hypothesis that the latter we predict, the more accuracy we could gain since more information has become available. That phenomenon is however not consistently seen with the predictive performance of the prediction made at 50% of the iteration's duration being slightly lower than those that made at 30% of the iteration's duration. The prediction made at 80% of the iteration's duration does however achieve the highest predictive performance – it achieves 0.318 NMAE, averaging across all projects. For the sake of completeness, our experiments covered a range of prediction times from 0% to 80% to sufficiently test a hypothesis that the latter we predict, the more accuracy we could gain. We however acknowledge that making a prediction at 80% might be less useful in practice since the outcomes have become obvious and/or it might be too late to change the outcomes.

We investigated further to see when predictions could be made while not losing too much predictive power. To do so, we analyzed the improvement of the predictive performance between different prediction time intervals. We found that when we delayed making a prediction by 30%, we gained only 7-12% improvement in predictive perfor-

Table 3.13: Number of iterations in each class in each project

Project	Percentile		Class		
	33^{th}	66^{th}	Over	Achieved	Under
Apache	-11	0	105	135	108
JBoss	-4	0	47	233	92
JIRA	-3	0	218	1211	444
MongoDB	-2	0	244	273	248
Spring	0	20	162	162	152

mance. In fact, there was not much difference in terms of predictive power when making a prediction at the 30% or 50% marks. This result suggests that it is reasonably safe to make a prediction early, even at the beginning of an iteration.

Answer to RQ4: The time when a prediction is made affects the predictive performance, but only small improvement is gained when we delay making the prediction.

Classifying the outcomes of an iteration (RQ5)

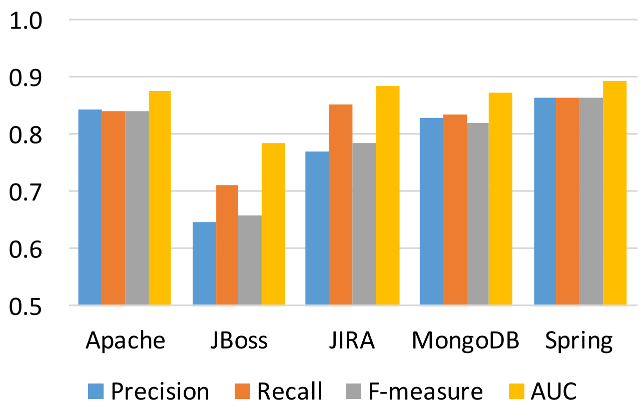


Figure 3.14: Evaluation results on predicting the outcomes of iterations in terms of precision, recall, F-measure, and AUC from each project (the higher the better)

We defined a tolerance margin to classify the outcomes of an iteration based on the statistical characteristics of each project. To maintain a relative balance between the classes, we used the 33^{th} and the 66^{th} percentile of $velocity(\texttt{Difference})$ as the tolerance margin: an iteration having $velocity(\texttt{Difference})$ below the 33^{th} percentile falls into *under-achieved* class, an iteration having $velocity(\texttt{Difference})$ above the 66^{th} percentile falls into *over-achieved* class, otherwise an iteration falls into *achieved* class. Table 3.13 shows the number of iterations in each class according to the 33^{th} and the 66^{th} percentile from each project. For example, in the Apache project, an iteration falls into *under-*

achieved class if $velocity(\texttt{Difference})$ is below -11, and falls into *over-achieved* class if $velocity(\texttt{Difference})$ is over 0.

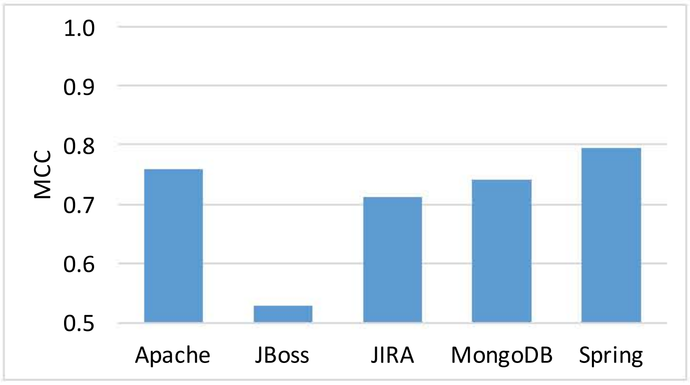


Figure 3.15: Matthews Correlation Coefficient (MCC) results from each project (the higher the better)

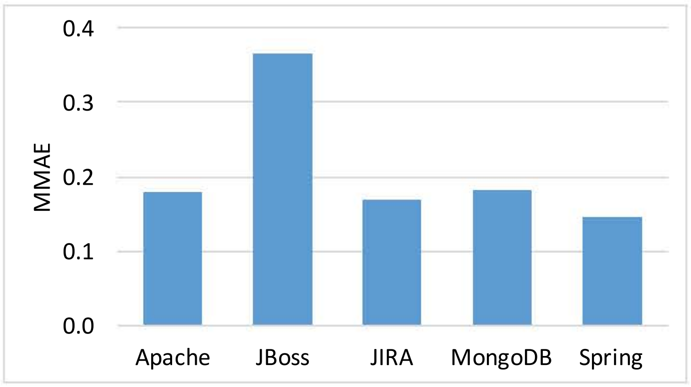


Figure 3.16: Macro-averaged Mean Absolute Error (MMAE) results (the lower the better)

In this experiment we also used GBMs with SA+BoWs and applied these margins to the predicted value of $velocity(\texttt{Difference})$. For example, in the Apache project, an iteration is predicted as *under-achieved* if the predicted value is below -11 (i.e. the 33^{th} percentile). Figure 3.14 shows the precision, recall, F-measure, and AUC achieved for each of five open source projects. **These evaluation results demonstrate the effectiveness of our predictive models in predicting the outcomes of iterations across the five projects**, achieving on average 0.79 precision, 0.82 recall, and 0.79 F-measure. The degree of discrimination achieved by our predictive models is also high, as reflected in the AUC results – the average of AUC across all projects is 0.86. The AUC quantifies the overall ability of the discrimination between classes. Our model performed best for the Spring project, achieving the highest precision (0.87), recall (0.85), F-measure (0.86), and AUC (0.90). The result from using MCC as a measure (Figure 3.15) also corresponds to the other measures. As can be seen from Figure 3.15, our approach achieved over 0.5

MCC in all cases – our approach achieves 0.71, averaging across all projects. MMAE is used to assess the performance of our models in terms of predicting ordered outcomes. As can be seen from Figure 3.16, our approach achieved 0.20 MMAE, averaging across all projects.

Answer to RQ5: Our predictive model is also highly accurate in classifying the outcomes of an iteration.

Important features

Table 3.14 reports the top-10 most important features and their weight obtained from running Random Forests with the combination of all aggregated features of issues. The weights here reflect the discriminating power of a feature since they are derived from the number of times the feature is selected (based on information gain) to split in a decision tree [164]. The weights are normalized in such a way that the most important feature has a weight of 1 and the least important feature has a weight of 0. **We observe that iteration features and statistical aggregation features are dominant in the top-10 list.** In many projects (e.g. Apache, JBoss, and MongoDB) iteration features such as the number of to-do issues, to-do velocity, and velocity at start time, are good predictors for foreseeing how an iteration will achieve against the target. It also corresponds to our results for finding the best combinations of aggregated features in RQ1. For example, in the JIRA and MongoDB projects, there are several aggregated features from statistical aggregation (SA) and Bag-of-Words aggregation (BoWs) that have high discriminating power since SA+BoWs performs best in those projects.

In addition, the changing of the other issue attributes (e.g. fix versions, description) are also in the top-10 in several projects (e.g. Apache and JBoss). In Spring, the graph-based aggregated features (e.g. sum of fan in/out) are good predictors. In the JBoss project, the features related to comments and number of team members are important for predicting the difference between the actual delivered velocity and the target velocity which may suggests that team collaboration is an important factor.

3.5.4 Implications and lessons learned

Results from our evaluations on five large open source projects suggest that our approach is highly reliable in predicting delivery capability at the iteration level. It allows project

Table 3.14: Top–10 most important features with their normalized weight

Apache		JBoss		JIRA	
To-do velocity	1.00	To-do velocity	1.00	To-do velocity	1.00
Velocity at start time	0.93	Velocity at start time	0.65	Cluster 5 (BoWs)	0.61
No. of issues at start time	0.53	No. of to-do issues	0.45	Cluster 53 (BoWs)	0.37
No. of to-do issues	0.50	No. of issues at start time	0.37	Cluster 93 (BoWs)	0.35
Type (Epic)	0.32	Priority (Critical)	0.27	Type (Story)	0.28
Changing of desc. (var)	0.23	Changing of fix versions (max)	0.19	Velocity at start time	0.26
Changing of desc. (mean)	0.20	Type (Feature Request)	0.16	Type (Major)	0.25
In-progress velocity	0.19	No. of comments (std)	0.15	Type (Improvement)	0.24
Type (Story)	0.19	No. of team mem.	0.14	No. of comments (std)	0.22
No. of team mem.	0.18	No. of fix versions (var)	0.14	Cluster 27 (BoWs)	0.20
MongoDB		**Spring**			
Velocity at start time	1.00	Type (Major)	1.00		
In-progress velocity	0.95	Done velocity	0.85		
To-do velocity	0.94	In-progress velocity	0.82		
No. of in-progress issues	0.71	No. of edges	0.76		
Added velocity	0.67	Sum of fan in	0.75		
No. of issues at start time	0.47	Sum of fan out	0.74		
Cluster 43 (BoWs)	0.46	No. of comments (std)	0.68		
Priority (Hard)	0.38	Cluster 56 (BoWs)	0.59		
Cluster 21 (BoWs)	0.36	Mean of fan in	0.58		
Done velocity	0.35	Added velocity	0.53		

managers and other decision makers to quickly foresee, at any given time during an ongoing iteration, if their team is at risk of not meeting the target deliverable set for this iteration. Predicting delivery capability early allows the team to deploy mitigation measures such as appropriate changes affecting the values of the top-5 predictors presented the previous section. Effective project management should also identify situations where an unexpected future event might present an opportunity to be exploited. Our approach supports iterative software development by forecasting whether the team is likely to deliver more than what has been planned for in an iteration. Knowing these opportunities early allows the team to plan for accommodating extra high-priority issues into the current iteration.

Story points are used to compute velocity, a measured use for the team's delivery capability per iteration. In practice, story points are however developed by a specific team based on the team's cumulative knowledge and biases, and thus may not be useful outside the team (e.g. in comparing performance across teams). Hence, the trained models are specific to teams and projects.

Our evaluation also demonstrates the high performance of the three ensemble methods used in building our prediction models. This suggests that ensemble methods such as Random Forests or Gradient Boosting Machines can be highly recommended for building software analytics models. Deep learning neural networks are only effective where there are significantly large amounts of data for training, which might not be the case for a range of software engineering problems.

One of the key novelties in our approach is deriving new features for an iteration from aggregating the features of its issues and their dependencies. These features can be derived by using a range of statistics over the issues' features or automatically learned using the bag-of-word approach. Our experimental results demonstrate the effectiveness of this approach and that they are complementary to each other. These results suggest that these feature aggregations techniques can be useful in other software analytics settings where features are located in different layers (similarly to iterations and issues).

In addition, there is well-established knowledge in machine learning that model accuracy depends critically on informative and comprehensive features extracted from data. It is also known that features are more useful when there are less redundancies. In our data, there are two main separate structures: the attributes associated within each issue and the dependency structure between issues. Our three feature sets are obtained through (a) aggregation of issue attributes, hence capturing the salient characteristics within a sprint, (b) building Bag-of-Words – which is essentially the well-known vector quantization technique – by finding distinct exemplars via k-means, hence reducing redundancies, and (b) exploiting graph characteristics, hence adding complementary information. Our experiments demonstrate that combining those three feature sets yields the best performance, thus confirming the prior knowledge. The increase in performance cannot be explained just by the increase in model complexity. This is because a more complex model will definitely fit the training data better, but it is more likely to hurt performance on test data due to the classic problem known as overfitting. If our goal is to derive a highly accurate model, then model complexity should not be a problem, as long as the model generalises better than simpler alternatives.

A major contribution of our work is demonstrating the utility of randomized methods to avoid the need for feature selection and reduction. While we acknowledge that a small set of independent features would be easy to understand, realistic problems are often complex enough to warrant a comprehensive feature set. The use of feature aggregation techniques has given us an extensive set of features to characterize a software development iteration. Although feature selection could be employed to filter out "weak" predictors, feature selection can be unstable: each selection method, running on different data samples, can produce a different subset of features. In modern machine learning techniques, feature correlation is no longer a crucial issue [139]. Randomized methods such as those used in this chapter need neither feature selection nor dimensionality reduction. This is because at each training step, only a small random subset of features is used – this also breaks the correlation between any feature pair since correlated features are much less likely to be in the same subset [165]–[167]. In addition, when making a prediction, a combination of many classifiers are used, each of which works on a smaller feature set.

3.5.5 Threats to validity

There are a number of threats to the validity of our study, which we discuss below.

Threats to construct validity: Construct validity concerns whether independent and dependent variables from which the hypothesized theory is constructed are relevant. We mitigated these threats by using real world data from iterations and issues recorded in several large open source projects. We collected iterations, all issues associated to these iterations, and all the relevant historical information available to ensure. The ground-truth (i.e. the difference between the actual delivered velocity and the target velocity) is based on the story points assigned to issues. Those story points were estimated by teams, and thus may contain human biases. However, story points are currently the best practices for measuring the delivery capability of a team, and are widely used in the industry. Hence, using story points makes our approach relevant to current industry practices.

Threats to conclusion validity: We tried to minimize threats to conclusion validity by carefully selecting unbiased error measures and applied a number of statistical tests to verify our assumptions [157] and following recent best practices in evaluating and comparing predictive models regarding effort estimation [157], [168]. In terms of predicting the three outcomes of an iteration (i.e. classification), our performance measures were also carefully designed against reporting bias towards majority classes. For example, while the F-measure is a balance between recall (often low for minority class) and precision (often high for minority class), we also employed the MCC performance measure which is insensitive to class imbalance. We used the MMAE performance measure for ordered classes. We however acknowledge that other techniques could also be used such as doing statistical undersampling, or artificially creating more samples for the undersampled class.

Threats to internal validity: Our great concern for threats to internal validity is data preprocessing. We found that around 30% of the issues across the five projects have not been assigned a story point (missing data). We filled those missing values using the mean of story points in each project. Our future work will investigate the use of other imputation techniques to handle such missing data. We also removed outliers (i.e. the difference velocity) and iterations involved with zero issue. We carefully processed the issue's change log to extract the features regard to a prediction time to prevent the leaking [128]. In addition, we also tried to avoid instability from applying additional data preprocessing algorithms (e.g. feature engineering) [43] by employing the ensemble randomized methods which overcome these problems (e.g. feature correlation, feature selection) [139].

Threats to external validity: We have considered almost 4,000 iterations and 60,000 issues from five large open source projects, which differ significantly in size, complexity, team of developers, and the size of community. All iteration and issue reports are real data that were generated during from the software development in open source settings. We however acknowledge that our data set may not be representative of all kinds of software projects, especially in commercial settings (although open source projects and commercial projects are similar in many aspects). Further investigation to confirm our findings for other open source and for closed source projects is needed.

3.6 Related work

Today's agile, dynamic and change-driven projects require different approaches to planning and estimating [125]. A number of studies have been dedicated to effort estimation in agile software development. Estimation techniques that rely on experts' subjective assessment are commonly used in practice, but they tend to suffer from the underestimation problem [30], [169]. Some recent approaches leverage machine learning techniques to support effort estimation for agile projects. The work in [170] developed an effort prediction model for iterative software development setting using regression models and neural networks. Differing from traditional effort estimation models (e.g. COCOMO [171], [172]), this model is built after each iteration (rather than at the end of a project) to estimate effort for the next iteration. The work in [173], [174] built a Bayesian network model for effort prediction in software projects which adhere to the agile Extreme Programming method. Their model however relies on several parameters (e.g. process effectiveness and process improvement) that require learning and extensive fine tuning.

Bayesian networks are also used [175] to model dependencies between different factors (e.g. sprint progress and sprint planning quality influence product quality) in a

Scrum-based software development project in order to detect problems in the project. Our work predicts not only the committed velocity against the actual achieved velocity but also the delivered against non-delivered velocity. More importantly, we use a comprehensive aggregation of features at both the iteration and issue levels, which represents the novelty of our work.

Several approaches have also been proposed to predict the resolving time of a bug or issue (e.g. [130], [176]–[179]). Those work however mainly focus on waterfall software development processes. This work, in contrast, aims to predict the quantum of achieved works at the level of iterations (and can be extended to releases as well) in agile software development. Here, we leverage feature/representation learning techniques which were not used in those previous work.

Graph-based characterization, one of the techniques we employed here, has also been used for building predictive models in software engineering. The study in [180] shows the impact of dependency network measures on post-release failures. The work in [138] built graphs representing source code, module and developer collaboration and used a number of metrics from those graphs to construct predictors for bug severity, frequently changed software parts and failure-prone releases. Similarly, Zimmermann and Nagappan [181] built dependency graphs for source code and used a number of graph-based measures to predict defects. Those approaches mostly work at the level of source code rather than at the software task and iteration level as in our work. Our recent work [132] proposed an approach to construct a network of software issues and used networked classification for predicting which issues are a delay risk. Those approaches however did not specifically look at the iterative and agile software development settings as done here in our work.

Our work is also related to the work on predicting and mining bug reports, for example, blocking bug prediction (e.g. [133]), re-opened bug prediction (e.g. [111], [113]), severity/priority prediction (e.g. [22], [23]), delays in the integration of a resolved issue to a release (e.g. [76]), bug triaging (e.g. [77], [182], [183]), duplicate bug detection ([184]–[189]), and defect prediction (e.g. [190], [191]).

3.7 Chapter summary

Iterative software development methodologies have quickly gained popularity and gradually become the mainstream approach for most software development. In this chapter, we have proposed a novel approach to delivery-related risk prediction in iterative development settings. Our approach is able to predict how much work gets done in the iteration. Our approach exploits both features at the iteration level and at the issue level. We also used a combination of three distinct techniques (statistical feature aggregation, bag-of-words feature learning, and graph-based measures) to derive a comprehensive set of features that best characterize an iteration. Our prediction models also leverage state-of-the-art machine learning randomized ensemble methods, which produce a strong predictive performance. An extensive evaluation of the technique on five large open source projects demonstrates that our predictive models outperform three common baseline methods in Normalized Mean Absolute Error and are highly accurate in predicting the outcome of an ongoing iteration. In the next chapter, we focus on the prediction at the issue level. We propose an approach to predict the delay of issues with due dates in Chapters 4 and 5.

Chapter 4

Delay prediction

ISSUE-TRACKING systems (e.g. JIRA) have increasingly been used in many software projects. An issue could represent a software bug, a new requirement or a user story, or even a project task. A deadline can be imposed on an issue by either explicitly assigning a due date to it, or implicitly assigning it to a release and having it inherit the release's deadline. This chapter presents a novel approach to providing automated support for project managers and other decision makers in predicting whether an issue is at risk of being delayed against its deadline. A set of features characterizing delayed issues are extracted and are selected to build predictive models to predict if the resolution of an issue will be at risk of being delayed. In addition, our predictive models are able to predict both the extend of the delay (e.g. impact) and the likelihood of the delay occurrence.

Delays constitute a major problem in software projects [192]. The studies in [1], [192], [193] have shown that ineffective risk management is one of the main reasons for the high rate of overrun software projects. An important aspect of risk management is the ability to predict, at any given stage in a project, which tasks (among hundreds to thousands tasks) are at risk of being delayed. Foreseeing such risks allows project managers to take prudent measures to assess and manage the risks, and consequently reduce the chance of their project being delayed. Making a reliable prediction of delays is therefore an important capability for project managers, especially when facing with the inherent dynamic nature of software projects (e.g. constant changes to software requirements). Current practices in software risk management however rely mostly on high-level, generic guidance (e.g. Boehm's "top 10 list of software risk items" [100] or SEI's risk management framework [194]) or highly subjective expert judgements.

Project managers need to ensure that issues are resolved in time against their respective due date. However, in practice project will never execute exactly as it was planned due to various reasons. One of the main challenges in project management is therefore predicting which tasks have a risk of being delayed, giving the current situation of a project, in order to come up with measures to reduce or mitigate such a risk. Hence, this approach aims to provide automated support for project managers and other decision makers in predicting whether an task is at risk of being delayed against its deadline. Making a reliable prediction of delays could allow them to come up with concrete measures to manage those issues.

We propose to analyze the historical data associated with a project (i.e. past issue reports and development/milestone/release plans) to predict whether a current issue is at risk of being delayed. Our approach mines the historical data associated with a project to extract past instances of delayed issues and cause factors. For example, a software developer being overloaded with the issues assigned to her, which may lead to that she may not complete some of those tasks in time. This knowledge allows us to extract a set of features characterizing delayed issues, which are then used to predict if an ongoing issue has a delay risk.

This chapter presents two main contributions:

- **Characterization of the issues that constitute a risk of delay.** We extracted a comprehensive set of *19* features (discussion time, elapsed time from when the issue is created until prediction time, elapsed time from prediction time until the deadline, issue's type, number of repetition tasks, percentage of delayed issues that a developer involved with, developer's workload, issue's priority, changing of priority, number of comments, number of fix versions, changing of fix versions, number of affect versions, number of issue link, number of blocked issues, number of blocking issues, topics of an issue's description, changing of description, and reporter reputation) from more than 60,000 issues collected from eight open source projects: Apache, Duraspace, Java.net, JBoss, JIRA, Moodle, Mulesoft, and WSO2. In addition, we performed two feature selection approaches – i.e. ℓ_1*-penalized logistic regression model* and *using p-value from logistic regression model* to select risk factors (i.e. features) with good discriminative power for each project.

- **Predictive models to predict which issues are a delay risk.**

  We developed accurate models that can predict whether an issue is at risk of being delayed. Our predictive models are able to predict both the impact (i.e. the degree of the delay) and the likelihood of a risk occurring. For example, given an issue X, our models are able to predict that there are (e.g.) 20% chance of X not posing a risk (e.g. causing no delay), (e.g.) 10% of being a minor risk (e.g. causing minor delay), (e.g.) 30% of being a medium risk (e.g. medium delay), and (e.g.) 40% a major risk (e.g. major delay). The performance of our predictive models were evaluated on eight different open source projects to ensure that they can be generalized. We achieved 79% precision, 61% recall, 68% F-measure, 83% Area Under the ROC Curve, and low error rates: 0.66 for Macro-averaged Mean Cost-Error and 0.72 for Macro-averaged Mean Absolute Error.

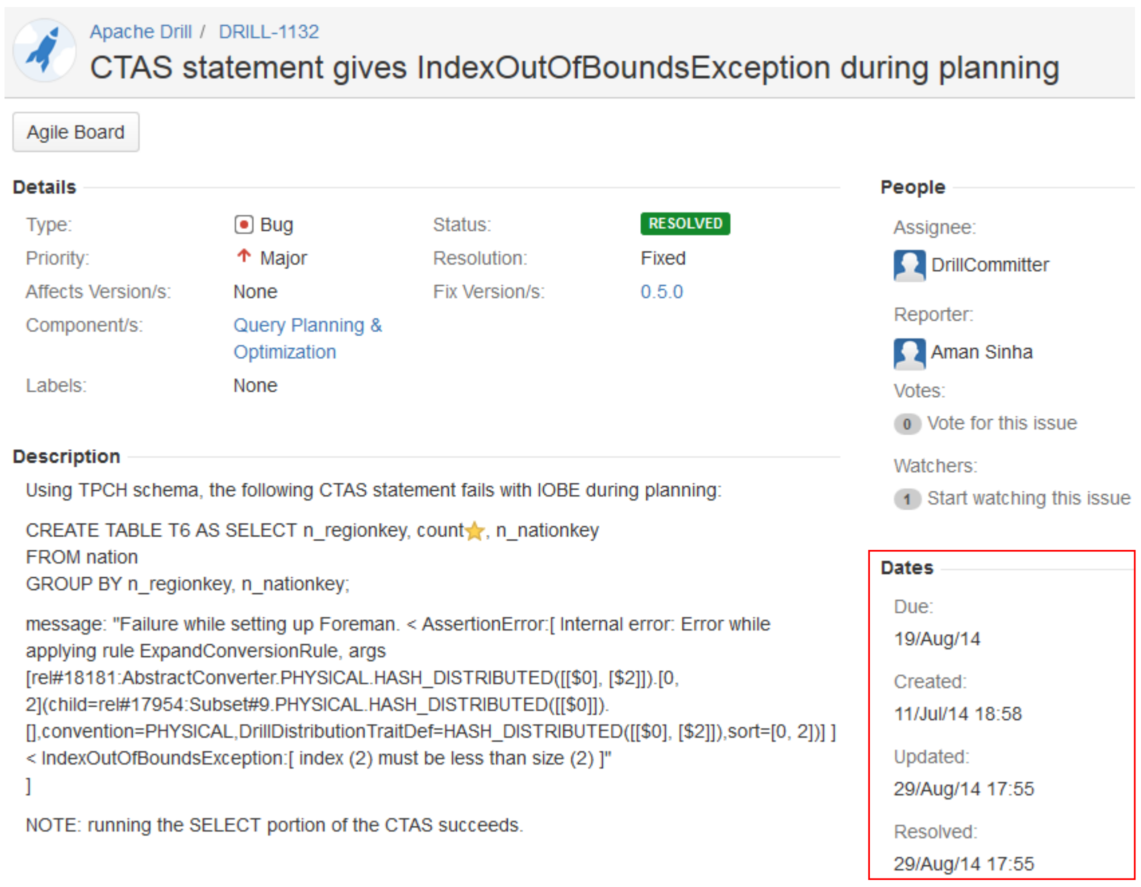
The remainder of this chapter is organized as follows. In Section 4.1 provides a conceptual framework of our approach. Section 4.2 presents a comprehensive set of features that are potentially associated with delayed issues. Section 4.3 describes how those features are selected to develop our predictive models (discussed in Section 4.4). We explain how we collect the data for our empirical study and evaluation in Section 4.5. Section 4.6 reports the experimental evaluations of our approach. Related work of this approach is discussed in Section 4.7 before we conclude in Section 4.8.

4.1 Approach

Typically, a (software) project requires a number of activities or tasks be completed. Each task usually has an estimated due date (e.g. deadline, project milestone). Currently, there are several project management supporting tools (e.g. JIRA software[a]). Those tools allow teams to collaborate among team members, plan their tasks, monitor their working progress, and to manage those tasks including define a due date (i.e. deadline) of a task. These supporting tools record a task in a form of an *issue*. Each issue usually has a planned due date, which can be set by either explicitly assigning a due date to it, or implicitly assigning it to a release and having it inherit the release's deadline. We refer to issues that were completed after their planned due date as *delayed issues* – as opposed to *non-delayed issues* which were resolved in time.

Figure 4.1 shows an example of an issue in the Apache project (recorded in JIRA software) which was assigned August, 19 2014 as due date. This issue is a delayed issue since it was resolved in August, 29 2014 – 10-day after its due date. The number of days that an issue are delayed also reflects the impact (e.g. severity) to software project, e.g.

[a]`https://www.atlassian.com/software/jira`



Apache Drill / DRILL-1132

CTAS statement gives IndexOutOfBoundsException during planning

Agile Board

Details

Type:	Bug	Status:	RESOLVED
Priority:	Major	Resolution:	Fixed
Affects Version/s:	None	Fix Version/s:	0.5.0
Component/s:	Query Planning & Optimization		
Labels:	None		

Description

Using TPCH schema, the following CTAS statement fails with IOBE during planning:

```
CREATE TABLE T6 AS SELECT n_regionkey, count(*), n_nationkey
FROM nation
GROUP BY n_regionkey, n_nationkey;
```

```
message: "Failure while setting up Foreman. < AssertionError:[ Internal error: Error while
applying rule ExpandConversionRule, args
[rel#18181:AbstractConverter.PHYSICAL.HASH_DISTRIBUTED([[$0], [$2]]).[0,
2](child=rel#17954:Subset#9.PHYSICAL.HASH_DISTRIBUTED([[$0]]).
[],convention=PHYSICAL,DrillDistributionTraitDef=HASH_DISTRIBUTED([[$0], [$2]]),sort=[0, 2])] ]
< IndexOutOfBoundsException:[ index (2) must be less than size (2) ]"
]
```

NOTE: running the SELECT portion of the CTAS succeeds.

People

Assignee: DrillCommitter

Reporter: Aman Sinha

Votes: 0 Vote for this issue

Watchers: 1 Start watching this issue

Dates

Due: 19/Aug/14

Created: 11/Jul/14 18:58

Updated: 29/Aug/14 17:55

Resolved: 29/Aug/14 17:55

Figure 4.1: An example of an issue assigned with a due date

major impact, minor impact. For example, a longer delay, the higher impact to a project. We consider delay as a risk and its impact is classified into a *risk class*.

Definition 3 (Risk class) *A risk class C presents an impact of delayed and non-delayed issues. A project can have more than one risk classes. For example, C_1 represents non-delay class, C_2 represents minor-delay class, C_3 represents medium-delay class, and C_4 represents major-delay class.*

A risk class is assigned to an issue based on the number of delay days. For example, the minor-delay class C_2 is assigned to issues that were delayed between 1 to 5 day(s), the medium-delay class C_3 is for 6 to 10 days delayed issues, and delayed-issues that were delayed more than 10 days are assigned to the major-delay class C_4, while non-delay issues have C_1 as their class.

The basic process of our approach is described in Figure 4.2. The process has two main phases: the learning phase and the execution phase. In the learning phase, let S_{train} be a set of historical issues that were resolved (i.e. issues having their resolved date recorded) and S_{test} be a set of ongoing issues that have not been resolved (i.e. no resolved

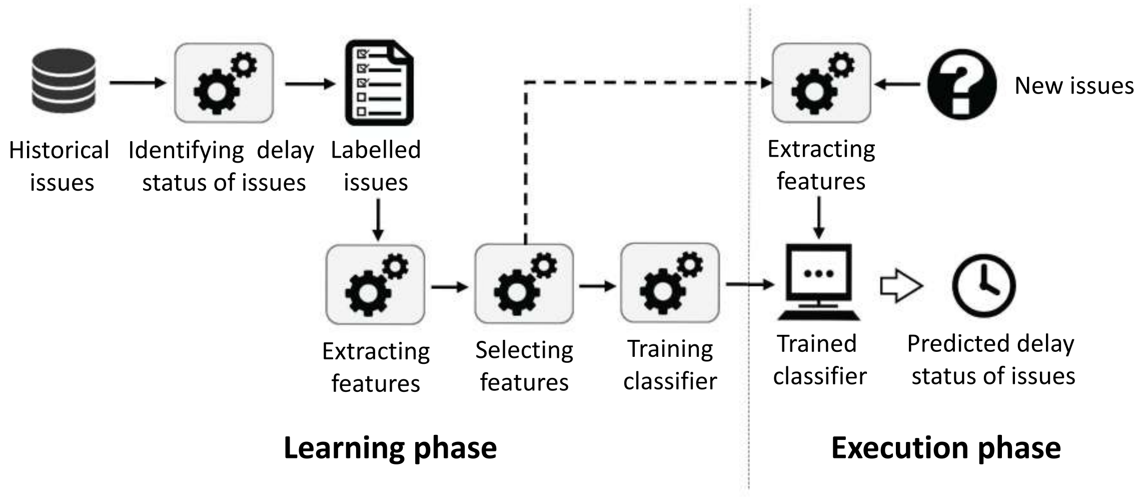


Figure 4.2: An overview of our approach

date recorded issues). The issues in S_{train} can be classified into one of the risk classes (e.g. C_1, C_2, C_3, and C_4) based on their delay status by comparing between their resolved date against their due date (i.e. data labelling). To characterized an issue, we extract a set of features from issues in S_{train} which can be considered as risk factors that could cause a delay. We then apply a feature selection technique to select a subset of features that provide a good predictive performance. For example, there are 5 features $[f_1, f_2, f_3, f_4, f_5]$ extracted from an issue. After applying a feature selection, only three features $[f_1, f_2, f_3]$ have been selected to build a predictive model since f_4 and f_5 are highly correlated features (e.g. these two features convey similar information). Our predictive model learns the characteristics of issues in S_{train} to classify an issue into a risk class. Our approach leverages classification techniques in machine learning (e.g. Random Forests) to build predictive models. The selected features are then extracted from the issues in S_{test} and fed into the trained classifier to make predictions in the execution phase.

4.2 Features of an issue

One of our objectives is to characterize risk factors that lead to a delayed issue. These factors form risk indicators (i.e. features) which are then used to predict if an issue will be delayed. In our model, our feature extraction process takes a prediction time (i.e. when a prediction is made) into account. We used the time when a prediction is made (*prediction time*) as the reference point when extracting the values of the features. During both training (i.e. learning) and testing (i.e. execution) phases, by processing an issue's change log, we collected the value which a feature had just *before* the prediction time. For example, if the final number of comments on an issue is 10, but there were no comments at the time when the prediction was made, then the value of this feature is 0. The details of issue's change log processing is provided in Section 4.5.

The underlying principle here is that, when making a prediction, we try to use as much information available at prediction time as possible. The historical information before the prediction time is used for training our predictive model, while the historical information after the prediction time is used for testing our model. Hence, our model is forward-looking analysis (predictive analytics), not retrospective analysis. The time when the prediction is made also has implications to its accuracy and usefulness. The later we predict, the more accuracy we could gain (since more information has become available) but the less useful it is (since the outcome may become obvious or it is too late to change the outcome). For example, if we assume prediction would be made after an issue has been discussed and assigned to a developer, then the features extracted and selected are historically relevant with respect to this prediction time. Doing this is to prevent information leakage in prediction [128]. We also explore when it is a good time to start predicting by examining three different prediction times.

We initially extracted a broad range of features related to an issue causing a delay, and then used feature selection techniques (see Section 4.3) to remove weak and redundant features. The following is the description of 19 features. We note that we studied on more than 60,000 issues collected from eight open source projects: Apache, Duraspace, Java.net, JBoss, JIRA, Moodle, Mulesoft, and WSO2 which we also provide statistical analysis results based on those collected issues to justify the extracted features. The details of data collection and extraction is provided in Section 4.5.

- **Discussion time**

  A software project can be viewed as a network of activities. Each activity is recorded as an issue whose details can be described and tracked in an issue tracking system. Hence, the time taken to complete a task (i.e. resolve an issue) contributes to the overall schedule of the project. More specifically, issues that take a significant, unusual amount of time to resolve may lead to delays. Discussion time is the period that a team spends on finding solutions to solve an issue. For example, the delayed issue MDL-38314 in version 2.5 of the Moodle project had 92 days in discussion. Figure 4.3 shows the distribution of the discussion time in each project. Teams tend to spend between 10 to 90 days for the discussion time.

- **Type**

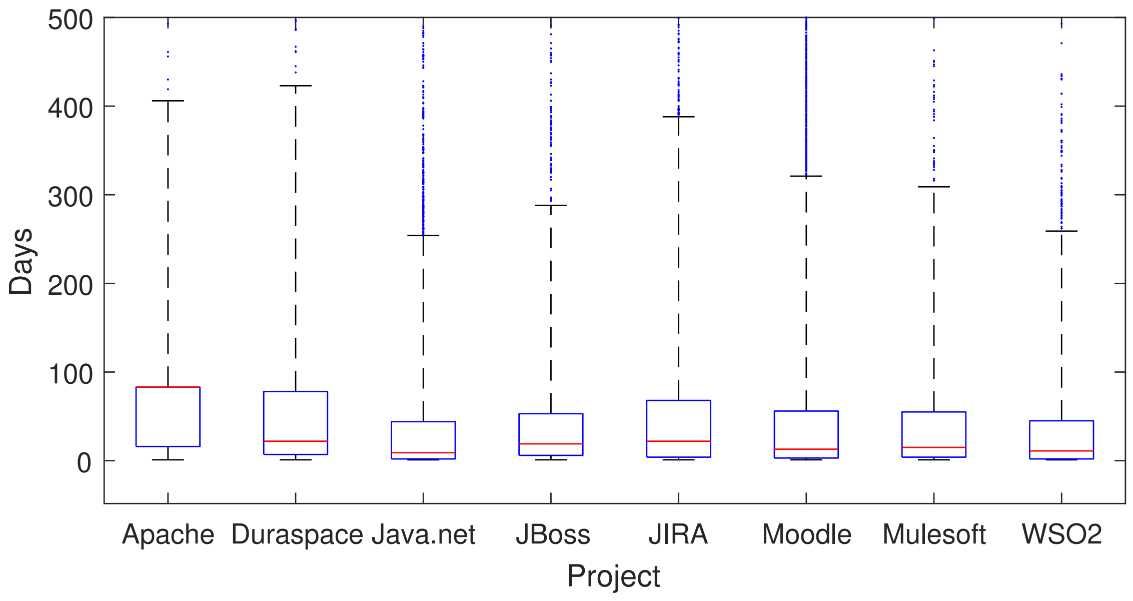


Figure 4.3: The distribution of the *discussion time* in each project

Each issue will be assigned a type (e.g. Task, Bug, New feature, Improvement, and Function Test) which indicates the nature of the task associated with resolving the issue (e.g. fixing a bug or implementing a new feature). Hence, we also consider issue type as a risk indicator. We note that some projects define some of their own issue types. For example, while there is no "Document" type for issues in the Moodle project, this type exists in the issues of JBoss and Duraspace. Thus, we considered only eight common types (Bug, Function Test, Improvement, New Feature, Story, Task, Sub-Task, and Suggestion) which 98.53% of issues have been assigned these types. Note that other types were set as *Null*.

- **Number of times that an issue is reopened**

Previous research, e.g., [111], [195], [113], [112], has shown that task repetitions (i.e. repetitions in the life-cycle of an issue) are considered as a degrading factor of the overall quality of software development projects. It often leads to additional and unnecessary rework that contributes to delays. An issue is reopened because of several reasons, e.g. if the problem has not actually been properly resolved, the issue needs to be reopened; or closed issues must be reopened since there are some errors found in the deployment phase after the issues were closed. We found that there are more than 20% of issues have been reopened. Especially, in the Apache project, 3,020 issues (48.02%) out of 6,289 were reopened at least one time.

- **Priority**

The issue's priority presents the order in which an issue should be attended with respect to other issues. For example, issues with blocker priority should be more concerned than issues with major or minor priority. Blocker priority is an issue that blocking other issues to be completed. In our study, we considered five priority levels: trivial, minor, major, critical, and blocker. 97% of the issues were assigned to one of these priority levels. We note that, in the WSO2 project, the level names are different (e.g. lowest, low, medium, high, highest), and thus were mapped to the five priority levels used in the other projects.

- **Changing of priority**

  The changing of an issue's priority may indicate the shifting of its complexity. In previous studies, Xia et al. showed that changing of issue's attributes (e.g. priority) could increase the bug fixing time, and delay the delivery of the software [134]. In addition, Valdivia et al. use the changing of priority as a feature to predict blocking bug [133]. In our context, we are particularly interested in the number of times an issue's priority was changed and considered it as a potential risk indicator.

- **Number of comments**

  Number of comments from developers during the prediction may be indicative of the degree of teams collaboration [196]. Panjer reported that a number of comments has an impact on the bug resolving time: bugs with two to six comments tend to be resolved faster than bugs with less than two comments and bugs with greater than six comments [130]. Note that we only consider the number of comments posted before the prediction time.

- **Number of fix versions**

  The "Fix Version" field on each issue indicates the release version(s) for which the issue was (or will be) fixed. Issues with a high number of fix versions need more attention in terms of developing, testing, and integrating. An intensive reviewing process is also required to validate that the resolving of an issue does not cause new problems for each fix version. We found that 84.29% of the issues were assigned at least one fix version.

- **Changing of fix versions**

  This feature reflects the number of times the fix versions associated with an issue have been changed (e.g. adding or remove some versions). The changing of the fix version(s) assigned to an issue may reflect some changes in the release planning and/or may be due to the nature of resolving the issue [134]. We thus consider the number of times the fix versions have been changed as a potential risk indicator.

- **Number of affect versions**

  The "Affect Version" field of an issue specifies versions in which it has been found. One issue can be found in many versions. For example, the issue MDL-48942 in the Moodle project has been found in version 2.7.5 and 2.8.2. It was planed to be fixed for versions 2.7.6 and 2.8.4. The number of affected versions is a potential risk indicator, e.g. more effort is needed to resolve an issue with a high number of affected versions. From our investigation, 77.72% of the issues were assigned with affected versions.

- **Number of issue links**

  Issue linking allows teams to create an association between issues. For example, an issue may duplicate another, or its resolution may depend on another issue. There are several type of issue links (e.g. relates to, duplicate, and block). Our previous work leverages the relationships among issues to predict delayed issues [132]. We consider all relations of issue link and use the number of those links as a risk indicator.

- **Number of issues that are blocked by this issue**

  Blocker is one of the issue linking types. This risk factor is the number of issues that are blocked by this issue. This type of issue dependency indicates the complexity of resolving issues since it directly affects the progress of other issues [131]. Thus, we deal with the blocker relationship separately.

- **Number of issues that block this issue**

  This risk factor is the number of other issues that are blocking this issue from being completed. The resolving of such an issue might take some time since all blocker issues need to be fixed beforehand. Thus, the number of blocker issues indicates the time allocated to solve an issue [133].

- **Topics of an issue's description**

  The description of an issue explains its nature and thus can be a good feature. To translate an issue's description into a feature, we have employed natural language processing techniques to extract the "topics" from such a description. These topics are used as a feature characterizing an issue. A topic here refers to a word distribution extracted from the description of an issue using Latent Dirichlet Allocation (LDA) [197]. LDA is used for finding topics by looking for independent word distributions within numerous documents (i.e. the description of issues). These documents are represented as word distributions (i.e. counts of words) to LDA. LDA usually attempts to discover a set of n topics, i.e. n word distributions that can describe the set of input documents. We used LDA to obtain a set of n topics in the form of a *topic-document matrix* for each issue. For example, the extracted topics representing the description of issue *MESOS-2652* from the Apache project is *"slider, mesos, application, app, support, container, executor, process, json, registry"*, where each *term* (e.g. slider) represents a generalized word in a set of issues' description. Note that the number of extracted topics is a parameter, and in this example was set to 10. We also performed a study on using different numbers of topics and the results are reported in Section 4.6.

- **Changing of description**

The description of an issue is important to all stakeholders of the issue. Changing the description of an issue indicates that the issue is not stable and may also create confusion and misunderstanding (and is thus a delay risk). Hence, we consider the number of times in which the issue's description was changed as a risk factor. We found that 13% of the issues have their description changed.

- **Reporter reputation**

Reporter reputation has been studied in previous work in mining bug reports, e.g., [111], [135], [176]. For example, Zimmermann et. al. found that bugs reported by low reputation people are less likely to be reopened [111]. Hooimeijer et al. used bug opener's reputation to predict whether a new bug report will receive immediate attention or not [135]. Bhattacharya et al. studied bug fix time prediction models using submitter's reputations [176]. In the context of predicting delayed issues, reporter reputation could be one of the risk factors since issue reporters with low reputation may write poor issue reports, which may result in a longer time to resolve the issue [195]. We use Hooimeijer's submitter reputation [135] as follows:

$$reputation(D) = \frac{|opened(D) \cap fixed(D)|}{|opened(D)| + 1}$$

The reputation of a reporter D is measured as the ratio of the number of issues that D has opened and fixed to the number of issues that D has opened plus one.

Figure 4.4 shows the distribution of the reporter reputation in each project. The mean of the reporter reputation in the Apache and Moodle projects are higher than that of the other projects (0.62-0.65), while the mean of the reporter reputation in WSO2 is the lowest. This indicates that reporters were not usually assigned to resolve issues in WSO2. In addition, Moodle has only 11.22% of issues were delayed. The JBoss project, in contrast, has the lower report reputation and a large number of delayed issues (53.88%) has been found.

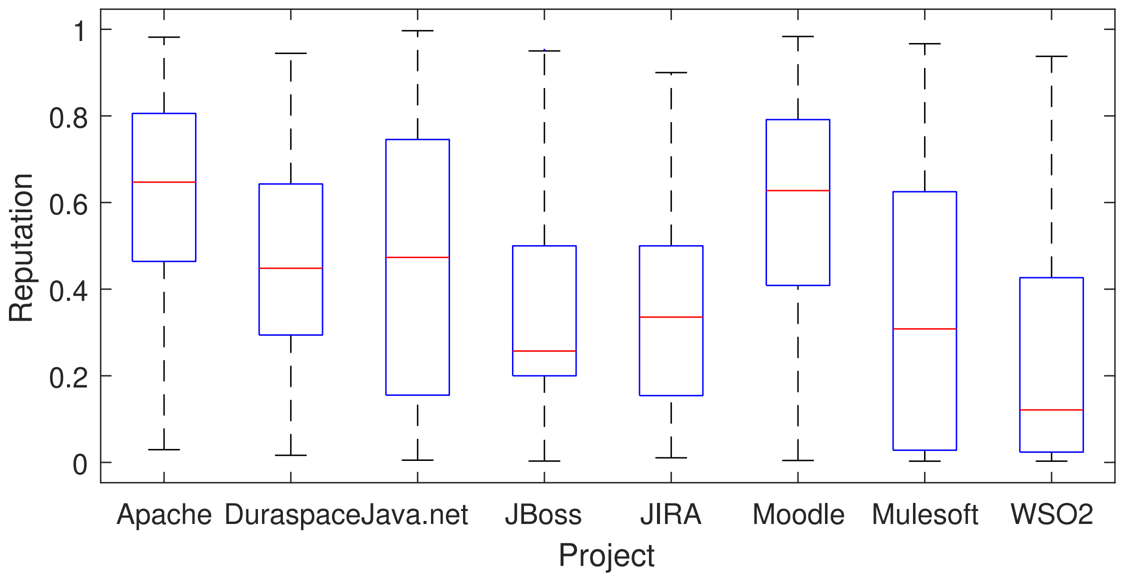


Figure 4.4: The distribution of the *reporter reputation* in each project

- **Developer's workload**

Developer workload reflects the quality of resource planning, which is crucial for project success. A lack of resource planning has implications to project failures [198], and developer workload may have significant impact on the progress of a project [101], [199]. A developer's workload is determined by the number of opened issues that have been assigned to the developer at a time. A developer's workload is (re-)computed immediately after the developer has been assigned an issue. Figure 4.5 shows the distribution of the developer's workload in each project. It can be noticed that a large number of delayed issues were found in the projects that have the high developer's workload (i.e. Apache and JBoss).

- **Percentage of delayed issues that a developer involved with**

Team members lacking specialized skills required by the project and inexperienced team members are amongst the major threats to schedule overruns [200]. Incompetent developers tend to not complete their tasks in time [101]. Boehm [100] also listed personnel shortfalls as one of the top-ten risks in software projects. On the other hand, recent research has shown that the best developers often produce the most bugs, since

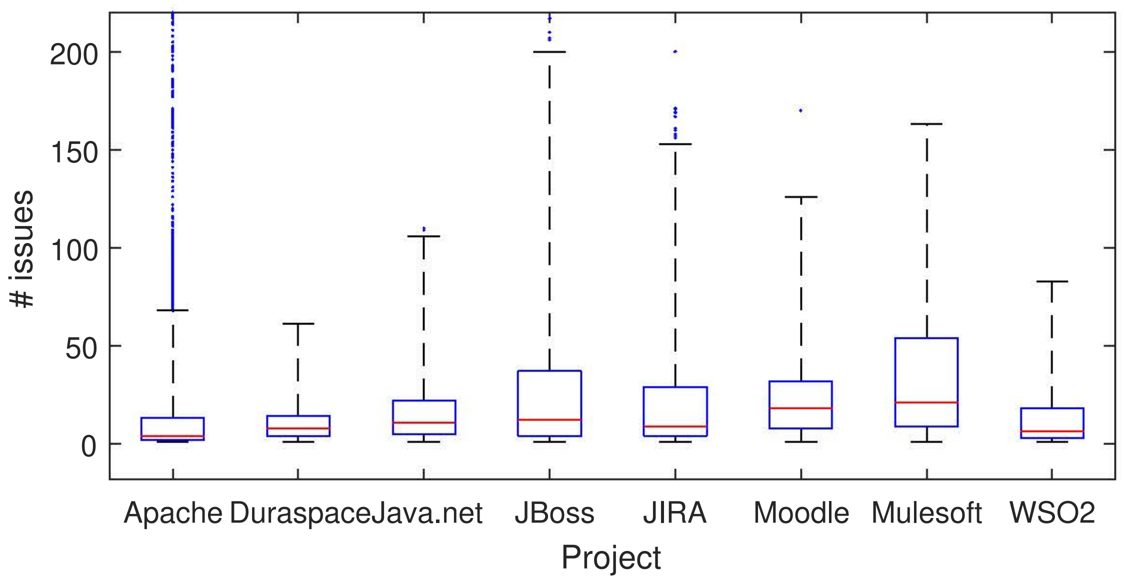


Figure 4.5: The distribution of the *developer's workload* in each project

they often choose or were given the most complex tasks [201]. This phenomenon might also hold for delayed issues: best developers may get the hardest issues and thus they take the longest time to complete it. A developer might have a large number of delayed issues simply because she is an expert developer who is always tasked with difficult issues.

We characterize this risk factor as the percentage of delayed issues in all of the issues which have been assigned to a developer. This metric is computed at the time when a developer has been assigned to solve an issue. For example, from the JBoss project, issues JBDS-188 and JBDS-1067 were assigned to the same developer but at different times. At that time when JBDS-188 had been assigned, the developer had 66% of delayed issues, while at a time of assigning the developer to JBDS-1067, the developer had 48% of delayed issues.

- **Elapsed time from when the issue is created until prediction time**

This factor is the number of days from when the issue is created until prediction time. For example, issue ID *DRILL-1171* in the Apache project was created on July 23, 2014. We assumed that the prediction was made at the time when the due date was assigned to the issue, i.e. August 14, 2014. Hence, the elapsed time between creation and prediction time is 22 days. Across all projects, the minimum, maximum, and median time between when a deadline is assigned to an issue and its creation are respectively 0, 1703, and 1 day with a standard deviation of 155.

- **Elapsed time from prediction time until the deadline**

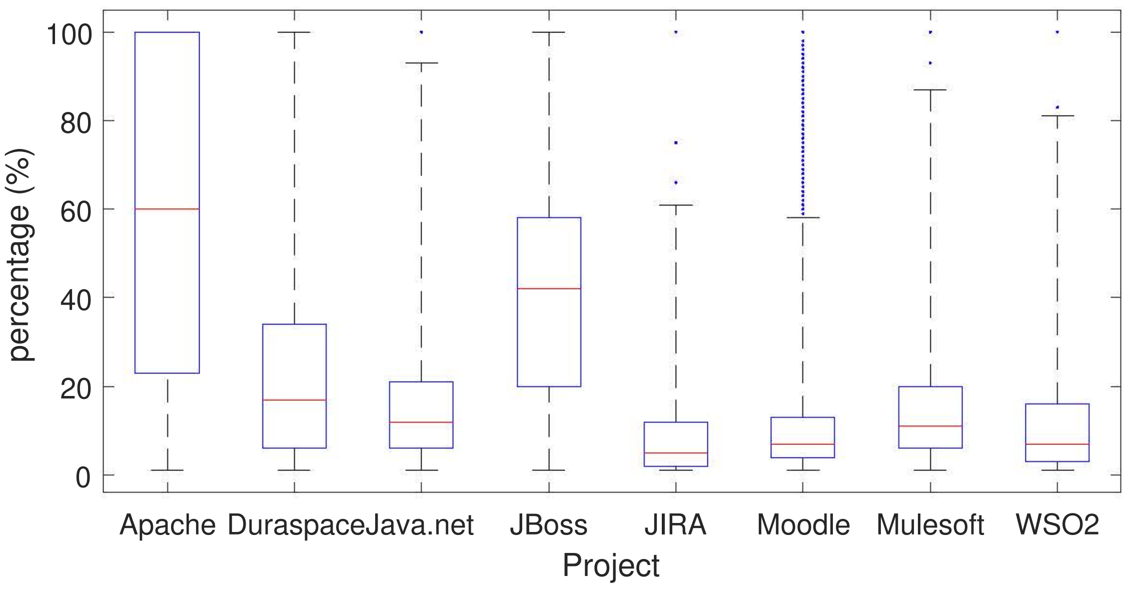


Figure 4.6: The distribution of the *percentage of delayed issues that a developer involved with* in each project

This factor is the number of days from prediction time until the due date which reflects the remaining time to resolve an issue before the deadline (not the actual time the issue was resolved). For example, August 21, 2014 is the deadline (i.e. due date) of issue ID *DRILL-1171*. If prediction time is on August 14, 2014, then the elapsed time from prediction time until the deadline is 7 days. Across all projects, the minimum, maximum, and median time between when a deadline is assigned to an issue and its resolution are respectively 1, 2049, and 31 days with a standard deviation of 176.

4.3 Feature selection

The next step of our approach involves selecting a compact subset of features (described in Section 4.2) that provide a good predictive performance. This process is known as *feature selection* in machine learning [202]. Sparse models, those models with a small number of features, are desirable since they are easier to interpret and acted upon by model users. They also enable the classifiers to train faster and critical to prevent over-fitting when training data is limited compared to the number of possible attributes. Over-fitting occurs when a model (i.e. classifier) learns the detail and noise in the training data or having too many features relative to the number of observations (i.e. issues). It negatively impacts the models ability to generalize [202].

In this section, we explore two distinct feature selection techniques: using *p-value from logistic regression model* and ℓ_1*-penalized logistic regression model.*

4.3.1 Using p-value from logistic regression model

Logistic regression model [203] can be used to predict binary outcomes (e.g. delayed and non-delayed classes). The model estimates the probability of an event occurring (e.g. will this issue be delayed?) under the presence of feature x as follows:

$$P(y = 1|x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1(x))}}$$

We built logistic regression models based on the features described in Section 4.2. This technique examines the contribution of each feature to the classification outcome independently. The coefficient β_1 indicates how strong the feature contributes to the delay. Its p-value measures the probability that the feature's contribution is due to random chance. A low p-value ($p < 0.05$) indicates that a feature is likely to associated with the outcome. A larger p-value, in contrast, suggests that the changes of a feature's value are not statistically associated with the changes of the delayed class. In our study, we select features with $p < 0.05$. However, assessing p-values of an individual feature ignores correlations among them which can cause multi-collinearity. Collinearity refers to an exact or approximate linear relationship between two or more features which it is difficult to obtain reliable estimates of their individual regression coefficients – i.e. two variables are highly correlated, they both convey essentially the same information. We thus filtered out features which exhibit multi-collinearity using the Belsley collinearity diagnostics [204] by selecting only one feature with the lowest p-value from those highly correlated features. The Belsley collinearity diagnostics [204] is the technique for assessing the strength and sources of collinearity among variables (i.e. features) in a multiple linear regression model which is a model of the form $Y = X\beta + \varepsilon$. X is a matrix of regression variables, and β is vector of regression coefficients. The output of this technique is Variance-Decomposition Proportions (VDP) which identifies groups of variates involved in near dependencies. VDP is determined from using the condition indices of a scaled matrix X which is the singular-value decomposition (SVD) [205] defined as a matrix S. From the singular-value decomposition of scaled matrix X with p columns (p equals to the number of selected features), let V be the matrix of orthonormal eigenvectors of $X'X$, and $S_{(1)} \geq S_{(2)} \geq ... \geq S_{(p)}$ be the ordered diagonal elements of the matrix S. Thus, the VDP is the proportion of the j^{th} term in the sum relative to the entire sum ($j = 1, ..., p$)

from the estimate of the i^{th} multiple linear regression coefficient, β_i:

$$V(i,1)^2/S^2_{(1)} + V(i,2)^2/S^2_{(2)} + ... + V(i,p)^2/S^2_{(p)}$$

where $V(i,j)$ denotes the $(i,j)^{th}$ element of V [204].

4.3.2 ℓ_1-penalized logistic regression model

While feature selection using p-value has been frequently used, it tends to overestimate the contribution of each feature in absence of other features. When all features are simultaneously considered, the individual contributions tend to be dampened due to correlation between features. Thus it might be better to assess the significance of all features at the same time. To that end, we developed a ℓ_1-penalized logistic regression model [206] for selecting features. The following is the detail of the ℓ_1-penalized logistic regression model.

Let $\{(x^i, y^i)\}_{i=1}^n$ be the training set, where $x \in \mathbb{R}^p$ be the feature vector and $y \in \pm 1$ be the binary outcome i.e., $y = 1$ if delay occurs and $y = -1$ otherwise. Let

$$f(x) = w_0 + \sum_{j=1}^{p} w_j x_j$$

where w_j is the feature weight (i.e. coefficient). We aim at minimizing the following penalized log-loss with respect to the weights:

$$L(w) = \frac{1}{n} \sum_{i=1}^{n} \log\left(1 + \exp\left(-y^i f(x^i)\right)\right) + \lambda \sum_{j} |w_j|$$

where $\lambda > 0$ is the penalty factor. The ℓ_1-penalty suppresses weak, redundant and irrelevant features by shrinking their weights toward zeros. It does so by creating competition among the features to contribute to the outcome. In our experiments, the penalty factor λ is estimated through cross-validation to maximize the Area Under the ROC Curve (AUC). To ensure weights are compatible in scale, we normalized features to the range [0-1]. The selected features were then used to build a predictive model which we describe in the next section.

4.4 Predictive model

Our predictive models are able to predict not only if an issue will be at risk of being delayed but also the degree of delay (in terms of the number of days overrun). To do so, we employ *multi-class* classification where the risk classes reflect the degree of delay. Since the number of delayed issues in our data set is small (compared to the number of non-delayed issues – see Table 4.2), we choose to use three risk classes: *major delayed*, *medium delayed*, and *minor delayed* (and the *non-delayed* class).

For each of our case study projects, the risk factors (i.e. features) that we selected are used to train a diverse set of seven classifiers: Random Forests (RF), Neural Networks (aNN), Decision Tree (C4.5), Naive Bayes (NB), NBTree, Deep Neural Networks with Dropouts (Deep Nets), and Gradient Boosting Machines (GBMs). Our predictive models are also able to provide the likelihood of a risk occurring, i.e. the chance of an issue causing no delay, minor delay, medium delay, and major delay. In the following subsection, we will describe this important aspect of our predictive models.

4.4.1 Predicting the likelihood of a risk occurring

Our objective is not only predicting risk classes but also estimating the class probabilities. Of the seven classifiers studied, Naive Bayes, Neural Networks, GBMs and Deep Nets with Dropouts naturally offer class probability. However, without appropriate smoothing, probability estimates from those two methods can be unreliable for small classes. Naive Bayes, for example, often push the probability toward one or zero due to its unrealistic assumption of conditional independence among risk factors. Neural networks and GBMs are implemented as nonlinear multiclass logistic regression, and thus the class probabilities are naturally produced. Decision-tree classifiers such as C4.5 and NBTree also generate probabilities which are class frequency assigned to the leave in the training data. However, these estimates are not accurate since leave-based probabilities tend to be pushed towards zero and one. In general, Naive Bayes, Neural networks and decision-tree methods require careful calibration to obtain class probabilities [207]. Random Forests, on the hand, rely on voting of trees, thus the probabilities can be estimated by proportions of votes for each class. With a sufficient number of trees, the estimates can be reliable. The process of probability calibration for all classifiers are discussed as follows.

Estimating probability in Random Forests

Random Forests generate many classification trees from random sampling of features and data. Thus with sufficient number of trees, the class probability can be estimated through voting. For example, in our context, assume that there are 100 trees generated from the data. An issue is predicted as major delayed because there are 60 trees that predict so, while only 25, 10 and 5 trees predict minor delayed, medium delayed, and non-delayed respectively. Thus, the voting result can be treated as the probability distribution, which means, the probability of the issue to be major delayed is 60%, 25% for minor delayed, 10% for medium delayed, and 5% for non-delayed.

Probabilistic decision trees

The decision tree probabilities naturally come from the frequencies at the leaves. Generally, the probability using frequency-based estimate at a decision tree leaf for a class y is:

$$P(y|x) = \frac{tp}{(tp+fp)}$$

Where tp is true-positive of class y, and fp is false-positive of class y [208].

For example, assume that y is the major delayed class. The probability of the issue X to be major delayed is the fraction between tp and $tp+fp$ of the major delayed class, where tp is the number of issues that are classified as major delayed and they are truly major delayed, and fp is the number of issues that are classified as major delayed when they are not major delayed.

Naive Bayes

Typically, Naive Bayes is a probability classifier model. Thus, we followed Bayes's theorem to determine a class probability. Given a class variable y (i.e., major, medium, minor, and non delayed risk) and a dependent feature vector x_1 through x_n which are our features in Section 4.2, the probability of class y is:

$$P(y|x_1,...,x_n) = \frac{P(y)P(x_1,...,x_n|y)}{P(x_1,...,x_n)}$$

Then, the instances are classified using Bayes's decision rule [209].

Combining decision tree and Naive Bayes

As mentioned earlier, NBTree is a hybrid of decision tree (C4.5) and Naive Bayes. The decision tree nodes contain univariate splits as regular decision trees, but the leaf nodes accommodate with Naive Bayes classifiers [52]. The leaf nodes thus give the probability estimation using Naive Bayes. For example, at one leave node in the decision tree, the probability distribution of each class is determined using Naive Bayes on the population that classified to that node [210].

4.4.2 Risk exposure prediction

Our predictive models are able to predict the exposure of a risk. Risk exposure is traditionally defined as the probability of a risk occurring times the loss if that risk occurs [211]. Risk exposure supports project managers to establish risk priorities [100]. Our predicted risk exposure $\bar{RE}$ is computed as follows.

For an issue i, let C_1, C_2, C_3, and C_4 be the costs associated with the issue causing no delay, minor delay, medium delay, and major delay respectively. Note that C_1 is generally 0 – no delay means no cost. The predicted risk exposure for issue i is:

$$\bar{RE}_i = C_1P(i,Non) + C_2P(i,Min) + C_3P(i,Med) + C_4P(i,Maj)$$

where $P(i,Non)$, $P(i,Min)$, $P(i,Med)$, and $P(i,Maj)$ are the probabilities of issue i being classified in non-delayed, minor delayed, medium delayed and major delayed classes respectively.

Note that all the costs, C_1, C_2, C_3 and C_4, are user defined and specific to a project. For example, assume that $C_1 = 0$, $C_2 = 1$, $C_3 = 2$, and $C_4 = 3$, and there is 30% chance that an issue causing no delay (i.e. $P(i,Non) = 0.3$), 40% chance causing minor delay (i.e. $P(i,Min) = 0.4$), 15% chance causing medium delay (i.e. $P(i,Med) = 0.15$), and 15% major delay (i.e. $P(i,Maj) = 0.15$), then the predicted risk exposure $\bar{RE}_i$ of the issue is 1.

4.5 Dataset

In this section, we describe how data were collected for our study and experiments.

4.5.1 Data collecting

We collected data (past issues) from eight open source projects: Apache, Duraspace, Java.net, JBoss, JIRA, Moodle, Mulesoft, and WSO2:

- Apache is a web server originally designed for Unix environments. The Apache web server then has been ported to Windows and other network operating systems. Currently, there are more than fifty sub-projects under Apache community umbrella (i.e. Hadoop, Apache CouchDB, and Apache Spark). According to the general structure of an Open-Source Software (OSS) community, the project's contributors could be both core members and peripheral members. There are more than three hundred core members and ten-thousand peripheral members contribute to the project [212]. Anyone with an interested in working on Apache development could join the Apache's issue tracking system.[b] There are more than three hundred core members and ten-thousand peripheral members contributing to the project [212]. All issues in Apache were recorded in Apache's issue tracking system.[c]
- The Duraspace project[d] supports the web platform digital asset management that contains several sub-projects in their repository i.e. VIVO, Islandora, Hydra Hypatia, and DuraCloud. There are about two-hundred contributors including reporters, developers, testers, and reviewers working for the Duraspace community and more than eighteen hundred institutions use and develop their open source software in partnership with DuraSpace. All issues in the Duraspace project were recorded in Duraspace's issue tracking system.[e]
- Java.net[f] is a project related to Java technology. There are more than two-hundred sub-projects under Java.net community umbrella (i.e. Glassfish, Jersey, Java Server Faces). Currently, this community is moderated Oracle Corporation since 2010 (the

[b] `https://issues.apache.org/jira`
[c] `https://issues.apache.org/jira`
[d] `http://duraspace.org`
[e] `https://jira.duraspace.org`
[f] `https://java.net/jira/`

acquisition of Sun Microsystems by Oracle Corporation was completed on January 27, 2010).[g]

- JBoss[h] is an application server program which supports the general enterprise software development framework. The JBoss community has been developing more than two-hundred sub-projects (e.g. JBoss Web Server, JBoss Data Grid, JBoss Developer Studio) with more than one-thousand contributors. The JBoss community records issues in JBoss's issue tracking system.[i]
- JIRA[j] is a project and issue tracking system provided by Atlassian. They also use JIRA for tracking their development progress. Note that the JIRA repository includes the demonstration projects for their customers which we did not include into our datasets.
- Moodle is an e-learning platform that allows everyone to join the community in several roles such as user, developer, tester, and QA. The Moodle tracker[k] is Moodle's issue tracking system which is used to collect issues and working items, and keep track issues status in development activities.
- The Mulesoft project[l] is a software development and platform collaboration tool. There are more than thirty sub-projects hosted by Mulesoft e.g. Mule Studio and API Designer. Their solutions focus on several industries (e.g. financial services, health care, education). Mulesoft uses JIRA as their issue tracker.[m]
- WSO2 is an open source application development software company which focuses on providing middleware solutions for enterprises. There are more than twenty sub-projects provided by WSO2 (e.g. API management, smart analytics) and their development progress are recorded in the WSO2's repository.[n]

Those eight projects have different sizes, number of contributors, and development processes. The reasons behind this variety of selections are to demonstrate that our approach is generalized for various software project's context and achieving a comprehensive evaluation.

[g]https://www.oracle.com/sun/index.html
[h]https://issues.jboss.org
[i]http://www.jboss.org/
[j]https://jira.atlassian.com
[k]https://tracker.moodle.org
[l]https://www.mulesoft.com/
[m]https://www.mulesoft.org/jira
[n]https://wso2.org/jira/

All the eight projects use JIRA[o], a well-known issue and project tracking software that allows teams to plan, collaborate, monitor and organize issues. We used the Representational State Transfer (REST) API provided by JIRA to query and collected past issue reports in JavaScript Object Notation (JSON) format. There are three main reports (i.e. three JSON files) which can be collected from three different APIs. Each of them contains different information of an issue. First, a primitive attribute report provides basic elements of an issue, for example issue's type, issue's priority, and issue's current status (e.g. in-progress, fixed). Figure 4.7 shows an example of the report in JSON format of issue *AURORA-1563* where due date can be identified from this report. Second, a change log report records all changes occurred on an issue, for example changing of issue's description, changing of issue's status. Figure 4.8 shows an example of a change log report of issue *AURORA-1563* which records that the status of this issue has been changed from *Reviewable* to *Resolved* on January 27, 2016. Several features discussed in Section 4.2 can be extracted from processing issue's change log, for example discussion time and changing of description. Third, a report of comments record all comments that team members having discussions on an issue. From processing a list comments, we can extract the number of comments at prediction time. Figure 4.9 shows an example of issue' comments of issue *AURORA-1563* where comment creation time is recored.

```
{"key": "AURORA-1563", ...
        "fields": {
        "issuetype": { "name": "Story", "subtask": false},
        "project": { "name": "Aurora", ...},
        "fixVersions": [{ "released": true,
                "releaseDate": "2016-02-07"}],
        "resolution": {"name": "Fixed"},
        "priority": {"name": "Major",}
        "duedate": {"2016-02-07"}
},...,}
```

Figure 4.7: Example of an issue JSON file *AURORA-1563*

Table 4.1 shows the number of collected issues. We collected 108,676 closed issues from eight open source projects: Apache, Duraspace, Java.net, JBoss, JIRA, Moodle, Mulesoft, and WSO2. For Apache, Java.net, JBoss, and Mulesoft, more than fifty-thousand issues opened from *January 01, 2004* to *March 19, 2016* were collected from their issue tracking system. For Duraspace and WSO2, the resolved or closed issues between *November 9, 2006* to *February 03, 2016* were collected. For Moodle, more than

[o]`https://www.atlassian.com/software/jira`

```
{..."key": "AURORA-1563",
        "changelog": { "histories": [...
                {"id": "15228187",
                 "created": "2016-01-27T20:18:56.902+0000",
                 "items": [{
                 "field": "status",
                 "fieldtype": "jira",
                 "from": "10006",
                 "fromString":"Reviewable",
                 "to": "5",
                 "toString": "Resolved"}
]},...]}}
```

Figure 4.8: Example of an issue's change log in JSON format of issue *AURORA-1563*

twenty-thousand issues opened from *January 01, 2004* to *March 14, 2016* were collected from the Moodle tracker. For Apache, the data collection went through two steps. First, we randomly selected 10-20% issues from a small number of projects in Apache including Hadoop Common, HBase, and IVY. We studied these issues to understand how to extract the due date and other issue attributes, how to process and extract information from issue change logs, how to identify customized attributes, etc. These issues form the first part of the Apache dataset we collected. After this initial study, we performed an exhaustive search across all Apache projects to collect all issues with the "due date" field explicitly recorded. These issues are from 304 projects in Apache and form the remaining part of our Apache dataset. A similar process was also applied to collecting data from other projects.

Table 4.1: Collected issues

Project	# collected issues	# issues with due date	% of issues with due date
Apache	10,553	6,409	60.73
Duraspace	6,509	3,768	57.89
Java.net	20,918	16,609	79.40
JBoss	10,215	4,009	39.25
JIRA	12,287	4,478	36.45
Moodle	26,642	26,030	97.70
Mulesoft	12,237	12,016	98.19
WSO2	9,315	5,346	57.39
Total	108,676	78,665	72.38

```
{"key": "AURORA-1563", ...
        "comment": {"startAt": 0, "maxResults": 2, "total": 2,
        "comments": [
        {...
         "displayName": "John Sirois",
         "active": true},
         "body": "https://reviews.apache.org/r/42816/",
         "created": "2016-01-26T19:09:11.471+0000",
         "updated": "2016-01-26T19:09:11.471+0000"},
        {
         "displayName": "John Sirois",
         "active": true},
         "body": "Deprecation removals in master @ http://git-wip-us...",
         "created": "2016-01-27T20:18:56.890+0000",
         "updated": "2016-01-27T20:18:56.890+0000"}]
}
```

Figure 4.9: Example of an issue JSON file *AURORA-1563*

4.5.2 Data preprocessing

In data preprocessing, we first identified delayed issues from the collected dataset and performed data cleansing process. In order to determine if an issue was delayed or not, we need to identify the due date assigned to the issue. We have however found that different projects have different practices in assigning a deadline to an issue, and thus we needed to be carefully in extracting the information. In some projects (e.g. Apache and JBoss), the issue due dates can be extracted directly from the "due date" attribute of an issue. Figure 4.1 shows an example of an issue in the Apache project (recorded in JIRA) which was assigned August, 19 2014 as due date. For issues that have a due date in their record, the delay was determined by checking the resolved date against the due date – i.e. a delayed issue is an issue has been resolved after a due date.

In some projects (e.g. Moodle and Mulesoft), this information is recorded in the "Fix Release Date" attribute of an issue (see `https://tracker.moodle.org/browse/MDL-12531` for an example), and thus was extracted from this attribute. In other projects (e.g. Duraspace, Java.net, JIRA, and WSO2), the due date is not directly recorded in an issue. For these projects, we infer the issue due date from the fix version(s) assigned to it – which can be extracted from the "Fix Version/s" field of an issue (see `https://java.net/jira/browse/GLASSFISH-13157` for an example). Note that when an issue is open, the "Fix Version/s" field conveys a target; and when an issue is closed, the "Fix Version/s" field conveys the version that the issue was fixed in. Hence, we needed to process the change log of an issue to extract the fix version assigned to the issue before our prediction time (when we predict if the issue was delayed or not). We then extracted the due date of the fix version (see `https://java.net/jira/browse/GLASSFISH/fixforversion/11015/` for an example), and used it as the due date for the issue. In the case when there are multiple fix versions assigned to an issue, we chose the one with the earliest due date. Note that we selected only the fix version that had been entered *before* the issues was resolved. Issues, which did not have any target release assigned before they were resolved, were not included in our dataset. We then identified delayed issues by comparing a due date with actual resolved date of the issues. We collected both *delayed* and *non-delayed* issues. For delayed issues, we also collected how many days of delay the issues were delayed.

We found that 72.38% of those issues whose due date can be extracted. The issues without due date were removed from our dataset. For example, in the Apache project, there were 25 issues created on January 01, 2004, but only two of them (i.e. *DIR-1* and *DIRSERVER-10*) were selected since they were the only two issues having a due date. Furthermore, we have found that some issues have very long period of delays (e.g. in some cases over 1,000 days). Such issues could have been left behind (e.g. no activity recorded in the issue report), and were eventually closed. Following the best practices [213] in identifying those outlier cases in each project, we used the actual mean resolving time of all (delayed and non-delayed) issues in the project plus its standard deviation as a threshold for each project. Issues which were delayed longer than the threshold were removed from our dataset. Overall, 13,918 issues identified as outliers were filtered out across the eight projects used in our study.

In total, we performed the study on 64,747 issues from the eight projects, which consist of 13,825 (21.35%) delayed and 50,922 (78.65%) non-delayed issues. Table 4.2 summarizes the characteristics of the eight projects on the delay time in terms of the minimum, maximum, median, mean, median, mode, and standard deviations of days late.

4.6 Evaluation

We empirically studied eight open source projects: Apache, Duraspace, Java.net, JBoss, JIRA, Moodle, Mulesoft, and WSO2 to build predictive models. In this section, we describes our experimental setting, the performance measures, applying feature selection

Table 4.2: Descriptive statistics of the delay time of the projects in our datasets

Project	# issues	# non-delay	# delay	min	max	mean	median	mode	std
Apache	6,289	3,683	2,606	1	1,169	132.27	21	9	218.97
Duraspace	3,676	2,706	970	1	335	64.89	31	3	73.56
Java.net	16,326	13,712	2,614	1	280	41.40	12	1	61.93
JBoss	3,526	1,626	1,900	1	410	87.57	27	1	120.76
JIRA	4,428	3,170	1,258	1	533	52.70	40	40	78.58
Moodle	17,004	15,095	1,909	1	503	70.74	39	1	94.23
Mulesoft	8,269	6,941	1,328	1	301	57.48	39	1	62.37
WSO2	5,229	3,989	1,240	1	258	43.84	18	1	55.12
Total	64,747	50,922	13,825						

# issues: number of issues, # non-delay: number of non-delayed issues,
# delay: number of delays issues, min: minimum of days late,
max: maximum of days late, mean: mean of days late,
median: median of days late, mode: mode of days late,
std: standard deviation of days late

techniques, and the evaluation results. Our evaluation aims to answer the following research questions:

RQ1 *How does our approach perform on different projects?*

We evaluate the predictive performance of our approach on eight projects: Apache, Duraspace, Java.net, JBoss, JIRA, Moodle, Mulesoft, WSO2. We also combine the issues from all projects together (called "All together") to evaluate our approach on a large, diverse number of issues.

RQ2 *Which classification techniques perform best?*

We employ seven well-known classifiers: Random Forests (RF), Neural Network (aNN), Decision Tree (C4.5), Naive Bayes (NB), NBTree, Deep Neural Networks with Dropouts (Deep Nets), and Gradient Boosting Machines (GBMs) to build the predictive models and compare the evaluation results to find the best performing classifier.

RQ3 *Which feature selection techniques perform best?*

We perform the experiments to compare the performance of different feature selection approaches: ℓ_1-*penalized logistic regression model* and using *p-value from logistic regression model*. We also discuss the outputs from applying the two techniques (e.g. the selected features) in Section 4.6.2.

RQ4 *How does the prediction time affect the predictive performance?*

We evaluate the predictive performance from the different prediction times to confirm our hypothesis that the later we predict, the more accuracy we gain. To do so, we perform the experiments to compare the performance between three prediction times: at the end of discussion time, at a time when a deadline (e.g. due date) was assigned to an issue, and at the creation time of an issue. There is very limited known information immediately after an issue was created. Note that our evaluation results in RQ1-RQ3 were obtained from the predictions made at the end of discussion time (see Section 4.2).

RQ5 *How does the number of topics affect the predictive performance?*

Applying topic modelling requires us to specify the number of extracted topics (i.e. parameter). We want to evaluate whether the number of topics affect the predictive performance by performing experiments to compare the performance of predictive models using different numbers of topics (i.e. 10, 100, 200, 300, and 400 topics). Note that we used 10 topics for the experiments in RQ1-RQ4.

RQ6 *How does our approach perform in terms of predicting risk exposure?*

Our predictive models are able to predict not only the impact (i.e. the risk classes), but also the likelihood of a risk occurring (i.e. the class probabilities). To evaluate the predictive performance in terms of estimating the probabilities, we employ Macro-averaged Mean Cost-Error (MMCE) and Macro-averaged Mean Absolute Error (MMAE) as our measurement to assess the error of the predicted probabilities. The details of MMCE and MMAE are discussed in Section 4.6.3.

RQ7 *How does the size of training data affect the predictive performance?*

To answer this question, we perform the experiments to compare the performance from two different sliding window settings. In the first setting, the predictive model learn from all issues in the past. The size of the training set thus increases over time (i.e. accumulate all past issues). The second setting, in contrast, uses only recent issues (e.g. one year past) for training. These two settings allow us to investigate several different size of training and test sets. The details of the sliding window settings are discussed in Section 4.6.1.

4.6.1 Experimental setting

We evaluated our predictive model using two different experimental settings which we try to mimic a real deployment scenario that prediction on a current issue is made using knowledge from the past issues. The first setting is the *traditional training/test splitting*. The second setting is *based on sliding window*. In the first setting, all issues collected in each of the eight case studies (see Section 4.5.1) were divided into a training set and a test set. The issues in training set are those that were opened before the issues in test set. This temporal constraint ensures that our models only learn from historical data and are tested from "future" data.

The sliding window setting is an extension to the first setting where the issue reports are first sorted based on the time they are opened. Then, we divide the issue reports into multiple sliding windows, and for each sliding window, we use data from the previous sliding windows to train a model [179]. Figure 4.16 illustrates an example of sliding window approach which the issues opened and resolved between year 2004 and 2006 have been learned to predict delayed issues in 2007, while predicting delayed issues in 2008 are based on the knowledge from the past windows (i.e. the issues opened and resolved between year 2004 and 2007). We also performed an experiment on the continuous sliding window setting which the issues from the only one previous window have been learned to predict delayed issues in the later window. For example, the issues opened and resolved in year 2006 have been learned to predict delayed issues in year 2007, and the issues opened and resolved in year 2007 have been learned to predict delayed issues in year 2008.

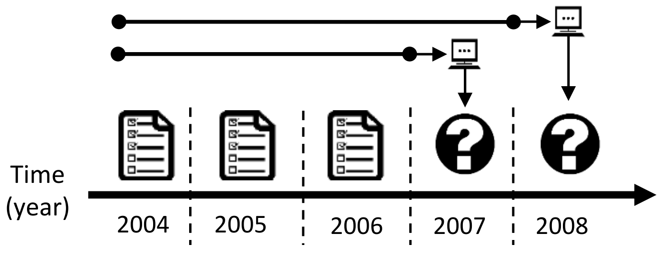


Figure 4.10: Sliding windows

Table 4.3 shows the number of issues in training set and test set for each project. During the training phase for our dataset, we use a threshold to determine which risk classes an issue belongs to (i.e. labeling). The threshold was chosen such that it gives a good balance between the three risk classes. The thresholds were determined based on the distribution of delayed issues with respect to the delay time in each project. Since we were interested in three levels of delay (minor, medium, and major) and no delay, we divided the delayed issues in each project into three groups: under the 33th percentile (minor delay), between the 33th percentile and the 66th percentile (medium delay), and above the 66th percentile (major delay). Note that the 33th percentile is the delay time below which

33% (or about one-third) of the delayed issues can be found. Since (major/medium/minor) delayed issues are rare (only 21% of all collected issues), we had to be careful in creating the training and test sets. Specifically, we placed 80% of the delayed issues into the training set and the remaining 20% into the test set. In addition, we tried to maintain a similar ratio between delayed and non-delayed issues in both test set and training set, i.e. stratified sampling. We also combined all issues collected across the eight projects as another dataset called "All together".

Table 4.3: Experimental setting

Project	Training set				Test set			
	Major	Medium	Minor	Non	Major	Medium	Minor	Non
Apache	698	431	688	3,202	187	265	337	481
Duraspace	242	218	246	2,064	87	93	84	642
Java.net	748	538	647	10,443	130	251	300	3,569
JBoss	551	423	382	1,320	93	196	255	306
JIRA	240	379	267	2,242	142	75	155	928
Moodle	409	401	408	10,976	237	218	236	4,119
Mulesoft	365	325	344	3,802	82	106	106	3,139
WSO2	307	284	266	2,932	107	118	158	1,057
All together	3,253	2,715	2,982	34,049	958	1,204	1,473	13,184

("All together" is an integration of issues from all the eight projects.)

4.6.2 Applying feature selection

We applied feature selection techniques on the training set of each project as well as in all the projects together. Table 4.4 shows all the risk factors and their associated p-value (last column – using all issues collected across the eight projects). Note that the very small p-value is shown as "< 0.001". In our study, we select risk factors with $p < 0.05$. For example, discussion time, percentage of delayed issues that a developer involved with, developer's workload, number of comments, priority changing, number of fix versions, and changing of fix versions are among the risk factors we selected for the Apache project.

The collinearity checking was applied on the selected features. Table 4.5 shows the indication of collinearity in terms of Variance-Decomposition Proportions (VDP) from applying Belsley collinearity diagnostics [204]. Variance-Decomposition Proportions (VDP) shows the variance proportion of the risk factors. A high VDP indicates the collinearity of at least two risk factors. In our study, we consider risk factors having

Table 4.4: P-value from logistic regression model, trained on the issues in the training set from the eight projects and “All together”

	Apache	Duraspace	Java.net	JBoss	JIRA	Moodle	Mulesoft	WSO2	All
discussion	<0.001	0.008	<0.001	0.608	0.003	<0.001	<0.001	<0.001	<0.001
repetition	0.086	<0.001	<0.001	<0.001	0.049	<0.001	0.928	0.408	0.008
perofdelay	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
workload	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	0.221	0.207	<0.001
#comment	<0.001	0.002	<0.001	0.607	<0.001	<0.001	<0.001	<0.001	<0.001
#p_change	0.007	<0.001	0.012	0.417	<0.001	<0.001	<0.001	<0.001	<0.001
#fixversion	<0.001	0.388	0.065	<0.001	<0.001	0.040	0.042	<0.001	<0.001
#fv_change	<0.001	0.253	<0.001	0.019	<0.001	<0.001	<0.001	<0.001	<0.001
#issuelink	<0.001	<0.001	<0.001	0.060	<0.001	<0.001	0.189	0.112	<0.001
#blocking	1	0.094	0.005	0.151	1	<0.001	1	1	0.023
#blockedby	1	0.065	0.002	0.262	1	<0.001	1	1	0.115
#affectver	0.835	0.955	0.965	<0.001	<0.001	<0.001	<0.001	0.984	<0.001
reporterrep	<0.001	<0.001	<0.001	0.712	0.086	<0.001	0.107	0	<0.001
#des_change	0.004	0.023	0.905	0.013	<0.001	<0.001	<0.001	0.002	<0.001
elapsedtime	<0.001	<0.001	<0.001	0.027	<0.001	<0.001	<0.001	<0.001	<0.001
remaining	<0.001	<0.001	<0.001	0.026	<0.001	<0.001	0.080	<0.001	0.396
topic	0.187	0.219	0.012	0.929	0.058	0.002	0.065	<0.001	0.040
t_Bug	<0.001	0.919	<0.001	<0.001	<0.001	<0.001	0.854	0.577	<0.001
t_FuncTest	1	1	1	1	1	<0.001	1	1	0.017
t_Imp	0.029	0.446	0.128	1	<0.001	0.253	0.106	0.471	<0.001
t_NewFeat	<0.001	0.126	<0.001	1	0.419	<0.001	0.837	0.281	<0.001
t_Story	1	<0.001	0.944	0.072	0.144	0.821	0.005	1	0.509
t_SubTask	<0.001	0.168	0.164	<0.001	<0.001	<0.001	0.086	0.825	0.085
t_Suggestion	1	1	1	1	<0.001	1	1	1	0.001
t_Task	0.026	<0.001	<0.001	0.102	0.015	0.562	0.734	0.118	<0.001
p_Trivial	0.003	0.017	0.793	0.709	0.001	0.262	0.106	0.009	0.012
p_Minor	0.739	<0.001	0.187	0.191	0.003	0.640	0.592	0.110	<0.001
p_Major	0.075	0.058	0.058	0.220	<0.001	<0.001	0.179	0.065	<0.001
p_Critical	0.241	0.483	0.006	0.221	0.008	<0.001	0.178	0.006	<0.001
p_Blocker	0.266	<0.001	0.327	0.948	0.140	0.001	0.159	0.054	0.001

discussion = Discussion time, repetition = Number of times that an issue is reopened, perofdelay = Percentage of delayed issues that a developer involved with, workload = Developer’s workload, #comment = Number of comments, #p_change = Changing of priority, #fixversion = Number of fix version, #fv_change = Changing of fix versions , #issuelink = Number of issue link, #blocking = Number of issues that are blocked by this issue, #blockedby = Number of issues that block this issue, #affectver = Number of affect version, reporterrep = Reporter reputation, #des_change = Changing of description, elapsedtime = Number of days from when the issue is created until prediction time, remaining = Number of days from prediction time until the due date, topic = issue’s topics, t_Bug = Bug type, t_FuncTest = Functional Test type, t_Imp = Improvement type, t_NewFeat = New Feature type, t_Story = Story type, t_SubTask = Sub-Task type, t_Suggestion = Suggestion type, t_Task = Task type, p_Trivial = Trivial level, p_Minor = Minor level, p_Major = Major level, p_Critical = Critical level, p_Blocker = Blocker level

$VDP > 0.5$ exhibit multi-collinearity. For example, in the Apache project, number of fix versions and two issue's types (Bug and Implementation) have VDP exceed the threshold ($VDP > 0.5$). In this case, among these high correlated risk factors, number of fix versions has the lowest p-value ($2.1574e-42$) is then selected, while the others are filtered out.

Table 4.6 shows all the risk factors and their weights in each project as well as in all the projects together (last column – using all issues collected across the seven projects) obtained from applying ℓ_1-penalized logistic regression model on the training set. The weights have intuitive meanings – this is in fact one benefit of logistic regression over other types of classifiers. The sign of a weight is its *direction* of correlation with the chance of an issue causing a delay. For example, in Mulesoft, the weight of the developer's workload is negative (–0.212), indicating that the developer's workload is negatively correlated with delayed issues. By contrast, the weight of the developer's workload in the other six projects (except the Java.net project) is positive (e.g. 3.298 in Apache), meaning the developer's workload being positively correlated with delayed issues. This cross-project diversity can also be observed in other risk factors (except the discussion time and the percentage of delayed issues that a developer involved with factors which are positively correlated with delayed issues in all the eight projects). We also note that the magnitude of each weight approximately indicates the degree to which a factor affects the probability of an issue causing delays [214]. Note that the exponential of a weight is an *odds-ratio* which is a measure of the association between the presence of a risk factor and a delay. The odds-ratio represents the odds that a delay will occur given the presence of risk factor, compared to the odds of the delay occurring in the absence of that risk factor. An odds is the ratio of the delay probability and the non-delay probability. This allows us to see the relative strength of the risk factors.

In our study, we select risk factors with *non-zeros* weight, i.e. we excluded factors that have no (either positive or negative) correlation with delayed issues. For example, discussion time, number of fix versions, number of issues that are blocked by this issue, number of issues that block this issues, and number of affect versions are among the risk factors we selected for the Duraspace project. Note that we selected different sets of risk factors for different projects due to the project diversity.

Table 4.5: Indication of collinearity in terms of Variance-Decomposition Proportions (VDP) from Belsley collinearity diagnostics, apply on the risk factors selected from using p-values

	Apache	Duraspace	Java.net	JBoss	JIRA	Moodle	Mulesoft	WSO2	All
discussion	0.0007	0.0002	0.0006		0.0005	0.1256	0.5328	0.0180	0.0005
repetition		0.0021	0.0039	0.0039	0.0003	0.0000			0.0002
perofdelay	0.0336	0.0020	0.0009	0.1863	0.0013	0.0063	0.0001	0.0004	0.0010
workload	0.0145	0.2086	0.0255	0.0183	0.0003	0.0016			0.0002
#comment	0.0001	0.0402	0.0045		0.0069	0.0010	0.0160	0.0709	0.0011
#p_change	0.0000	0.0237	0.0049		0.0083	0.0007	0.0209	0.0065	0.0038
#fixversion	0.7823			0.9433	0.0063	0.7814	0.0266	0.9449	0.1529
#fv_change	0.0035		0.0152	0.0030	0.0078	0.0531	0.7983	0.4787	0.0022
#issuelink	0.0014	0.0523	0.9395		0.0000	0.0000			0.0003
#blocking			0.5021			0.0006			0.0002
#blockedby			0.8427			0.0001			
#affectver				0.4759	0.0034	0.4394	0.8212		0.0006
reporterrep	0.0409	0.3237	0.0011			0.0094		0.0752	0.2125
#des_change	0.0017	0.0029		0.0079	0.0022	0.0001	0.0024	0.0007	0.0006
elapsedtime	0.0052	0.0064	0.0006	0.0001	0.0025	0.1919	0.9097	0.0416	0.0012
remaining	0.0233	0.0089	0.0003	0.0000	0.0005	0.0000		0.0100	
topic			0.0204			0.0068		0.6498	0.2125
t_Bug	0.7129		0.0061	0.0576	0.8720	0.1850			0.3803
t_FuncTest						0.0478			0.0590
t_Imp	0.5078				0.6604				0.2256
t_NewFeat	0.2530		0.0004			0.0054			0.1172
t_Story		0.0066					0.0000		
t_SubTask	0.3485			0.0890	0.5711	0.0459			
t_Suggestion					0.0105				0.1019
t_Task	0.2653	0.0034	0.0002		0.2662				0.1823
p_Trivial	0.0003	0.0590			0.3478			0.0031	0.2745
p_Minor		0.6322			0.6953				0.8313
p_Major					0.8409	0.0009			0.9330
p_Critical			0.0002		0.3836	0.0033		0.0702	0.7323
p_Blocker		0.4525				0.0052			0.6405

discussion = Discussion time, repetition = Number of times that an issue is reopened, perofdelay = Percentage of delayed issues that a developer involved with, workload = Developer's workload, #comment = Number of comments, #p_change = Changing of priority, #fixversion = Number of fix version, #fv_change = Changing of fix versions , #issuelink = Number of issue link, #blocking = Number of issues that are blocked by this issue, #blockedby = Number of issues that block this issue, #affectver = Number of affect version, reporterrep = Reporter reputation, #des_change = Changing of description, elapsedtime = Number of days from when the issue is created until prediction time, remaining = Number of days from prediction time until the due date, topic = issue's topics, t_Bug = Bug type, t_FuncTest = Functional Test type, t_Imp = Improvement type, t_NewFeat = New Feature type, t_Story = Story type, t_SubTask = Sub-Task type, t_Suggestion = Suggestion type, t_Task = Task type, p_Trivial = Trivial level, p_Minor = Minor level, p_Major = Major level, p_Critical = Critical level, p_Blocker = Blocker level

Table 4.6: Descriptive ℓ_1-penalized logistic regression model for risk probability, trained on the issues in the training set from the eight projects and “All together”

	Apache	Duraspace	Java.net	JBoss	JIRA	Moodle	Mulesoft	WSO2	All
discussion	4.897	3.404	4.806	1.090	2.960	6.998	11.990	4.185	5.596
repetition	2.038	10.293	6.230	4.716	2.395	-3.520	0.000	2.458	2.292
perofdelay	22.942	17.079	6.390	6.740	29.805	20.697	6.796	43.263	12.876
workload	3.298	2.232	-1.260	2.890	0.799	18.522	-0.212	7.404	1.455
#comment	-1.017	-11.796	-5.685	0.050	-0.743	-22.192	-7.156	-6.949	-9.259
#p_change	0.607	18.045	1.474	6.190	-2.274	-1.193	0.157	1.792	8.911
#fixversion	-1.695	-0.133	3.671	-0.479	0.565	2.541	1.385	-2.014	0.766
#fv_change	-5.041	-0.381	-5.219	-0.045	-7.375	-13.764	-9.289	-6.045	-3.431
#issuelink	0.003	3.967	3.857	4.492	-1.863	2.431	0.149	0.063	3.667
#blocking	0.000	-5.261	0.254	-2.189	0.000	9.784	0.000	0.000	1.890
#blockedby	0.000	-3.274	0.806	-6.288	0.000	-0.436	0.000	0.000	0.062
#affectver	1.490	0.535	-0.008	-6.288	-4.389	-6.204	-8.862	-1.606	-2.860
reporterrep	-1.092	-0.618	-0.453	0.208	-0.610	-0.486	-0.163	-1.943	-0.484
#des_change	-0.843	-3.025	1.192	1.412	-1.034	2.073	-0.546	0.038	-1.866
elaspedtime	-2.333	-4.522	1.539	-0.634	-1.062	2.073	6.141	1.877	-1.124
remaining	0.068	-4.234	-4.733	-3.181	5.588	-3.636	-1.148	-2.424	-0.362
topic	0.055	0.036	0.208	0.136	-0.418	-0.298	-0.374	0.041	0.017
t_Bug	-0.624	-0.056	-0.370	-0.900	0.117	-2.257	0.000	0.000	-1.103
t_FuncTest	0.000	0.000	0.000	0.000	0.000	-2.321	0.000	0.000	-0.692
t_Imp	-0.517	0.000	-0.138	0.000	1.763	-1.907	-0.237	0.406	-0.663
t_NewFeat	0.020	0.000	0.044	0.000	-0.184	-1.432	0.000	0.134	-0.585
t_Story	0.000	-0.307	-0.061	0.003	-0.301	-2.020	1.995	0.000	-1.038
t_SubTask	-0.358	0.086	-0.126	-0.920	-0.347	-2.328	0.223	-0.430	-0.899
t_Suggestion	0.000	0.000	0.000	0.000	1.770	0.000	0.000	0.000	-1.314
t_Task	0.227	0.511	0.004	-0.542	0.826	-2.001	0.129	-0.002	-0.665
p_Trivial	-0.409	-0.375	-0.270	-0.001	0.876	-0.564	0.658	0.000	-0.060
p_Minor	0.031	0.016	-0.061	0.000	2.427	-0.085	0.039	0.786	0.250
p_Major	0.076	-0.001	0.141	-0.004	1.838	-0.296	0.880	-0.097	0.309
p_Critical	0.115	-0.433	-0.299	-0.477	1.812	-0.059	1.934	-0.386	-0.111
p_Blocker	-0.265	0.192	-0.257	-0.014	1.214	-0.057	1.949	-0.352	0.033

discussion = Discussion time, repetition = Number of times that an issue is reopened, perofdelay = Percentage of delayed issues that a developer involved with, workload = Developer’s workload, #comment = Number of comments, #p_change = Changing of priority, #fixversion = Number of fix version, #fv_change = Changing of fix versions , #issuelink = Number of issue link, #blocking = Number of issues that are blocked by this issue, #blockedby = Number of issues that block this issue, #affectver = Number of affect version, reporterrep = Reporter reputation, #des_change = Changing of description, elapsedtime = Number of days from when the issue is created until prediction time, remaining = Number of days from prediction time until the due date, topic = issue’s topics, t_Bug = Bug type, t_FuncTest = Functional Test type, t_Imp = Improvement type, t_NewFeat = New Feature type, t_Story = Story type, t_SubTask = Sub-Task type, t_Suggestion = Suggestion type, t_Task = Task type, p_Trivial = Trivial level, p_Minor = Minor level, p_Major = Major level, p_Critical = Critical level, p_Blocker = Blocker level

4.6.3 Performance measures

As our risk classes are ordinal and imbalanced, standard report of precision/recall for all classes is not fully applicable. In addition, no-delays are the default and they are not of interest to risk management. Reporting the average of precision/recall across classes is likely to overestimate the true performance. Furthermore, class-based measures ignore the ordering between classes, i.e., major-risk class is more important than minor-risk. Hence, we used a number of predictive performance measures suitable for ordinal risk classes for evaluation described as below.

Precision/Recall/F-measures/AUC

A confusion matrix is used to evaluate the performance of our predictive models. As a confusion matrix does not deal with a multi-class probabilistic classification, we reduce the classified issues into two binary classes: delayed and non-delayed using the following rule:

$$C_i = \begin{cases} delayed, & if P(i,Maj) + P(i,Med) + P(i,Min) > P(i,Non) \\ non-delayed, & otherwise \end{cases}$$

where C_i is the binary classification of issue i, and $P(i,Maj)$, $P(i,Med)$, $P(i,Min)$, and $P(i,Non)$ are the probabilities of issue i classified in the major delayed, medium delayed, minor delayed, and non-delayed class respectively. Basically, this rule determines that an issue is considered as delayed if the sum probability of it being classified into the major, medium, and minor delayed class is greater than the probability of it being classified into the non-delayed class.

The confusion matrix is then used to store the correct and incorrect decisions made by a classifier. For example, if an issue is classified as delayed when it was truly delayed, the classification is a true positive (tp). If the issue is classified as delayed when actually it was not delayed, then the classification is a false positive (fp). If the issue is classified as non-delayed when it was in fact delayed, then the classification is a false negative (fn). Finally, if the issue is classified as non-delayed and it was in fact not delayed, then the classification is true negative (tn). We thus compute the widely-used Precision, Recall, F-measure, and AUC for the delayed issues to evaluate the performance of the predictive models. These measures are described in Section 2.1.6.

Macro-averaged Mean Cost-Error (MMCE)

The confusion matrix however does not take into account our multi-class *probabilistic* classifications and the cost associated with each risk class. Hence, we propose a new measure known as Macro-averaged Mean Cost-Error (MMCE) to assess *how close our predictive risk exposure is to the true risk exposure* (the distance between them in the sense that the smaller the better).

Let y^i be the true class and $\hat{y}^i$ be the predicted class of issue i. Let n_k be the number of true cases with class k where $k \in \{1,2,3,4\}$ – there are 4 classes in our classification – i.e., $n_k = \sum_{i=1}^{n} \delta\left[y^i = k\right]$ and $n = n_1 + n_2 + n_3 + n_4$. Here $\delta\,[.]$ is the indicator function.

The Macro-averaged Mean Cost-Error[p] is defined as below:

$$\text{MMCE} = \frac{1}{4}\sum_{k=1}^{4}\frac{1}{n_k}\sum_{i=1}^{n}|\bar{RE}_i - C|\,\delta\left[y^i = k\right]$$

where $\bar{RE}$ is the predicted risk exposure computed in Section 4.4.2 and C is the actual risk exposure. The normalization against the class size makes MMCE insensitive to the class imbalance.

For example, an issue is predicted to be 50% in major delayed, 15% in medium delayed, 30% in minor delayed, and 5% in non-delayed. Assume that the issue was actually minor delayed (i.e. the true class is minor delayed) and the costs C_1 (no delay), C_2 (minor delay), C_3 (medium delay) and C_4 (major delay) are respectively 0, 1, 2, and 3. The predicted risk exposure $\bar{RE}$ is 2.1 and the actual risk exposure is 1 (see Section 4.4.2 for how a risk exposure is calculated). Hence, the MMCE error between actual and predicted risk exposure for this issue is 1.1.

Macro-averaged Mean Absolute Error (MMAE)

We also used another metric called Macro-averaged Mean Absolute Error (MMAE) [158] to assess the distance between actual and predicted classes. MMAE is suitable for ordered classes like those defined in this chapter. For example, if the actual class is non-delayed ($k = 1$), and the predicted class is major delayed ($k = 4$), then an error of 3 has occurred. Here, we assume that the predicted class is the one with the highest probability, but we

[p]Here we deal with only 4 classes but the formula can be easily generalized to n classes.

acknowledge that other strategies can be used in practice. Again the normalization against the class size handles the class imbalance.

Macro-averaged Mean Absolute Error:

$$\text{MMAE} = \frac{1}{4}\sum_{k=1}^{4}\frac{1}{n_k}\sum_{i=1}^{n}\left|\hat{y}^i - k\right|\delta\left[y^i = k\right]$$

For example, an issue is predicted to be 30% in major delayed, 35% in medium delayed, 25% in minor delayed, and 10% in non-delayed. Thus, the predicted class of this issue is medium delayed (k = 3). Assume that the actual class of the issue is non-delayed (k = 1), then the distance between actual and predicted classes of this issue is 2.

4.6.4 Results

Comparison of different projects (RQ1)

Figure 4.11 shows the precision, recall, F-measure, and AUC achieved for each of eight open source projects and in all the projects (labeled by “All together”), averaging across all classifiers and across all feature selection techniques. Overall, the evaluation results on eight projects (i.e. Apache, Duraspace, Java.net, JBoss, JIRA, Moodle, Mulesoft, WSO2) and All together achieve on average 0.79 precision, 0.61 recall, and 0.68 F-measure. We however noted that the imbalance of delayed and non-delayed classes may impact on the predictive performance since there are only 7.2% of issues in the major delayed class in the test set. In particular, for projects that our predictive models struggled, there might be some changes in the projects (e.g. additional contributors joined) between the training time and test time, and thus patterns present at training time may not entirely repeat later on at test time which shows the variety of open source projects nature.

The degree of discrimination achieved by our predictive models is also high, as reflected in the AUC results. The AUC quantifies the overall ability of the discrimination between the delayed and non-delayed classes. The average of achieved AUC across all projects and across all classifiers is 0.83. Our model performed best in the Duraspace project, achieving the highest precision (0.85), recall (0.72), F-measure (0.77), and AUC (0.93).

Answer to RQ1: Overall, the evaluation results demonstrate the effectiveness of our predictive models across the eight projects.

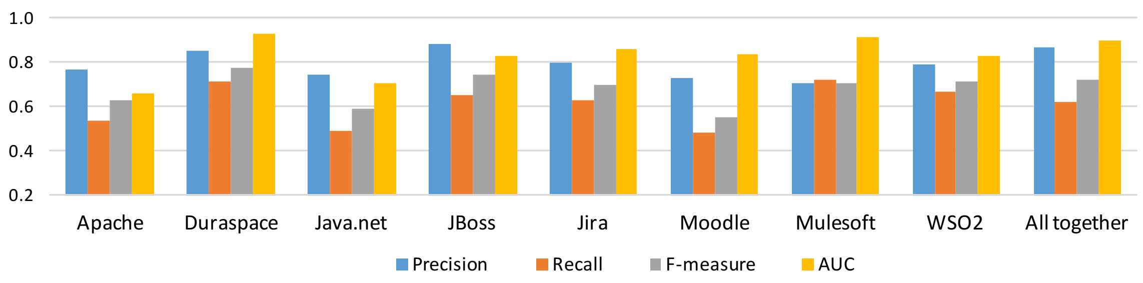


Figure 4.11: Evaluation results for different projects

Comparison of different classifiers (RQ2)

Table 4.7 shows the precision, recall, F-measure, and AUC achieved by Random Forests (RF), Neural Network (aNN), Decision Tree (C4.5), Naive Bayes (NB), NBTree, Deep Neural Networks with Dropouts (Deep Nets), and Gradient Boosting Machines (GBMs) in each project (averaging across two feature selection techniques) using the training/test set setting. As can be seen in Table 4.7, Random Forests achieve the highest F-measure of 0.72 (averaging across all projects and across two feature selection techniques). It also outperforms the other classifiers in terms of F-measure in four projects: Apache, Duraspace, Java.net, and Mulesoft. In the JBoss project, Random Forests achieve the highest precision of 0.96 (averaging across two feature selection techniques), while the other classifiers achieve only 0.77–0.92 precision. It should be noted that all classifiers achieve more than 0.5 AUC while Random Forests is also the best performer in this aspect with 0.99 AUC.

Answer to RQ2: Random Forests is the best performer in precision, recall, F-measure, and AUC (averaging across all projects).

Comparison of different feature selection approaches (RQ3)

We performed the experiments to compare the performance of different feature selection approaches: ℓ_1*-penalized logistic regression model* and using *p-value from logistic regression model*. As can be seen in Figure 4.12, the ℓ_1-penalized logistic regression model produced the best performance in terms of F-measure – it achieved 0.79 precision, 0.63 recall, 0.69 F-measure, and 0.83 AUC, while the performance achieved using p-value from logistic regression model is 0.79 precision, 0.60 recall, 0.67 F-measure, and 0.83 AUC (averaging across all projects and across all classifiers). This could be interpreted that the feature selection technique based on the assessing of the significance of all fea-

Table 4.7: Evaluation results for different classifiers in each project

Proj.	Classifier	Prec	Re	F	AUC	Proj.	Classifier	Prec	Re	F	AUC
AP	RF	0.80	0.63	**0.71**	**0.77**	JI	RF	0.90	0.60	0.72	0.86
	aNN	0.74	0.54	0.62	0.68		aNN	0.86	0.65	**0.73**	0.84
	C4.5	0.80	0.52	0.63	0.66		C4.5	0.72	**0.67**	0.69	0.89
	NB	0.67	0.48	0.55	0.54		NB	0.65	**0.67**	0.67	0.82
	NBTree	0.70	**0.64**	0.67	0.59		NBTree	0.82	0.62	0.69	**0.91**
	Deep Nets	0.81	0.50	0.61	0.67		Deep Nets	0.73	0.63	0.67	0.82
	GBMs	**0.86**	0.45	0.59	0.71		GBMs	**0.92**	0.52	0.67	0.85
DU	RF	**0.95**	0.75	**0.83**	**0.99**	MO	RF	0.73	0.63	0.67	**0.91**
	aNN	0.85	0.65	0.74	0.87		aNN	**0.85**	0.26	0.39	0.76
	C4.5	0.88	**0.77**	0.82	0.93		C4.5	0.73	0.49	0.58	0.76
	NB	0.59	0.62	0.59	0.77		NB	0.47	0.59	0.51	0.79
	NBTree	0.80	0.74	0.77	0.98		NBTree	0.84	0.40	0.54	0.88
	Deep Nets	0.91	0.75	0.82	0.97		Deep Nets	0.74	**0.69**	**0.71**	**0.91**
	GBMs	0.93	0.73	0.81	0.98		GBMs	0.72	0.33	0.45	0.85
JA	RF	0.75	0.57	**0.74**	**0.88**	MU	RF	0.72	0.79	**0.77**	**0.93**
	aNN	0.75	0.54	0.62	0.68		aNN	**0.83**	0.69	0.75	0.91
	C4.5	0.80	0.52	0.63	0.66		C4.5	0.73	0.65	0.66	0.90
	NB	0.64	0.48	0.54	0.54		NB	0.71	0.74	0.73	0.92
	NBTree	0.70	**0.65**	0.67	0.60		NBTree	0.70	0.72	0.71	0.92
	Deep Nets	**0.84**	0.50	0.63	0.86		Deep Nets	0.57	**0.81**	0.67	0.92
	GBMs	0.73	0.22	0.28	0.72		GBMs	0.66	0.69	0.67	0.85
JB	RF	**0.96**	0.63	0.77	0.84	W2	RF	0.76	**0.69**	0.72	0.85
	aNN	0.77	0.58	0.66	0.77		aNN	**0.94**	0.65	0.75	0.82
	C4.5	0.88	**0.76**	**0.81**	0.85		C4.5	0.85	0.66	0.74	0.86
	NB	0.82	0.64	0.71	0.78		NB	0.55	0.68	0.60	0.76
	NBTree	0.90	0.72	0.79	**0.89**		NBTree	0.84	0.68	0.75	0.86
	Deep Nets	0.87	0.67	0.76	0.81		Deep Nets	0.70	**0.69**	0.69	0.84
	GBMs	0.92	0.60	0.72	0.85		GBMs	0.90	0.68	**0.77**	**0.88**

AP: Apache, DU: Duraspace, JA: Java.net, JB: JBoss,
JI: JIRA, MO: Moodle, MU: Mulesoft, W2:WSO2

tures (i.e. ℓ_1*-penalized logistic regression model*) effectively contributes to the predictive performance.

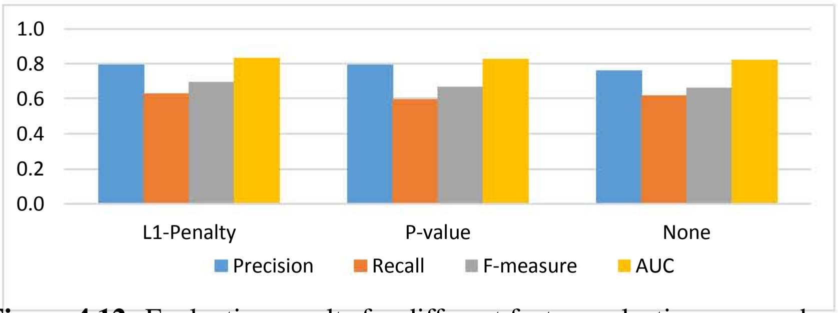


Figure 4.12: Evaluation results for different feature selection approaches

There was 34.5% of features eliminated using feature selection, averaging across all projects and across the two feature selection techniques. We also evaluated our predictive models without using a feature selection technique (labeled by "None") (i.e. all extracted features are fed into classifiers). As can be seen in Figure 4.12, the ℓ_1-penalized logistic regression model also produced the better performance than the predictive models without a feature selection technique. It should however be noted that the predictive performance obtained from not using a feature selection is comparable to the others – it achieves 0.76 precision, 0.62 recall, 0.66 F-measure, and 0.82 AUC averaging across all projects and across all classifiers. There are two interpretations which can be made here. First, we could eliminate more than one-third of the extracted features without greatly affecting the predictive performance. Second, the two state-of-the-art randomized ensemble methods (i.e. Random Forests and Deep Neural Networks with Dropouts) have a capability to reduce prediction variance, prevent overfitting, and be tolerant to noisy data. Thus, they might eliminate the need for feature selection in our particular setting while still can improve the overall predictive accuracy. Especially, Deep Neural Networks with Dropouts is known to perform better with extensively large datasets.

Answer to RQ3: Feature selection using the ℓ_1-penalized logistic regression model is the best performer.

Comparison of different prediction times (RQ4)

As can be seen in Figure 4.13, predicting at the issue creation time produced a low performance – it achieves only 0.35 precision, 0.15 recall, 0.18 F-measure, and 0.55 AUC averaging across all projects and across all classifiers, while the predictive performance achieved by making the prediction at a time when a due date was assigned is comparable to the latter prediction (i.e. at the end of discussion time) – it achieves 0.70 precision, 0.59 recall, 0.61 F-measure, and 0.78 AUC averaging across all projects and across all classifiers. However, Predicting at later times may be less useful since the outcome may become obvious or it is too late to change the outcome. This result confirms our hypothesis that the time when the prediction is made has implications to its accuracy.

Answer to RQ4: The prediction time affects the predictive performance: the later we predict, the more accuracy we could gain since more information has become available.

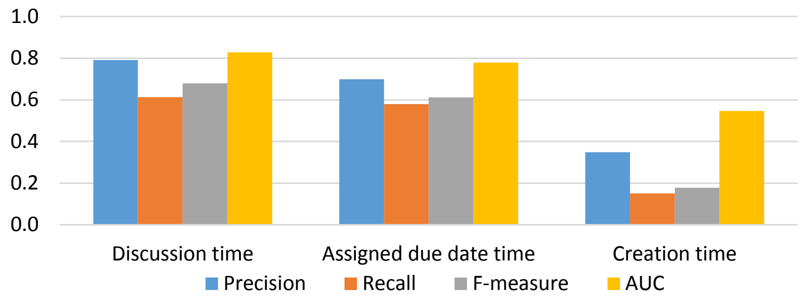


Figure 4.13: Evaluation results for different prediction times

Comparison of different numbers of topics (RQ5)

We also performed the experiments to compare the performance of predictive models using different numbers of topics (i.e. 10, 100, 200, 300, and 400 topics). Figure 4.14 shows the precision, recall, F-measure, and AUC achieved for the different number of topics (averaging across the eight projects). The predictive models were trained by using Random Forests and ℓ_1-*penalized logistic regression model* since this combination is the best performer in several aspects. As can be seen in Figure 4.14, varying the number of topics do not significantly improve the performance – it achieves 0.83 precision, 0.65 recall, 0.75 F-measure, and 0.89 AUC averaging across all projects using 10 topics, while it achieves 0.83 precision, 0.66 recall, 0.73 F-measure, and 0.86 AUC averaging across all projects using 400 topics. Table 4.8 shows the weight of topics from ℓ_1-*penalized logistic regression model*. We also noticed that, in the Duraspace project, the increasing of the number of topics causes zero discriminative power. We note that advanced techniques to learn features from textual information (e.g. vector representation for text using deep learning approaches) have been proposed which could be used in our future work.

Answer to RQ5: The number of topics do not significantly affect the predictive performance.

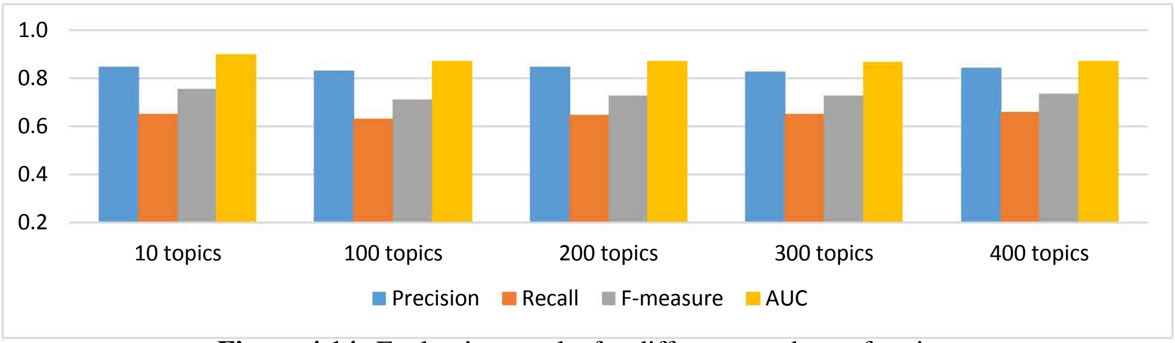


Figure 4.14: Evaluation results for different numbers of topics

Table 4.8: Weight obtained from ℓ_1-penalized logistic regression model using the different number of topics

Project	10 topics	100 topics	200 topics	300 topics	400 topics
Apache	0.055	0.05	0.044	-0.002	-0.397
Duraspace	0.036	-0.075	-0.025	0	0
Java.net	0.208	0.315	-0.038	0.077	0.061
JBoss	0.136	0.009	0.067	0.199	-0.046
JIRA	-0.418	0.101	0.153	0.473	0.156
Moodle	-0.298	0.188	0.167	0.066	-0.126
Mulesoft	-0.374	0.335	-0.246	0.04	-0.149
WSO2	0.041	0.731	0.142	0.335	0.135

MMAE and MMCE as performance measures (RQ6)

MMAE and MMCE are used to assess the performance of our models in terms of predicting risk exposure. Figure 4.15 shows the MMAE and MMCE achieved by the eight classifiers using two feature selection techniques. The evaluation results show that MMAE and MMCE are generally consistent with the other measures. For example, Random Forests have the highest precision and recall, and the lowest MMAE and MMCE – it achieves 0.72 MMAE and 0.66 MMCE averaging across all projects and across the two feature selection techniques.

Answer to RQ6: Our approach consistently performs well in predicting the risk exposure of issue delays.

Sliding window approach (RQ7)

For the sliding window approach, we performed the experiments on "All together" dataset. In the first setting, all historical issues from past windows are accumulated and learned

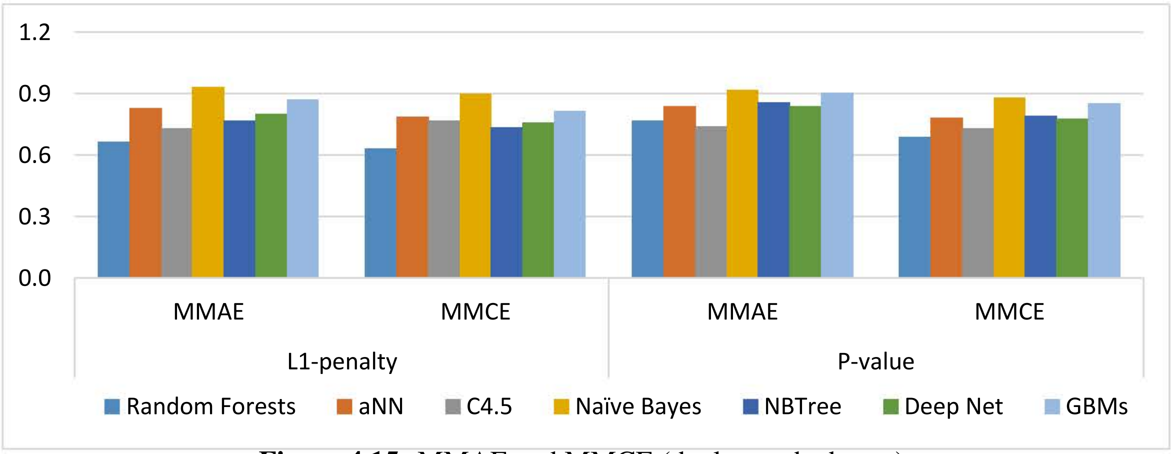


Figure 4.15: MMAE and MMCE (the lower the better)

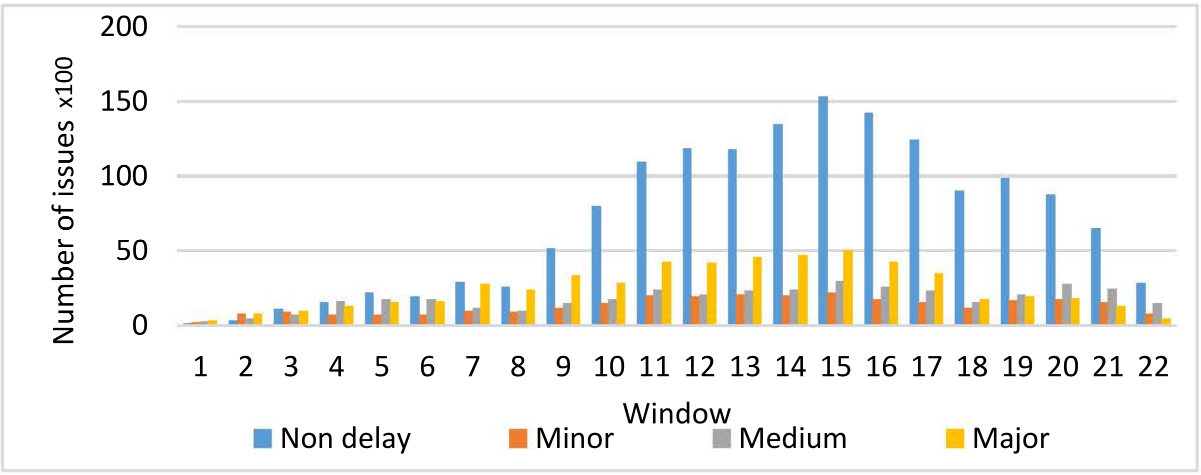


Figure 4.16: Number of issues in each window (6-month window)

to predict delayed issues in the next window. Figure 4.16 shows the number of issues in each class splitted into 22 windows. The time period of each window is six months. For example, the first window contains the issues created between January 01, 2004 – June 30, 2004, which are learned to predict the outcome of the issues created between July 1, 2004 – December 31, 2004 (i.e. the second window). Note that the issues opened after *December, 31 2015* were included to the last window (22^{th} window). We however acknowledge that the numbers of issues available in the 1^{st} to 8^{th} window are very small. This is known as “cold-start problem” [215] where there are not enough data available to begin a predictions. Hence, we started doing the prediction at the 6^{th} window – the issues from the 1^{st} to 5^{th} window are learned to predict the outcome of the issues in the 6^{th} window and so on.

Figure 4.17 shows the evaluation results from the first sliding window setting. Random Forests and the ℓ_1-penalized logistic regression model are employed because it achieved the highest performance on the traditional setting (i.e. training/test splitting).

The predictive performance from predicting the issues in the 6th window achieves 0.66 precision, 0.79 recall, 0.72 F-measure, and 0.83 AUC. As can be observed from Figure 4.17, The predictive performance in terms of F-measure then decreases after the 8th window (it achieves the high recall and the low precision). This could be due to the class imbalance problem (i.e. lacking of delayed class in the training set) since a number of non-delayed issues drastically increases after the 8th window. The predictive performance measures (i.e. precision, recall, and F-measure) then converges at the 16th window and slightly increases after the 18th window.

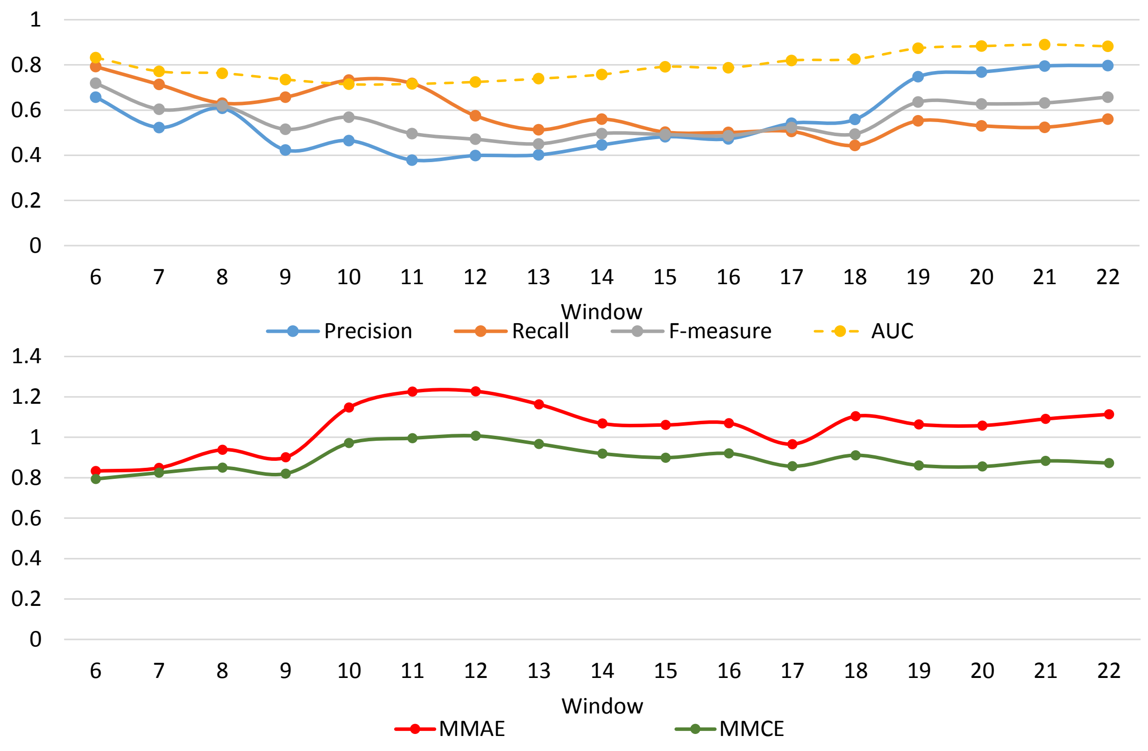


Figure 4.17: Evaluation results using the 6-month sliding window setting

In the continuous sliding window setting, we expanded the window size from 6 months to 2 years in order to increase the number of issues in each window. Hence, the issues were divided into 6 windows (the data were available from the year 2004 to 2016). Note that no prediction was made for the first window since the training set was not available for it. Figure 4.18 shows the evaluation results from the continuous sliding window using Random Forests and the ℓ_1-penalized logistic regression model. The predictive performance achieved by the continuous sliding window shows the similar pattern to the first setting. It achieves the high recall and the low precision. The precision then increases in later windows. However, it can be clearly seen that the overall performance achieved by the continuous sliding window is better than the first setting which all the historical issues were learned in a prediction.

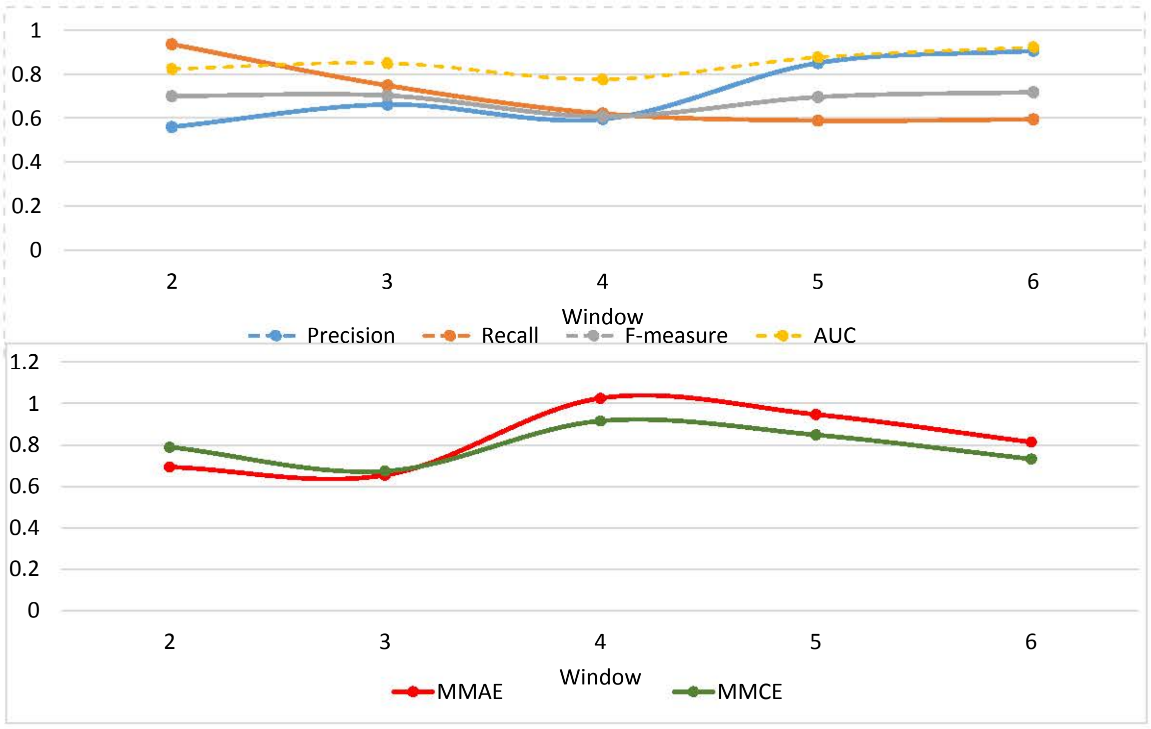


Figure 4.18: Evaluation results using the 2-year sliding window setting

Answer to RQ7: The evaluation results demonstrate that the predictive model trained from the *recent* issues can increase the predictive performance (e.g. preventing the overfitting problem).

4.6.5 Implications

Outcomes from the risk factor selection process (see Section 4.3) help us identify the best factors for predicting delayed issues. Although the risk factors and the degree to which they affect the probability of an issue causing delay are different from project to project, we have also seen some common patterns, e.g."the discussion time" and "the percentage of delayed issues that a developer is involved with" are positive factors across all the eight projects. In terms of the discrimination power of the risk factors, the top three highest discrimination power factors are: the developer's workload, the discussion time, and the percentage of delayed issues that a developer is involved with. Especially, the developer's workload is a strong factor having positive correlation with delayed issues in the Apache, Moodle, and WSO2 projects. In addition, it also corresponds to p-values from logistic regression model that developer's workload, discussion time, and the percentage of delayed issues that a developer is involved with are also significant ($p < 0.05$) for determining delayed issues in most projects (e.g. Apache, Moodle, and Java.net). Moreover,

Table 4.9: Top-10 most important risk factors with their normalized weight in each project

Apache		Duraspace		Java.net		JBoss	
Feat.	Imp.	Feat.	Imp.	Feat.	Imp.	Feat.	Imp.
workload	1	perofdelay	1	discussion	1	perofdelay	1
discussion	0.59	no_blocking	0.61	workload	0.86	workload	0.95
no_comment	0.56	discussion	0.49	no_comment	0.63	discussion	0.70
no_des_change	0.32	workload	0.38	no_des_change	0.58	no_des_change	0.67
no_issuelink	0.27	no_comment	0.22	no_priority_change	0.32	perofdelay	0.33
perofdelay	0.27	no_des_change	0.20	no_issuelink	0.22	RemainingDay	0.21
RemainingDay	0.25	no_priority_change	0.11	RemainingDay	0.22	no_blocking	0.20
no_priority_change	0.22	no_issuelink	0.11	no_fixversion_change	0.21	no_priority_change	0.19
reporterrep	0.14	RemainingDay	0.09	perofdelay	0.21	no_fixversion_change	0.18
no_fixversion_change	0.13	reporterrep	0.09	reporterrep	0.15	reporterrep	0.16
JIRA		Moodle		Mulesoft		WSO2	
Feat.	Imp.	Feat.	Imp.	Feat.	Imp.	Feat.	Imp.
workload	1	perofdelay	1	perofdelay	1	perofdelay	1
perofdelay	0.64	workload	0.60	workload	0.51	workload	0.89
no_comment	0.40	no_issuelink	0.28	no_comment	0.40	no_des_change	0.54
no_des_change	0.31	no_comment	0.26	no_des_change	0.37	discussion	0.53
discussion	0.28	no_des_change	0.21	no_issuelink	0.23	no_issuelink	0.37
type_NewFeature	0.22	no_priority_change	0.19	RemainingDay	0.16	no_priority_change	0.29
no_fixversion_change	0.19	RemainingDay	0.10	no_priority_change	0.11	RemainingDay	0.29
RemainingDay	0.14	no_blocking	0.09	reporterrep	0.08	reporterrep	0.11
no_priority_change	0.13	discussion	0.08	no_fixversion_change	0.08	repetition	0.10
reporterrep	0.10	no_fixversion_change	0.06	no_fixversion	0.08	type_Bug	0.09

the changing of priority and fix version are significant ($p < 0.05$) risk factors in most projects (e.g. Apache, JIRA, and WSO2). In most projects, the number of issue links is a significant risk factors ($p < 0.05$), while the number of blocking and blocked issues are not a significant factor ($p > 0.05$).

The issue types that have less impact on causing a delay are “Sub-Task” and “Functional Test”, while the “Story” type shows a strong indicative of delays in the Mulesoft project. Furthermore, the “Blocker” and “Critical” priority have a stronger impact on causing a delay (particularly for the Mulesoft project) than the other priority levels. By contrast, in the JIRA project the “Minor” priority has stronger indicative of delays than the “Blocker” or “Critical” priority, which may due to the different practices among projects in resolving issues.

We also investigated the important risk factors obtained using outcomes from our Random Forests model. Table 4.9 reports the top-10 most important risk factors and their weight for each project. The weights here reflect the discriminate power of a feature since they are derived from the number of times the feature is selected (based on information gain) to split in a decision tree [164]. The weights are normalized in such a way that the most important feature has a weight of 1 and the least important feature has 0 weight. We notice that the discussion time, the developer’s workload, and the percentage of delayed issues that a developer is involved with are good predictors across the eight projects which correspond to the result obtained from applying feature selection techniques. For example, in Mulesoft project, the discussion time is significant ($p < 0.05$) and has the highest discrimination power (11.990) for determining delayed issues. In most projects (e.g. Java.net and Moodle), the changing of issue's attributes (i.e. priority, fix version, and description) holds the top-5 rank which might reflect that changing of issue's attributes can be a good indicator for determining delayed issues. These would provide insightful and actionable information for project managers in risk management.

Results from our feature selection and prediction models allow us to identify a number of good factors which help decision makers determine issues at risk of being delayed. Consistently across the eight projects we studied, the developer workload, the discussion time, and the developer track record (in terms of the percentage of delayed issues that a developer has been involved) are the top three factors that have the highest discrimination power. There is a strong correlation between these factors and the delay risk, which allows us to propose a range of actionable items. Firstly, issues with long discussion time tend to have a very high risk of being delayed. This suggests that the project manager should carefully keep track the discussion involved in an issue, and should possibly impose a certain time limit on it. Our further investigation on those issues reveal that the long discussion was mostly due to the complexity of the issue. Therefore, if the discussion is still not resolved after a certain time, the team should consider, for example, splitting an issue into a number of smaller issues (known as the divide-and-conquer strategy), each of which is easier to deal with.

Secondly, the workload and track record of a developer assigned to resolve an issue are the most important indicators of whether an issue is at risk of being delayed. Our result confirms a long known phenomenon that if a developer is overloaded with many tasks, there is a high risk that they are not able to complete all of them in time. To mitigate such a risk, the project manager should try to maintain a balanced workload across team members when assigning issues to them. The track record of a developer here refers to the percentage of delayed issues in all of the issues which was assigned to a developer. For example, if a developer was assigned to 10 issues in the past and 6 of them were delayed, then their track record would be 60%. Our results show that issues assigned to developers having a high percentage of delayed issues are at a high risk of being delayed. We however note that this track record may not reflect the actual performance or skills of a developer. In practice, the best developers are usually tasked with the most difficult issues, and therefore they may take a longer time to resolve those issues. To mitigate this risk, we would therefore recommend the project manager identify the developers who involved with many delayed issues in the past, and consider for example assigning less tasks to them or allocating more time for their tasks.

4.6.6 Threats to validity

Internal validity: Our data set has the class imbalance problem. The majority of issues (over 75% of the total data) are non-delayed issues. This has implications to a classifier's ability to learn to identify delayed issues. We have used stratified sampling to mitigate this problem. We also designed and used two performance measures that are insensitive to class imbalance: the MMCE and MMAE. In addition, classifiers generally make a certain assumptions about the data, e.g. Naive Bayes assumes that the factors are conditionally independent (which may not be true in our context,) or the other classifiers generally assume that training data is sufficiently large. We have used a range of different classifiers, and performed feature selection to minimize this threat. Feature selection reduces the feature space, limits the chance of variations, and thus requires less data to learn patterns out of features. Especially, our feature selection is based on maximizing the predictive performance in held-out data, and consequently it helps deal with over-fitting and overoptimistic estimation.

Another threat to our study is that the patterns that hold in the training data may not reflect the situation in the test. There are a number of reasons for this such as the team and management having changed their approach or managed the risks they perceived. We deliberately chose the time to split training and test sets to mimic a real deployment (as opposed to traditional settings where data is split randomly). We also minimized this threat by employing the sliding window approach discussed in Section 4.6.1. In addition, we have attempted to cover most important risk factors causing delays in resolving an issue. However, we acknowledge that the set of risk factors identified in this chapter are by no means comprehensive to encompass all aspects of software projects.

External validity: We have considered more than 60,000 issue reports from the eight projects which differ significantly in size, complexity, development process, and the size of community. All issue reports are real data that were generated from open source project settings. We cannot claim that our data set would be representative of all kinds of software projects, especially in commercial settings. The primary distinct between open source project and commercial projects is the nature of contributors, developers and project's stakeholders. In open source projects, contributors are free to join and leave the communities, resulting in high turn over rate [216]. In contrast, developers in the commercial setting tend to be stable and fully commit to deliver the project's progress. Hence, further study of how our predictive models perform for commercial projects is needed.

4.7 Related work

Software risk management has attracted great attention since Boehm's seminal work, e.g., [100], [211], in the early nineties. Risk management consists of two main activities: risk assessment and risk control. Our current work focuses on risk assessment, which is a process of identifying risks, analyzing and evaluating their potential effects in order to prioritize them [100], [217]. Risk control aims to develop, engage, and monitor risk mitigation plans [100].

Statistical and machine learning techniques have been used in different aspects of risk management. For example, Letier *et al.* proposed a statistical decision analysis approach to provide a statistical support on complex requirements and architecture [218]. Their model merges requirements and constraints from various decision options to determine cost and benefit and to reduce uncertainty in architecture decisions. Pika *et al.* used statistical outlier detection techniques to analyze event logs in order to predict process delay using a number of process risk indicators such as execution time, waiting time, and resource involvement. Bayesian networks have also been used to model dependencies and probabilistic relationships between causes and effects in risk analysis [101]. For example, the work in [11] developed a Bayesian network to analyze causality constraints and eliminate the ambiguity between correlation and causality of risk factors. However, the input data of this model is the questionnaire-based analysis, which may not reflect the current situation of projects.

Another line of research that is closely related to our work is mining bug reports for fix-time prediction, e.g., [179], [177], [130], [178], [176], blocking bug prediction, e.g., [133], [131], re-opened bug prediction, e.g., [111], [113], [112], severity/priority prediction, e.g., [23], [22], [79], automatic bug categorization, e.g., [219], delays in the integration of a resolved issue to a release, e.g., [220], bug triaging, e.g., [78], [182], [221], [77], bug report field reassignment, e.g., [134], [103], bug resolver prediction, e.g., [222] and duplicate bug detection [184], [185], [186], [187], [188]. Particularly, the thread of research on predicting the fix time of a bug is mostly related to our work, and thus we briefly discuss some of those recent work here. The work in [179] estimates the fixing effort of a bug by finding the previous bugs that have similar description to the given bug (using text similar techniques) and using the known effort of fixing those previous bugs. The work in [130] used several primitive features of a bug (e.g. severity, component, number of comments, etc.) to predict the lifetime of Eclipse bugs using decision trees and other machine learning techniques. Recently, the work in [178] proposed to use Random Forrest to predict bug's fixing time using three features: location, reporter and description. The work in [177] also computed prediction models (using decision trees) for predicting bug's fixing time. They tested the models with initial bug report data as well as those with post-submission information and found that inclusion of post-submission bug report data of up to one month can further improve prediction models. Since those techniques used classifiers which do not deal with continuous response variables, they need to discretize the fix-time into categories, e.g. within 1 month, 1 year and more than 1 year as in [178]. Hence, they are not able to predict the exact time needed to resolve an issue, and thus are not readily applicable to predict if an issue will be delayed. Our future work would involve investigating how to extend those techniques for delay prediction and compare them with our approach.

4.8 Chapter summary

In this chapter, we have performed a study in eight major open source projects and extracted a comprehensive set of features (i.e. risk factors) that determine if an issue is at risk of being delayed with respect to its due date. We have developed a sparse logistic regression model and performed feature selection on those risk factors to choose those with good discriminative power. In our study, we compared two different feature selection techniques: *ℓ_1-penalized logistic regression model* and *using p-value from logistic regression model*. From our evaluation, the best predictive performance can be obtained from the assessing of the significance of all features using *ℓ_1-penalized logistic regression model*. In addition, the outcomes from feature selection techniques also confirmed the diversity of projects. Using those selected risk factors, we have developed accurate models to predict if an issue will be at risk of being delayed, if so to what extend the delay will be (risk impact), and the likelihood of the risk occurring. The evaluation results demonstrate a strong predictive performance of our predictive models with 79% precision, 61% recall, 72% F-measure, and above 80% AUC. The error rate of our predictive models measured by Macro-averaged Mean Cost-Error (MMCE) and Macro-averaged Mean Absolute Error (MMAE) are only 0.66 and 0.72, respectively. In particular, our evaluation using the sliding window approach also shows that the more learned data, the better achieved predictive performance. Moreover, the effect from the different prediction times (i.e. at the end of discussion time and at the creation time of an issue) on the predictive performance is also confirmed by our evaluation results.

Our approach described in this chapter focuses on building predictive model based on information extracted from individual issues. However, the dependency between issues can also cause a delay in software project. In the next chapter, we propose an approach to build predictive models that take relationships between issues into account.

Chapter 5

Delay prediction using networked classification

MAKING a reliable prediction of delays is an important capability for project managers, especially when facing with the inherent dynamic nature of software projects (e.g. constant changes to software requirements). In order to address that need, a number of recent proposals have leveraged data mining and machine learning technologies to estimate the fix time of a software bug (e.g. [130], [176]–[179]). In line with a large body of work in data mining for software engineering, these approaches employ traditional machine learning classification techniques to perform classification on each issue or bug *independently* using its attributes or features. The work we presented in the previous chapter follows such an approach that does *not* take into account the role of the underlying network of inter-relationships between the tasks of resolving those bugs or issues. This is a gap, given the preponderance of task dependencies in software projects – approximately 57% of 11,851 tasks (i.e. issues) in the five open source projects selected for our study were related to at least one other task. These task dependencies form the *networked data* that we will seek to leverage in this work.

Networked data are seen in many different forms in our daily life, such as hyperlinked Web pages, social networks, communication networks, biological networks and financial transaction networks. They are used in various applications such as classifying Web pages [223], scientific research papers [224], [225], protein interaction and gene expression data [226]. We demonstrate that a similar class of networked data (i.e. networked tasks) can also provide valuable information for predicting delays in software projects. For example, if a task blocks another task and the former is delayed, then the latter is also at risk of getting delayed. This example demonstrates a common *delay prop-*

agation phenomenon in (software) projects, which has not been considered by previous approaches.

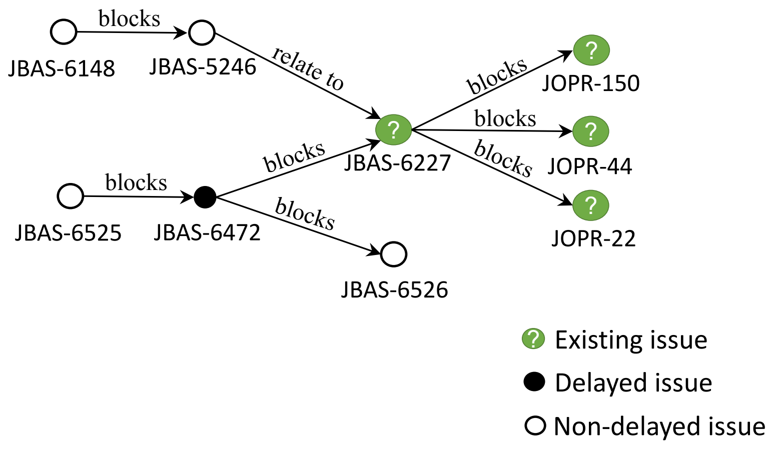


Figure 5.1: An example of task dependencies in the JBoss project

For example, Figure 5.1 shows nine issues (represented by their ID) extracted from the JBoss[a] project, only four of which (i.e. issues JBAS-6148, JBAS-5246, JBAS-6525, and JBAS-6526) were completed on time whilst there was one delayed issue (i.e. JBAS-6472). One of the main challenges in project management is therefore predicting which issues have a risk of being delayed, giving the current situation of a project, in order to come up with measures to reduce or mitigate such a risk. In this example, assume that we are trying to predict if issues JBAS-6227, JOPR-150, JOPR-44, and JOPR-22 will be delayed. In most cases, the issues in a project are however related to each other, and the delay status (e.g. major delayed, minor delayed or non-delayed) of one issue may have an influence on that of its related issue. For example, there are several "blocking" relationships between the nine JBoss issues in Figure 5.1, e.g. issue JBAS-6525 blocks JBAS-6472, indicating that the former needs to be finished before the completion of the latter. The issue dependencies form the *networked data* which contain interconnected issues. Networked data provides additional, valuable information which can be used to improve the predictive performance of techniques solely relying local features. For example, the local features are not sufficiently to provide accurate prediction for issue JBAS-6227 – it was predicted as non-delayed but it is in fact a delayed issue. On the other hand, by examining its relationships with other issues whose delay status are known – JBAS-6227 were blocked by one delayed issue and relate to 1 non-delayed issue (see Figure 5.1) – we may be able to *infer* the risk of issue JBAS-6227 being delayed.

[a]`http://www.jboss.org`

This example motivates the use of networked data to make *within-network estimation*. Here, the data has an important characteristics: issues with known delay status are useful in two aspects. They serves not only as training data but also as *background knowledge* for the inference process. Hence, the traditional separation of data into training and test sets need to carefully take this important property into account. In addition, networked issues support *collective classification*, i.e. the delay status of various related issues can be predicted simultaneously. For example, the prediction of issue JBAS-6227 can be used to influence the estimation of issue JOPR-150, JOPR-44, and JOPR-22 as they are linked, thus we should do both predictions at the same time.

In this chapter, we propose a novel approach to leverage issue dependencies for predicting delays in software projects. This chapter makes two main contributions:

- **A technique for constructing an issue network of software development tasks**

  This technique enabled us to extract various relationships between issues in software projects to build an issue network. Issue relationships can be explicit (those that are explicitly specified in the issue records) or implicit (those that need to be inferred from other issue information). Explicit relations usually determine the order of issues, while implicit relations reflect other aspects such as issues assigned to the same developer, issues affecting the same software component or similar issues.

- **Predictive models to predict delays using an issue network.**

  We developed accurate predictive models that can predict whether an issue is at risk of getting delayed. Our predictive models have three components: *local classifier*, *relational classifier* and *collective inference*. The local classifier uses non-relational (i.e. local) features of an issue: discussion time, waiting time, type, number of repetition tasks, percentage of delayed issues that a developer involved with, developer's workload, priority, number of comments, changing of priority, number of fix version, number of affect version, changing of description, number of votes, number of watches and reporter reputation. The relational classifier makes use of the issue relations in an issue network to predict if an issue gets delayed based on the delay information of its neighbors. Finally, collective inference allows us to make such a prediction *simultaneously* for multiple related issues.

The remainder of this chapter is organized as follows. Section 5.1 discusses a conceptual framework of our approach. Section 5.2 serves to describe how an issue network is built for a software project. Section 5.3 presents our networked predictive models. We provide the description of the dataset for our empirical study and evaluation in Section

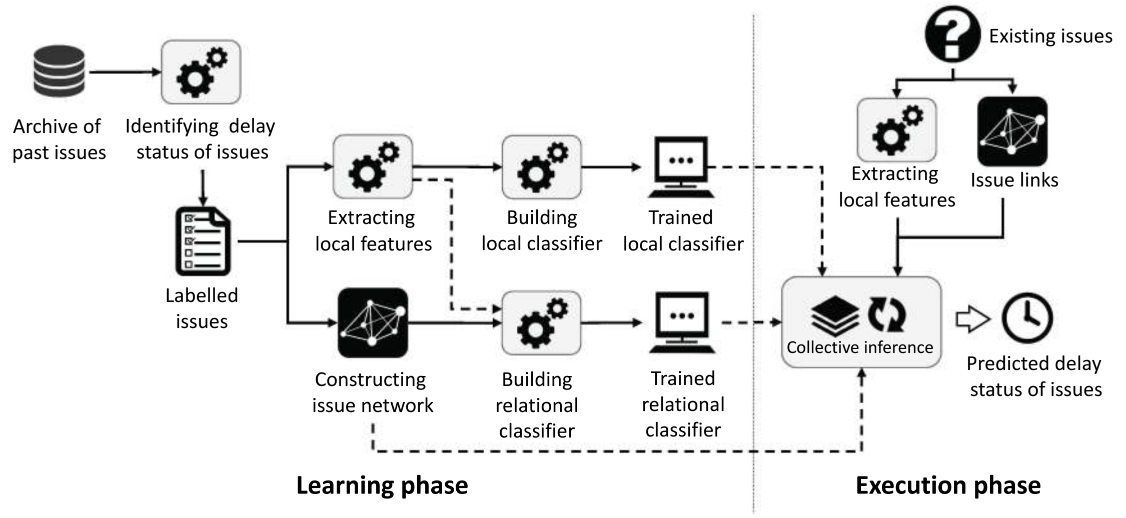

Archive of past issues
Identifying delay status of issues
Labelled issues
Extracting local features
Building local classifier
Trained local classifier
Constructing issue network
Building relational classifier
Trained relational classifier
Existing issues
Extracting local features
Issue links
Collective inference
Predicted delay status of issues
Learning phase
Execution phase

Figure 5.2: An overview of our approach

5.4. Section 5.5 reports the experimental evaluations of our approach. Related work is discussed in Section 5.6 before we provide the conclusion in Section 5.7.

5.1 Approach

Our approach leverages classification techniques in machine learning to predict the riskiness of a issue being delayed. A given issue is classified into one of the classes in $\{c_1, c_2, ..., c_k\}$ where each class c_i represents the risk impact in terms of the degree of delay, e.g. major delayed, minor delayed or non-delayed. Historical (i.e. completed) issues are labeled, i.e. assigned to a class membership, based on examining the difference between their actual completion date and due date. For example, in our studies issues completed by their due date are labeled as "non-delayed", whilst issues finished more than 60 days after their due date are labeled as "major delayed".

The basic process of our approach is described in Figure 5.2. The process has two main phases: the learning phase and the execution phase. The learning phase involves using historical data from past issues to train classifiers, which are then used for classify new issues in the execution phase. Our approach extracts data associated with software tasks to build an issue network which is defined as below.

Definition 4 (Issue network) *An issue network is a directed graph G = (V, E) where:*

- *each vertex $v \in V$ representing an issue in the form of $\langle ID, c, attrs \rangle$ where ID is a unique identifier of the issue, c is the risk class, i.e. label (e.g. non-delayed, minor*

Table 5.1: Local features of an issue

Feature	Short description
Discussion time	The period that a team spends on finding solutions to solve an issue
Waiting time	The time when an issue is waiting for being acted upon
Type	Issue type
Task repetition	The number of times that an issue is reopened
Priority	Issue priority
Changing of priority	The number of times an issue's priority was changed
No. comments	The number of comments from developers during the discussion time
No. fix versions	The number of versions for which an issue was or will be fixed
No. affect versions	The number of versions for which an issue has been found
Changing of description	The number of times in which the issue description was changed
Reporter reputation	The measurement of the reporter reputation
Developer's workload	The number of opened issues that have been assigned to a developer at a time
Per. of delayed issues	The percentage of delayed issues in all of the issues which have been assigned to a developer

delayed or major delayed), which the issue belongs to, and attrs is a set of the issue's attribute-value pairs $(attr_i, val_i)$ *(i.e. local features).*

- *each edge* $e \in E$ *representing a link between issues u and v in the form of* $\langle\langle u,v\rangle$ $,types,weights\rangle$ *where types is set of the link's type and weigths is set of link's weight.*

The set of issues (or nodes) V in an issue network is further divided into two disjoint groups: issues with known class labels, V^K, and issues whose labels need to be estimated (unknown class), V^U which $V^U = V \setminus V^K$. Labeled issues are used for training, and also serve as background knowledge for inferring the label of issues in V^U.

A set of attributes ($attrs$) for an issue are also extracted (see Table 5.1). These features represents the local information of each individual issue in the network. The local features are used to build a *local classifier* which treats issues independently from each other. Traditional state-of-the-art classifiers (e.g. Random Forest [42], Multiclass Support Vector Machines [227], or Multiclass Logistic Regression [228]) can be employed for this purpose.

The second important component in our approach is the *relational classifier*. Unlike the local classifier, the relational classifier makes use of the relations between issues in the network (represented by edges) to estimate an issue's label using the labels of its neighbors. Relational classifier models exploit a phenomenon that is widely seen in relational data: the label of a node is influenced by the labels of its related nodes. Relational classifier models may also use local attributes of the issues. Links between issues in the network are established by extracting both explicit and implicit relations. Explicit relations refer to the issue dependencies explicitly set by the developers (e.g. the block dependency). On the other hand, implicit relations can be inferred from the resources assigned to the issues (e.g. assigned to the same developer) or the nature of the issues. We will discuss these types of issue relations in details in Section 5.2. Each type of relationship can be assigned to a weight which quantitatively reflects the strength of the relationship.

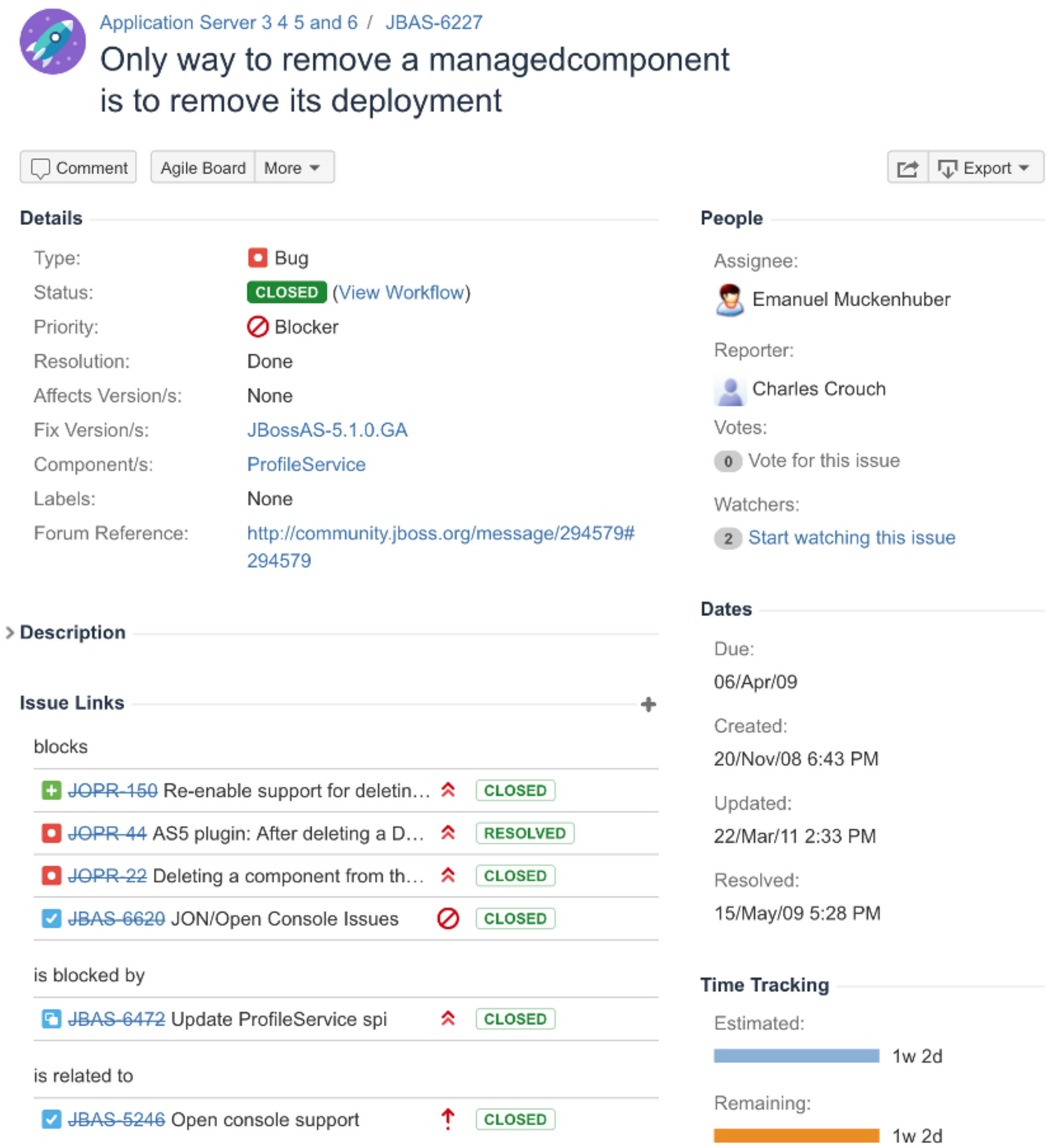
Another novel aspect of our approach is the *collective inference* component which *simultaneously* classifies a set of related issues. Details of these approaches will be provided in Section 5.3.

5.2 Issue network construction

An important part of our approach is building an issue network for past and current issues. In most of modern issue tracking system (e.g. JIRA), some dependencies between issues are explicit recorded (i.e. in a special field) in the issue reports and can be easily extracted from there. Figure 5.3 shows an example of issue JBAS-6227 in the JBoss project (from our motivating example in Section 5). This issue is blocked by 1 issues (i.e. JBAS-6472) and it also blocks 4 issues (JOPR-150, JOPR-44, JOPR-22, and JBAS-6620) from resolving. We refer to these dependencies as *explicit relationships*. There are however other types of issue dependency that are not explicitly recorded (e.g. issues assigned to the same developer), and we need to infer them from extracting other information of the issues. These are referred to as *implicit relationships*. We now discuss these types of relationships in details.

Explicit relationships

There are a number of dependencies among issues which are explicitly specified in the issue reports. These typically determine the order in which issues need to be performed.



Application Server 3 4 5 and 6 / JBAS-6227

Only way to remove a managedcomponent is to remove its deployment

Details

Type:	Bug
Status:	CLOSED (View Workflow)
Priority:	Blocker
Resolution:	Done
Affects Version/s:	None
Fix Version/s:	JBossAS-5.1.0.GA
Component/s:	ProfileService
Labels:	None
Forum Reference:	http://community.jboss.org/message/294579#294579

Description

Issue Links

blocks

~~JOPR-150~~ Re-enable support for deletin… CLOSED
~~JOPR-44~~ AS5 plugin: After deleting a D… RESOLVED
~~JOPR-22~~ Deleting a component from th… CLOSED
~~JBAS-6620~~ JON/Open Console Issues CLOSED

is blocked by

~~JBAS-6472~~ Update ProfileService spi CLOSED

is related to

~~JBAS-5246~~ Open console support CLOSED

People

Assignee:
Emanuel Muckenhuber

Reporter:
Charles Crouch

Votes:
0 Vote for this issue

Watchers:
2 Start watching this issue

Dates

Due:
06/Apr/09

Created:
20/Nov/08 6:43 PM

Updated:
22/Mar/11 2:33 PM

Resolved:
15/May/09 5:28 PM

Time Tracking

Estimated:
1w 2d

Remaining:
1w 2d

Figure 5.3: Example of an issue report with issue links

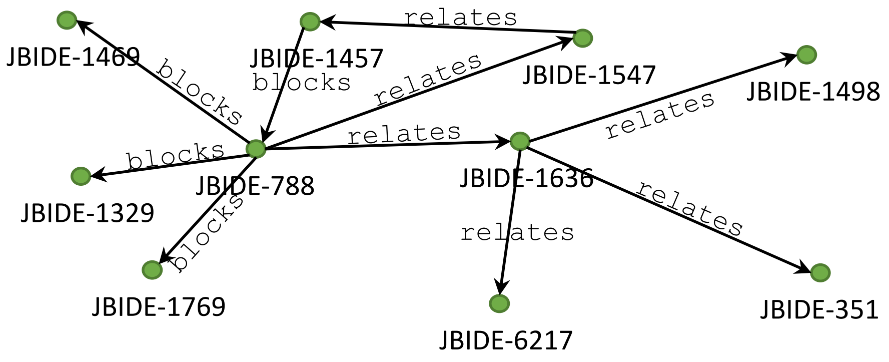


Figure 5.4: Example of explicit issue relationships in JBoss

There are generally four different types of relationships of the preceding tasks to the succeeding tasks: finish to start (predecessor must finish before successor can start), start to start (predecessor must start before successor can start), finish to finish (predecessor must finish before successor can finish), and start to finish (predecessor must start before successor can finish). For example, blocking is a common type of relationships that is explicitly recorded in issue/bug tracking systems. Blocking issues are issues that prevent other issues from being resolved, which could fall into the finish to start or finish to finish category.

Figure 5.4 shows some explicit relationships between issues in the JBoss project, which uses the JIRA issue tracking system. JIRA provides the issue link attribute to specify the relationship between two or more related issues. The explicit relationships are extracted directly from the dataset. For example, JBIDE-788 blocks JBIDE-1469, which is represented by a directed edge connected the two nodes. In addition to blocking, JIRA also provides three other default types of issue links: relates to, clones and duplicates. Figure 5.4 shows some examples of the "relates to" relationship, e.g. issue JBIDE-788 relates to JBIDE-1547.

Implicit relationships

While explicit relationships are specified directly in the issue reports, implicit relationship need to be inferred from other issue information. There are different issue information that can be extracted to identify a (implicit) relationship between issues. We classified them into three groups as described below.

- **Resource-based relationship:** this type of relationships exists between issues that share the same (human) resource. The resource here could be the developers as-

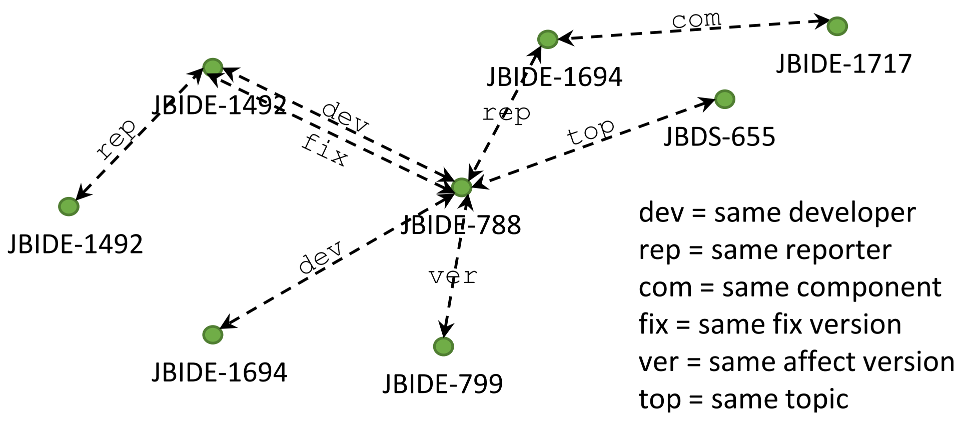


Figure 5.5: Example of implicit issue relationships in JBoss

signed to perform the tasks or the same person who created and reported the issues. Resource-based relationship is important in our context since a resource's skills, experience, reputation and workload may affect a chance of delayed issues (i.e. a developer who causes a delay of a current issue may do so again in the future). For example, from Figure 5.5, JBIDE-788 has a relationship with JBIDE-1694 since both of them are assigned to the same developer. Issue JBIDE-788 is also related to JBIDE-1694 since they were reported by the same person.

- **Attribute-based relationship:** issues can be related if some of their attributes share the same values. For example, there is a relationship between issues related to the same component since they may affect the same or related parts of code. For issue reports recorded in JIRA, we extract this type of relationship by examining three attributes: *affect version*, *fix version* and *component*. For example, issue JBIDE-788 and JBIDE-799 affects the same version while JBIDE-1694 and JBIDE-1717 affects the same component as shown in Figure 5.5.

- **Content-based relationship:** issues can be similar in terms of how they are conducted and/or what they affect. The similarity may form an implicit relationship between issues which can be established by extracting the description of the issues. Different extraction techniques can be applied here, ranging from traditional information retrieval techniques to recent NLP techniques like topic modeling. We use Latent Dirichlet Allocation (LDA) [197] to build a topic model representing the content of an issue description. We then establish relationships between on the basis that related issues share a significant number of common topics. Figure 5.5 shows some example of content-based relationships in JBoss, e.g. issue JBIDE-788 has the same topic with JBDS-655. The common topics shared between these two issues are “code, access control exception, and document types”.

5.3 Predictive model

Our predictive models are built upon three components: local classifier (as done in previous work), relational classifier, and collective inference. Local classifiers treat issues as being independent, making it possible to estimate class membership on an issue-by-issue basis. Relational classifiers posit that the class membership of one issue may have an influence on the class membership of a related issue in the network. Collective inference infers the class membership of all issues simultaneously [229]. In the following we discuss the details of each components.

5.3.1 Local (non-relational) classifier

There are several available state-of-the-art algorithms and techniques that we could employ to develop local classifiers. We employ the state-of-the-art classifier which is Random Forest (RF) [42] – the best performing technique in our experiments.

5.3.2 Relational classifier

Relational classifiers make use of information about issue links to estimate the label probability. For simplicity, we use only direct relations for class probability estimation:

$$P(c \mid G) = P(c \mid N_i)$$

where N_i is a set of the immediate neighbors of issue v_i (i.e. those that are directly related to v_i) in the issue network G, such that $P(c \mid N_i)$ is independent of $G \setminus N_i$. This is based on a theoretical property known as the Markov assumption which states that given the neighborhood (also known as the Markov blanket), it is sufficient to infer about the current label without knowing the other labels in the network [230].

For developing a relational classifier, we employ two highly effective methods. One is Weighted-Vote Relational Neighbor (wvRN) [231] which is one of the best relational classification algorithms reported in [229]. The other is Stacked Graphical Learning

[232], where classifiers are built in a stage-wise manner, making use of relational information in the previous stage.

Weighted-Vote Relational Neighbor

Weighted-Vote Relational Neighbor (wvRN) estimates class membership probabilities based on two assumptions [233]. First, the label of a node depends only on its immediate neighbors. Second, wvRN relies on the principle of homophily which assumes that neighboring class labels were likely to be the same [234]. Thus, wvRN estimates $P(c|v_i)$ as the (weighted) mean of the class membership of the issues in the neighborhood (N_i):

$$P(c \mid v_i) = \frac{1}{Z} \sum_{v_j \in N_i} w(v_i, v_j) P(c \mid N_j)$$

where $Z = \sum_{v_j \in N_i} w(v_i, v_j)$ and $w(v_i, v_j)$ is the weight of the link between issue v_i and issue v_j. Our experiments applied the same weight of 1 to all relationship types, i.e. $w(v_i, v_j) = 1$. The optimized weights could be determined using the properties of a network topology such as assortativity coefficient [229], [235], [236]). We denote the prior class probability distributions from a relational classification as M_R.

Stacked Graphical Learning

One inherent difficulty of the weighted-voting method is the computation of the neighbor weights. Since there are multiple relations, estimating the weights are non-trivial. Stacked learning offers an alternative way to incorporate relational information.

The idea of stacking is to learn joint models by multiple steps, taking into relational information of the previous step to improve the current step. At each step, relational information together with local features are fed into a standard classifier (e.g., Random Forests). We consider relations separately and the contribution of each relation is learnt by the classifier through the relational features. The classifier is then trained. Its prediction on all data points (vertices in the network) will be then used as features of the next stage. We adapt the idea from [232]. Our contribution is in the novel use of Random Forests as a strong local classifier rather than linear classifiers as used in [232].The stacked learning algorithm is described in Algorithm 1. It returns T classifiers for Tsteps. At the first step, the local classifier is used. At subsequent steps, relational classifiers are trained on both local features and relation-specific averaged neighbor probabilities.

Algorithm 1 The stacked learning algorithm (adapted from [232])

```
1: Train of the 1-st local classifiers on training nodes, ignoring relations
2: for step t=2,3,..,T do
3:     Compute the class probabilities for all data points using the (t-1)th classifier
4:     for each node $i$ do
5:         for each relation $r$ that this node has with its neighbor do
6:             if relation weight exist then
7:                 Average all probabilities of its neighbors $j$ who have the relation $r$ with relation weight
8:             else
9:                 Set relation weight to 1
10:                Average all probabilities of its neighbors $j$ who have the relation $r$
11:            end if
12:            Prepare $k-1$ features using these averaged probabilities (k probabilities sum to 1)
13:        end for
14:        Concatenate all relational features together with the original features
15:    end for
16:    Train the t-th local classifier on *training nodes* and new feature sets.
17: end for
18: Output T classifiers (one local, T-1 relational)
```

5.3.3 Collective inference

Collective inference is the process of inferring class probabilities simultaneously for all unknown labels in the network conditioned on the seen labels. We employ two methods: Relaxation Labeling (RL) [237] and Stacked Inference (SI). RL is applicable to any non-stagewise relational classifiers (e.g. wvRN described in Section 5.3.2). It has been found to achieve good performance in [229]. SI, on the other hand, is specific to stacked classifiers (e.g., see Section 5.3.2).

Relaxation Labeling

Relaxation Labeling (RL) has been shown to achieve good results in [229]. RL initializes the class probabilities using the local classifier model. RL then iteratively corrects this initial assignment if the neighboring issues have labels that are unlikely according to the prior class distribution estimated by M_R (see Section 5.3.2). Algorithm 2 describes the Relaxation Labeling technique.

Algorithm 2 The Relaxation Labeling algorithm (adapted from [233])

```
1: Use the 1-st classifier to predict the class probabilities using only local features
2: for step t=2,3,..,T do
3:     Estimate the prior class probabilities using the relational classifier, $M_R$, on the current state of network
4:     Reassign the class of each $v_i \in V^U$ according to the current class probabilities estimation
5: end for
6: Output the class probabilities of vertices with unknown labels.
```

Stacked Inference

Following the stacked learning algorithm in Section 5.3.2, stacked inference is described in Algorithm 3. It involved T classifiers returned by the stack learning algorithm. At the first step, the local classifier is used to compute the class probabilities. At $T-1$ subsequent steps, relational classifiers receives both the local features and relation-specific weighted neighbor probabilities and outputs class probabilities. The final class probabilities are the outcome of the inference process.

Algorithm 3 The stacked inference algorithm

```
1: Use the 1-st classifier to predict the class probabilities using only local features
2: for step t=2,3,..,T do
3:     Prepare relational features using the neighbor probabilities computed from the previous step
4:     Use the t-th classifiers to predict the class probabilities using local features and relational features.
5: end for
6: Output the class probabilities of vertices with unknown labels.
```

5.4 Dataset

In this study, the prediction task is similar to the previous chapter (i.e. predicting issue delays). Since we use the issue reports recorded in JIRA platform, we refer to Section 4.5 for data collecting, labeling, and preprocessing. However, this study makes use of the issue links (i.e. the explicit relationships) recorded in issue reports. This information of issue links can be also obtained from the collected JSON files. Figure 5.6 shows an example of issue links of issue *JBAS-6227* from the JBoss repository. For example, issue *JBAS-6227* blocks issues *JOPR-150*, *JOPR-44*, and *JOPR-22*, and it is blocked by issue

JBAS-6472. Issues were collected from the JIRA issue tracking system in five well-known open source projects: Apache, Duraspace, JBoss, Moodle, and Spring.

```
{"key": "JBAS-6227", ...
"fields": {
   "issuelinks": [{
   "type": {
      "name": "Dependency",
         "inward": "blocks",
         "outward": "is blocked by",},
      "outwardIssue": {
         "key": "JBAS-6472", ...}
      "inwardIssue": {
         "key": "JOPR-150", ...
         "key": "JOPR-44", ...
         "key": "JOPR-22", ...}
},...,}
```

Figure 5.6: Example of issue links in an issue JSON file

An issue network is built by extracting both explicit and implicit links among the issues. We employ a number of measures to describe different properties of an issue network: the number of nodes, the number of edges, and the average node degree (i.e. the number of connections a node has to other nodes). In addition, assortativity coefficient [138] is used to measure the correlation between two nodes: the preference of network nodes to connect to other nodes that have similar or different degrees. Positive values of assortativity coefficient indicate a correlation between nodes of similar degree (e.g. highly connected nodes tends to be connected with other high degree nodes), while negative values indicate relationships between nodes of different degree (e.g. high degree nodes tend to connect to low degree nodes). As can be seen from Table 5.2, the inclusion of implicit relationships significantly increases the density of the network issues across all the five projects that we studied. By contrast, the assortativity coefficient remains nearly the same with or without implicit relationships.

A weight is also applied to each edge type in an issue network. This allows us to better quantify the strength of a relationship between issues. By default, each edge is equally assigned the weight of 1. However, different weights can also be applied to different types of relationships. More complex approaches can also be applied here. For example, the weights could be decreased over time to reflect the fading of the relationships, e.g. the issues have been assigned to the same developer for long time ago.

Table 5.2: Datasets and networks' statistics

Project	Relationship	Num Nodes	Num Edges	Avg. node degree	Node Assort.
Apache	Explicit	496	246	1.597	0.256
	Implicit	496	27,460	55.362	0.246
	All	496	27,706	55.858	0.225
Duraspace	Explicit	1,116	563	1.700	0.257
	Implicit	1,116	383,677	343.796	0.240
	All	1,116	384,240	344.301	0.230
JBoss	Explicit	8,206	4,904	2.057	0.235
	Implicit	8,206	4,908,164	598.118	0.249
	All	8,206	4,913,068	598.716	0.247
Moodle	Explicit	1,439	1,283	3.055	0.222
	Implicit	1,439	197,176	137.022	0.215
	All	1,439	198,748	138.115	0.208
Spring	Explicit	597	222	1.219	0.250
	Implicit	597	63,430	106.247	0.249
	All	597	63,652	106.619	0.242

5.5 Evaluation

Our empirical evaluation aims to answer the following research questions:

RQ1 *Does the networked classification techniques improve the predictive performance?*

We perform an experiment to compare the predictive performance between traditional classifiers (i.e. Random Forrests) and the two networked classifiers (i.e. Weighted-Vote Relational Neighbor with Relaxation Labeling, and stacking method) on five projects: Apache, Duraspace, JBoss, Moodle, and Spring to find the best performer.

RQ2 *Does collective inference improve the predictive performance of relational classifiers?*

This research question focuses on evaluating the predictive performance achieved by using collective inference on relational classifiers. To do so, we setup two experiments: one using Weighted-Vote Relational Neighbor and the other using both Weighted-Vote Relational Neighbor and Relaxation Labeling.

RQ3 *Does using different relationship types affect the predictive performance?*

We perform a number of experiments to evaluate the predictive performance achieved by different sets of issue's relationships. We test with five different combinations: networks with explicit relationships, networks with explicit and resource-based relationships, networks with explicit and attribute-based relationships, networks explicit and content-based relationships, and networks with all explicit and implicit relationships.

RQ4 *How does the size of training data affect the predictive performance?*

We aim to assess the proportion of past issues (i.e. labeled issues) is needed to achieve a good predictive performance. To answer this question, we vary the number of issues between 20% to 100% of the total issues in the training set for developing the predictive models.

5.5.1 Experimental setting

The dataset was divided into a training set and a test set (see Table 5.3). We try to mimic a real project management scenario that prediction on a current issue is made using knowledge from the past issues, the collected issues in training set are those that were opened before the issues in test set. The collected datasets are shown in Table 5.2. Since the number of delayed issues in our datasets is small, we chose to use two classes of delay: major delayed and minor delayed (and the non-delayed class).

Table 5.3 shows the number of issues in training set and test set for each project. Major delayed issues are those that have actual completed date (resolved date) greater than 30 days from planned to completed date and less than 30 days of delays is minor delayed. Note that the size of delayed can be defined by project managers who realize the impact of schedule overruns to the projects. Since (major/minor) delayed issues are rare and imbalanced, we had to be careful in creating the training and test sets. Specifically, we placed 60% of the delayed issues into the training set and the remaining 40% into the test set. In addition, we tried to maintain a similar ratio between delayed and non-delayed issues in both test set and training set, i.e. stratified sampling.

5.5.2 Performance measures

Reporting the average of precision/recall across classes is likely to overestimate the true performance, since our risk classes are ordinal and imbalanced and no-delays are the de-

Table 5.3: Experimental setting

Project	Training set			Test set		
	Major	Minor	Non	Major	Minor	Non
Apache	10	52	236	6	34	158
Duraspace	23	71	575	16	47	384
JBoss	666	679	3,579	444	452	2,386
Moodle	42	52	770	28	34	513
Spring	13	34	310	8	22	207

fault and they are not of interest to the prediction of delayed issues. Hence, our evaluation is focus on the predicting of risk classes as described below. A confusion matrix is used to evaluate the performance of our predictive models. As a confusion matrix does not deal with a multi-class probabilistic classification, we reduce the classified issues into two binary classes: delayed and non-delayed using the following rule:

$$C_i = \begin{cases} delayed, & if P(i,Maj) + P(i,Min) > P(i,Non) \\ non-delayed, & otherwise \end{cases}$$

where C_i is the binary classification of issue i, and $P(i,Maj)$, $P(i,Min)$, and $P(i,Non)$ are the probabilities of issue i classified in the major delayed, minor delayed, and non-delayed classes respectively. Basically, this rule determines that an issue is considered as delayed if the sum probability of it being classified into the major and minor delayed classes is greater than the probability of it being classified into the non-delayed class. Note that our work on this chapter focuses on predicting delayed and non delayed issues. Our evaluations thus emphasize on measuring the performance of predicting whether issues will cause a delay. We however acknowledge that the ability to distinguish between major and minor is also important. Hence, future work involves using several appropriate performance metrics (e.g. Macro-averaged mean absolute error [158]) to measure the performance of our models in distinguishing between the two delayed classes (major and minor delayed).

The confusion matrix is then used to store the correct and incorrect decisions made by a classifier. Those values are used to compute the Precision, Recall, F-measure, and AUC for the delayed issues to evaluate the performance of the predictive models (see Section 2.1.6).

Table 5.4: Evaluation results of traditional classification, wvRN+RL, and stacked learning

Project	Method	Precision	Recall	F-measure	AUC
Apache	Traditional	0.38	0.65	0.48	0.75
	Collective inference	0.46	0.75	0.57	0.78
	Stack learning	**0.62**	**0.85**	**0.72**	**0.83**
Duraspace	Traditional	0.43	0.71	0.54	0.80
	Collective inference	**0.97**	0.46	0.62	0.92
	Stack learning	0.63	**0.83**	**0.71**	**0.95**
JBoss	Traditional	0.44	0.70	0.54	0.68
	Collective inference	0.46	0.70	0.56	0.78
	Stack learning	**0.59**	**0.83**	**0.69**	**0.82**
Moodle	Traditional	0.21	0.45	0.29	0.64
	Collective inference	0.46	**0.80**	0.57	0.79
	Stack learning	**0.83**	0.65	**0.73**	**0.89**
Spring	Traditional	0.51	0.60	0.55	0.91
	Collective inference	**0.74**	0.77	**0.75**	0.84
	Stack learning	0.55	**0.97**	0.70	**0.94**
Avg	Traditional	0.39	0.62	0.48	0.76
	Collective inference	0.62	0.70	0.61	0.82
	Stack learning	**0.64**	**0.82**	**0.71**	**0.89**

5.5.3 Results

Comparison of different classification approaches (RQ1)

We compare three different settings: local classifier using Random Forrests (traditional classification), Weighted-Vote Relational Neighbor (wvRN) with Relaxation Labeling (RL), and stacking method (with stacked inference). Table 5.4 shows the precision, recall, F-measure, and AUC achieved by three different classification approaches in each project and averaging across all projects. Note that the stacking method uses Random Forests as the base classifier.

The evaluation results indicate that the predictive performance achieved by stacked learning is better and more consistent than traditional classification and relational classification using wvRN+RL. Stacked learning achieved the best precision of 0.64 (averaging across five projects), while the traditional classification achieved only 0.39 precision (averaging across five projects). It should however be noted that wvRN+RL achieved the highest precision of 0.97 for Duraspace. In addition, the precision achieved by stacked learning is more consistent and steady in all projects. By contrast, the performance of wvRN+RL are varied between projects. Relational classification with wvRN+RL is based on the principle of homophily, which may not always hold in some projects. This is reflected by its low performance in some cases (i.e. only 0.46 precision for Apache).

On the other hand, stacked learning provides a more generalized approach to learn the relationships within networked data – it achieved above 0.5 precision across the five projects.

Stacked learning also outperforms the other classification approaches in terms of recall and F-measure: it achieved the highest recall of 0.82 and the highest F-measure of 0.71 (averaging across five projects). The highest recall of 0.97 was also achieved by stack learning for the Spring project. The degree of discrimination achieved by our predictive models is also high, as reflected in the AUC results. The AUC quantifies the overall ability of the discrimination between the delayed and non-delayed classes. The average of AUC across all classifiers and across all projects is 0.83. All classifiers achieved more than 0.65 AUC while stacked learning is the best performer with 0.88 AUC (averaging across five projects) and 0.95 for Duraspace.

Overall, the evaluation results demonstrate the effectiveness of our predictive models, achieving on average 46%–97% precision, 46%–97% recall, 56%–76% F-measure, and 78%–95% Area Under the ROC Curve. Our evaluation results also show a significant improvement over traditional approaches (local classifiers): 49% improvement in precision, 28% in recall, 39% in F-measure, and 16% in Area Under the ROC Curve.

Answer to RQ1: Using the networked classification techniques significantly improve the predictive performance over traditional approaches. Stacked learning (stacking method with stacked inference) is the best performer with respect to all performance measures.

The usefulness of collective inference (RQ2)

Table 5.5 shows the comparison of the precision, recall, F-measure, and AUC achieved by the relational classification with collective inference (wvRN+RL) and without collective inference (only wvRN). Overall, the predictive performance achieved by relational classification with collective inference is better than that without collective inference in all measures. The relational classification with collective inference achieves the highest precision of 0.62, recall of 0.70, F-measure of 0.62, and 0.83 AUC (averaging across five projects). Although, the predictive performance of Relaxation Labeling is lower than stacked learning as we discussed earlier, the evaluation results still support that collective inference significantly improve the performance of relational classifiers. However, collective inference applied on top of the wvRN still follows a strong assumption of homophily

Table 5.5: Evaluation results of relational classifier with collective inference and without collective

Project	Method	Precision	Recall	F-measure	AUC
Apache	Non CI	0.46	0.75	0.57	0.78
	With CI	**0.48**	**0.77**	**0.59**	**0.80**
Duraspace	Non CI	0.88	**0.48**	**0.62**	0.91
	With CI	**0.97**	0.46	**0.62**	**0.92**
JBoss	Non CI	**0.48**	**0.71**	**0.57**	**0.82**
	With CI	0.46	0.70	0.56	0.78
Moodle	Non CI	0.10	0.11	0.10	0.65
	With CI	**0.46**	**0.80**	**0.57**	**0.79**
Spring	Non CI	0.74	0.77	0.75	0.87
	With CI	**0.76**	**0.78**	**0.76**	**0.88**
Avg	Non CI	0.53	0.56	0.52	0.81
	With CI	**0.62**	**0.70**	**0.62**	**0.83**

theory and as a result, it causes an inconsistent predictive performance among different projects.

Answer to RQ2: Combining collective inference and relational classifiers significantly improves the predictive performance.

The influence of explicit and implicit relationships (RQ3)

As can be seen from Figure 5.7, the highest predictive performance is achieved by using both explicit and implicit relationships: it achieved the highest precision of 0.62 and the highest recall of 0.70 (averaging across five projects). By contrast, the networks using only explicit relationships achieved the lowest precision, i.e. 0.31, while the networks using explicit and content-based relationships produced the lowest recall. In general, using both explicit relationships and implicit relationships (resource-based, attribute-based, and content-based) significantly increases the predictive performance: 66.23 % increased in precision and 21.62 % increased in recall (compare to using only explicit relationships).

Answer to RQ3: The predictive models using both explicit and implicit relationships perform best.

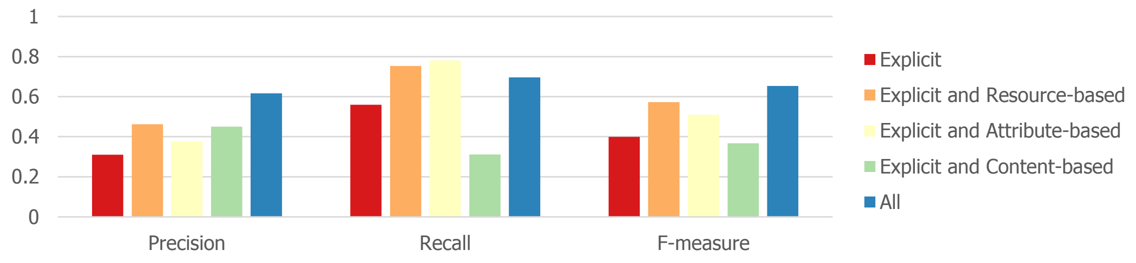


Figure 5.7: Evaluation results on different sets of relationships

The effect of the size of training data (RQ4)

In these experiments, given a data set, $G = (V, E)$, V^K (i.e. labeled issues) is created by selecting samples of 20% – 100% of the training set (see Table 4.3). The test set, V^U, is then defined as $V \setminus V^K$. Figure 5.8 shows the predictive performance from samples of 20% to 100% of V in terms of F-measure. The results clearly demonstrate that F-measure (averaging across five projects) increases as more labeled data is used for training.

Answer to RQ4: Using more labeled data for training improves the predictive performance of the model.

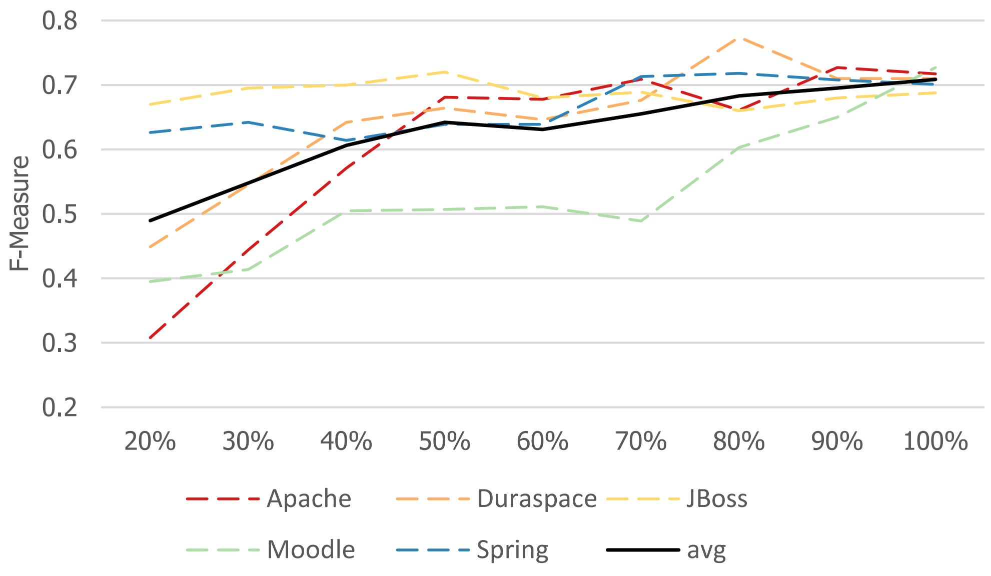


Figure 5.8: Evaluation results on different sizes of training data

5.5.4 Threats to validity

One relational setting involves the use of wvRN, which assumes the homophily property among issues, that is, related issues should have similar delay risk. This is a strong assumption and may not hold in reality, and this has been revealed in our experiments. We have addressed this threat by proposing stacked learning approach which does not rely on the the homophily assumption but rather estimates the contribution of separate relationships.

We have attempted to identify all possible relationships among issues. However, we acknowledge that the implicit relationships we have inferred are by no means comprehensive to represent all issue dependencies. Another threat to our study is that our data set has the class imbalance problem (over 90% of the total data are non-delayed issues), which may affect a classifier's ability to learn to identify delayed issues. We have used stratified sampling to mitigate this problem. We however acknowledge such a sampling approach could be an external threat to validity. Further experiments to evaluate sampling techniques used in practice are thus needed. In addition, patterns that hold in the train data may not reflect the situation in the test, e.g. the team and management having changed their approach or managed the risks they perceived. To address this threat, instead of splitting the data randomly (as done in traditional settings), we deliberately chose the time to split training and test sets to mimic a real deployment.

We have considered 11,851 issue reports from the five projects which differ significantly in size, complexity, development process, and the size of community. Although these are real data, we however cannot claim that our data set would be representative of all kinds of software projects, especially in commercial settings. Although open source projects and commercial projects share similarities in many aspects, they are also different in the nature of contributors, developers and project's stakeholders. For example, open source contributors are free to join and leave the communities (i.e. high turn over rate), while developers in the commercial setting tend to be stable and fully commit to deliver the project's progress. Hence, further study is need to understand how our predict models perform for commercial projects.

5.6 Related work

In this section, we discuss the thread of related work that makes use of networked data in software projects. Zimmermann et al. [181] proposed a defect prediction model using network analysis approach on (directed-)dependency graphs built from source code. They constructed the program dependencies as a directed relationship between two pieces of code using two different types of dependencies: data dependencies (i.e. the relationships between the definition and the value passing) and call dependencies (i.e. the relationships between the functions and the locations that they were called). Their prediction model is then built based on several metrics (i.e. features) from applying network analysis techniques on the constructed dependency graphs for example size, tie, density, and shortest paths. They have reported that the using of the network metrics improves the predictive performance by 30% over object-oriented complexity metrics. In addition, the formal method for constructing program dependencies that represent relationships between two or more pieces of code has been previously proposed by Pogdurski et al. [238] in 1990. Program dependencies have been used in several aspects in software development (e.g. testing [239], code optimization [240], and software debugging [241]).

There are several methods for constructing a developer network based on the information recorded in source code repositories have been proposed. For example, Lopez-Fernandez et al. [242] proposed an approach to construct developer networks e.g., two developers are linked when they commit to same modules. They then applied network analysis techniques on the constructed developer networks to study the patterns of developer's collaborations in software projects. Huang et al. [243] also leveraged the data from source code repositories to construct developer networks. They studied the evolutionary process of learning behavior of developers in open source software projects to classify developers into core and non-core groups. The work in [244] and [245] also makes use of developer networks for predicting whether a software building fails. We, however, proposed an approach to construct an issue network which also considers the relationships between developers as the implicit links in the network.

The networked data also used in several aspects such as predicting software quality using social network analysis (e.g. [180], [246]), predicting software evolution in terms of estimating bug severity, efforts, and defect-prone releases using Graph-based analysis (e.g. [138]).

5.7 Chapter summary

In this chapter, we have proposed a novel approach to predict whether a number of existing issues in a software project are at risk of being delayed. Our approach exploits not only features specific to individual issues but also the relationships between the issues (i.e.

networked data). We have developed several prediction models using local classifiers, relational classifiers and collective inference. The evaluation results demonstrate a strong predictive performance of our networked classification techniques compared to traditional approaches: achieving 49% improvement in precision, 28% improvement in recall, 39% improvement in F-measure, and 16% improvement in Area Under the ROC Curve. In particular, the stacked graphical learning approach consistently outperformed the other techniques across the five projects we studied. The results from our experiments indicate that the relationships between issues have an impact on the predictive performance.

Chapter 6

Story point estimation

EFFORT estimation is an important part of software project management, particularly for planning and monitoring a software project. Effort estimates may be used by different stakeholders as input for developing project plans, scheduling iteration or release plans, budgeting, and costing [247]. Hence, incorrect estimates may have adverse impact on the project outcomes [24], [27]–[29]. Research in software effort estimation dates back several decades and they can generally be divided into model-based methods, expert-based methods, and hybrid methods which combine model-based and expert-based methods [248]. Model-based approaches leverages data from old projects to make predictions about new projects. Expert-based methods rely on human expertise to make such judgements. Most of the existing work (e.g. [7], [8], [95], [96], [98], [99], [249]–[254]) focus on the effort required for completing a whole project (as opposed to user stories or issues). These approaches estimate the effort required for developing a complete software system, relying on a set of features manually designed for characterizing a software project.

Software is developed through repeated cycles (iterative) and in smaller parts at a time (incremental) in modern agile development settings. A project has a number of *iterations* (e.g. *sprints* in Scrum [126]). Each iteration requires the completion of a number of user stories. There is thus a need to focus on estimating the effort of completing a single user story at a time rather than the entire project. In fact, it has now become a common practice for agile teams to go through each user story and estimate the effort required for completing it. *Story points* are commonly used as a unit of effort measure for a user story [125]. Currently, most agile teams heavily rely on experts' subjective assessment (e.g. planning poker, analogy, and expert judgment) to arrive at an estimate. This may lead to inaccuracy and more importantly inconsistencies between estimates [30].

We propose a prediction model which supports a team by recommending a story-point estimate for a given user story. Our model learns from the team's previous story point estimates to predict the size of new issues. This prediction system will be used in conjunction with (instead of a replacement for) existing estimation techniques practiced by the team. It can be used in an completely automated manner, i.e. the team will use the story points given by the prediction system. Alternatively, it could be used as a decision support system and takes part in the estimation process. This is similar to the notions of combination-based effort estimation in which estimates come from different sources, e.g. a combination of expert and formal model-based estimates [248]. The key novelty of our approach resides in the combination of two powerful *deep learning* architectures: long short-term memory (LSTM) and recurrent highway network (RHN). LSTM allows us to model the long-term context in the textual description of an issue, while RHN provides us with a deep representation of that model. We named this approach as Deep learning model for Story point Estimation (Deep-SE). Our prediction system is end-to-end trainable from raw input data to prediction outcomes without any manual feature engineering.

We have performed an extensive evaluation on 23,313 issues. The evaluation results demonstrate that our approach consistently outperforms three common baseline estimators: Random Guessing, Mean, and Median methods and four alternatives (e.g. using Doc2Vec and Random Forests) in Mean Absolute Error, Median Absolute Error, and the Standardized Accuracy. These claims have also been tested using a non-parametric Wilcoxon test and Vargha and Delaney's statistic to demonstrate the statistical significance and the effect size.

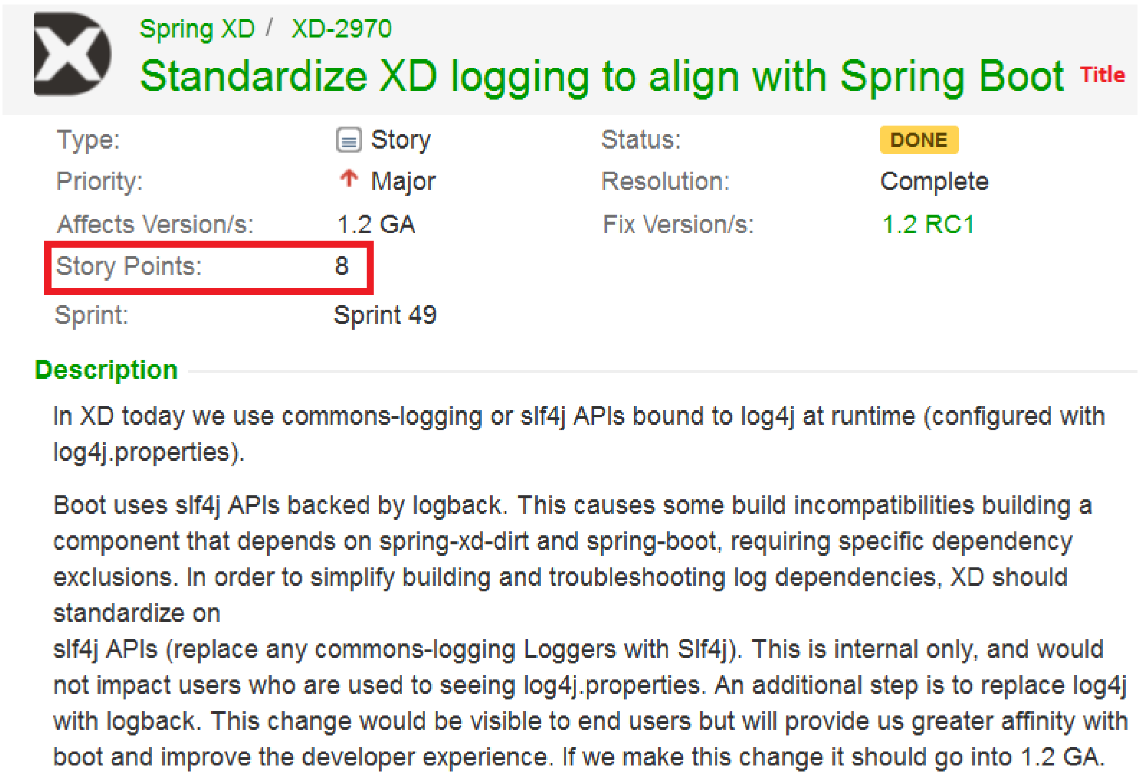
The remainder of this chapter is organized as follows. Section 6.1 provides a background of the story point estimation and the deep learning techniques. Section 6.2 presents the Deep-SE model. We also explain how it can be trained in Section 6.3. We explain how we collect the data for our empirical study and evaluation in Section 6.4. Section 6.5 reports on the experimental evaluation of our approach. Related work is discussed in Section 6.6 before we conclude our work in Section 6.7.

6.1 Story point estimation

When a team estimates with story points, it assigns a point value (i.e. story points) to each user story. A story point estimate reflects the *relative* amount of effort involved in resolving or completing the user story: a user story that is assigned two story points should take twice as much effort as a user story assigned one story point. Many projects have now adopted this story point estimation approach [30]. Projects that use issue tracking systems (e.g. JIRA[a]) record their user stories as *issues*. Figure 6.1 shows an example of issue XD-2970 in the Spring XD project[b] which is recorded in JIRA. An issue typically has a title (e.g. "Standardize XD logging to align with Spring Boot") and description. Projects that use JIRA Agile also record story points. For example, the issue in Figure 6.1 has 8 story points.

Definition 5 (Story point) *A story point estimate reflects the* relative *amount of effort involved in resolving or completing the user story: a user story that is assigned two story points should take twice as much effort as a user story assigned one story point.*

- *a story point assigned to an issue must be a numerical value*



Type:	Story	Status:	DONE
Priority:	Major	Resolution:	Complete
Affects Version/s:	1.2 GA	Fix Version/s:	1.2 RC1
Story Points:	8		
Sprint:	Sprint 49		

Description

In XD today we use commons-logging or slf4j APIs bound to log4j at runtime (configured with log4j.properties).

Boot uses slf4j APIs backed by logback. This causes some build incompatibilities building a component that depends on spring-xd-dirt and spring-boot, requiring specific dependency exclusions. In order to simplify building and troubleshooting log dependencies, XD should standardize on
slf4j APIs (replace any commons-logging Loggers with Slf4j). This is internal only, and would not impact users who are used to seeing log4j.properties. An additional step is to replace log4j with logback. This change would be visible to end users but will provide us greater affinity with boot and improve the developer experience. If we make this change it should go into 1.2 GA.

Figure 6.1: An example of an issue with estimated story points

Story points are usually estimated by the whole project team. For example, the widely-used Planning Poker [255] method suggests that each team member provides an estimate and a consensus estimate is reached after a few rounds of discussion and (re-)estimation. This practice is different from traditional approaches (e.g. function points) in several aspects. Both story points and function points are a measure of size. However,

[a]`https://www.atlassian.com/software/jira`
[b]`https://jira.spring.io/browse/XD-2970`

function points can be determined by an external estimator based on a standard set of rules (e.g. counting inputs, outputs, and inquiries) that can be applied consistently by any trained practitioner. On the other hand, story points are developed by a specific team based on the team's cumulative knowledge and biases, and thus may not be useful outside the team (e.g. in comparing performance across teams). Since story points represent the effort required for completing a user story, an estimate should cover different factors which can affect the effort. These factors include how much work needed to be done, the complexity of the work, and any uncertainty involving in the work [125]. Hence, it is important that the team is *consistent* in their story point estimates to avoid reducing the predictability in planning and managing their project. A machine learner can help the team maintain this consistency, especially in coping with increasingly large numbers of issues. It does so by learning insight from past issues and estimations to make future estimations.

6.2 Deep-SE

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

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

Figure 6.2 shows the Deep learning model for Story point Estimation (Deep-SE) that we have designed for the story point prediction system. It is composed of four components arranged sequentially: (i) word embedding, (ii) document representation using Long Short-Term Memory (LSTM) [66], (iii) deep representation using Recurrent Highway Net (RHWN) [256]; and (iv) differentiable regression. Given a document which consists of a sequence of words $s = (w_1, w_2, ..., w_n)$, e.g. the word sequence *(Standardize, XD, logging, to, align, with, ...)* in the title and description of issue `XD-2970` in Figure 6.1.

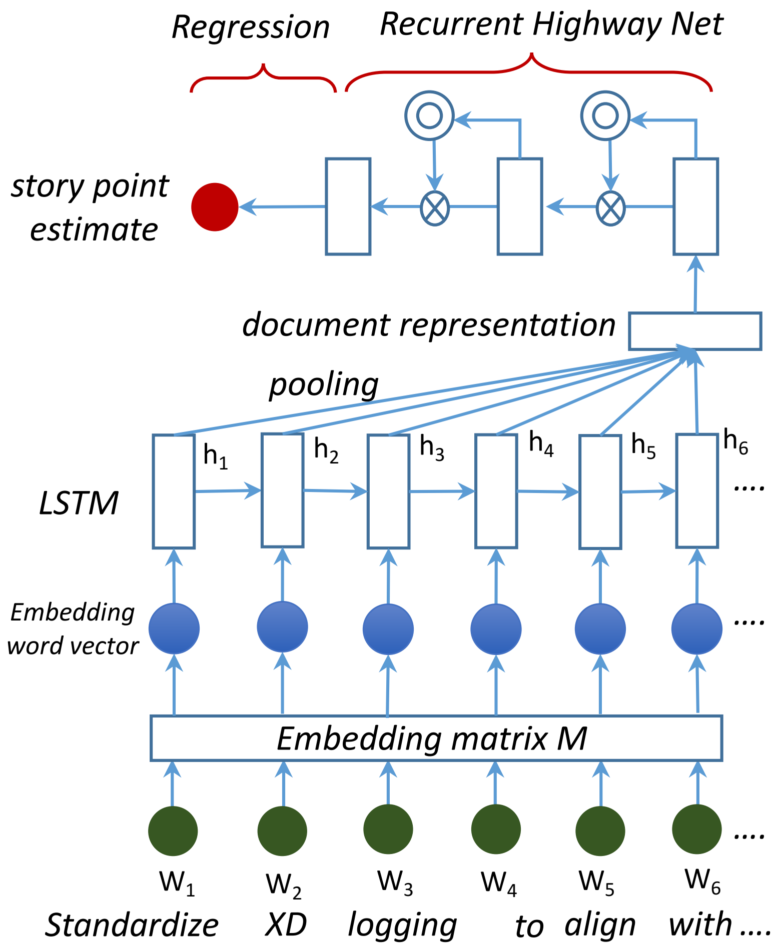


Figure 6.2: Deep learning model for Story point Estimation (Deep-SE). The input layer (bottom) is a sequence of words (represented as filled circles). Words are first embedded into a continuous space, then fed into the LSTM layer. The LSTM outputs a sequence of state vectors, which are then pooled to form a document-level vector. This global vector is then fed into a Recurrent Highway Net for multiple transformations (See Eq. (6.1) for detail). Finally, a regressor predicts an outcome (story-point).

We model a document's semantics based on the principle of compositionality: the meaning of a document is determined by the meanings of its constituents (e.g. words) and the rules used to combine them (e.g. one word followed by another). Hence, our approach models document representation in two stages. It first converts each word in a document into a fixed-length vector (i.e. word embedding). These word vectors then serve as an input sequence to the Long Short-Term Memory (LSTM) layer which computes a vector representation for the whole document.

The mechanism of LSTM allows the model to learn long-term dependencies in text effectively. Consider trying to predict the last word in the following text extracted from the description of issue `XD-2970` in Figure 6.1: *"Boot uses slf4j APIs backed by* *logback. This causes some build incompatibilities .... An additional step is to replace log4j with __.".* Recent information suggests that the next word is probably the name of a logging library, but if we want to narrow down to a specific library, we need to remember that "logback" and "log4j" are logging libraries from the earlier text. There could be a big gap between relevant information and the point where it is needed, but LSTM is capable to learn to connect the information.

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

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

6.2.1 Word embedding

We represent each word as a low dimensional, continuous and real-valued vector, also known as *word embedding*. Here we maintain a look-up table, which is a word embedding matrix $\mathscr{M} \in \mathbb{R}^{d\times|V|}$ where d is the dimension of word vector and $|V|$ is vocabulary size. These word vectors are pre-trained from corpora of issue reports, which will be described in details in Section 6.3.1.

6.2.2 Document representation using LSTM

Since an issue document consists of a sequence of words, we model the document by accumulating information from the start to the end of the sequence. A powerful accumulator is a Recurrent Neural Network (RNN) [64], which can be seen as multiple copies of the same single-hidden-layer network, each passing information to a successor. Thus, recurrent networks allow information to be accumulated. While RNNs are theoretically powerful, they are difficult to train for long sequences [64], which are often seen in issue reports (e.g. see the description of issue `XD-2970` in Figure 6.1). Hence, our approach employs Long Short-Term Memory (LSTM), a special variant of RNN.

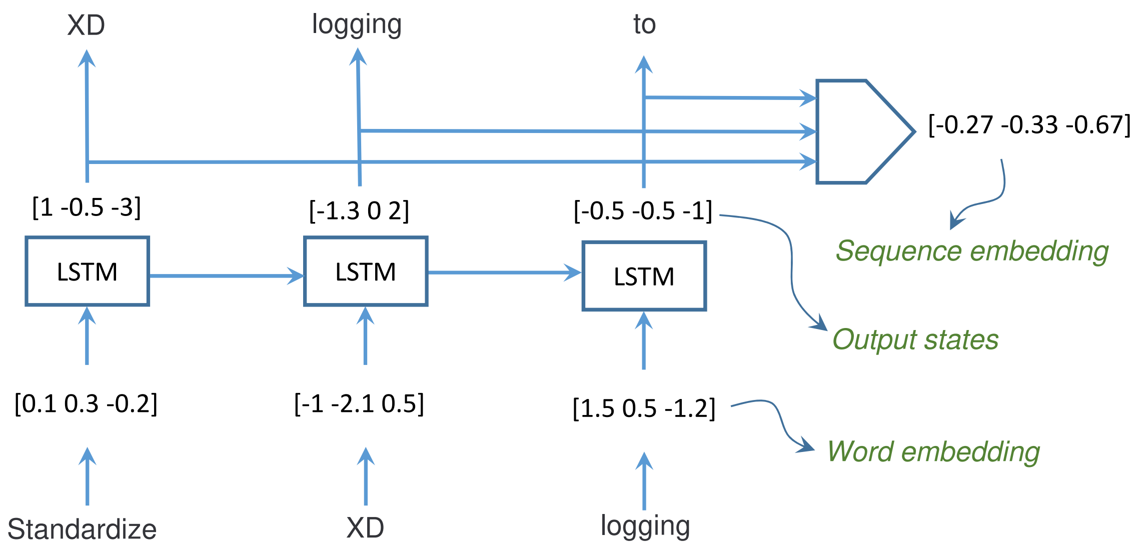


Figure 6.3: An example of how a vector representation is obtained for issue reports

After the vector output state has been computed for every word in the input sequence, the next step is aggregating those vectors into a single vector representing the whole document (see Figure 6.3). The aggregation operation is known as pooling. There are multiple ways to perform pooling, but the main requirement is that pooling must be length invariant. In other words, pooling is not sensitive to variable length of the document. For example, the simplest statistical pooling method is mean-pooling where we take the sum of the state vectors and divide it by the number of vectors. Other pooling methods are such as max pooling (e.g. choose the maximum value in each dimension), min pooling and sum pooling. From our experience in other settings, a simple but often effective pooling method is averaging, which we also employed here [257].

6.2.3 Deep representation using Recurrent Highway Network

Given that vector representation of an issue report has been extracted by the LSTM layer, we can use a differentiable regressor for immediate prediction. However, this may be sub-optimal since the network is rather shallow. Deep neural networks have become a popular method with many ground-breaking successes in vision [258], speech recognition [259] and NLP [260], [261]. Deep nets represent complex data more efficiently than shallow ones [262]. Deep models can be expressive while staying compact, as theoretically analysed by recent work [263]–[267]. This have been empirically validated in recent record-breaking results in vision, speech recognition and machine translation. However, learning standard feedforward networks with many hidden layers is notoriously difficult due to two main problems: (i) the number of parameters grows with the number of layers, leading to overfitting; and (ii) stacking many non-linear functions makes it difficult for the information and the gradients to pass through.

To address these problems, we designed a deep representation that performs multiple non-linear transformations using the idea from Highway Networks. Highway Nets are the latest idea that enables efficient learning through those many non-linear layers [268]. A Highway Net is a special type of feedforward neural networks with a modification to the transformation taking place at a hidden unit to let information from lower layers pass *linearly* through. Specifically, the hidden state at layer l is defined as:

$$h_{l+1} = \alpha_l * h_l + (1 - \alpha_l) * \sigma_l(h_l) \tag{6.1}$$

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

We need to learn a mapping from the raw words in an issue description to the story points. A deep feedforward neural network like Highway Net effectively breaks the mapping into a series of nested simple mappings, each described by a different layer of the network. The first layer provides a (rough) estimate, and subsequent layers iteratively refine that estimate. As the number of layers increase, further refinement can be achieved. Comparing to traditional feedforward networks, the special gating scheme in Highway Net is highly effective in letting the information and the gradients to pass through while stacking many non-linear functions. In fact, earlier work has demonstrated that Highway Net can have up to a thousand layers [268], while traditional deep neural nets cannot go beyond several layers [269].

We have also modified the standard Highway Network by *sharing* parameters between layers, i.e. all the hidden layers having the same hidden units. This is similar to the notion of a recurrent network, and thus we called it a Recurrent Highway Network. Previous work [256] has demonstrated the effectiveness of this approach in pattern recognition. This key novelty allows us to create a very compact version of Recurrent Highway Network with only one set of parameters in α_l and σ_l. This clearly produces a great advantage of avoiding overfitting. We note that the number of layers here refers to the number of hidden layers of a Recurrent Highway Network, not the number of LSTM layers. The number of LSTM layers is the same as the number of words in an issue's description.

6.2.4 Regression

At the top-layer of Deep-SE, we employ linear activation function in a feedforward neural network as the final regressor (see Figure 6.2) to produce a story-point estimate. This function can be defined as follows.

$$y = b_0 + \sum_{i=1}^{n} b_i x_i \tag{6.2}$$

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

6.3 Model training

6.3.1 Pre-training

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

$$P\left(w_{t+1} = k \mid w_{1:t}\right) = \frac{\exp\left(U_k h_t\right)}{\sum_{k'} \exp\left(U_{k'} h_t\right)} \tag{6.3}$$

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

We note that the probability of the first word $P(w_1)$ in a sequence is the number of sequences in the corpus which has that word w_1 starting first. At step t, h_t is computed by feeding h_{t-1} and w_t to the LSTM unit (see Figure 2.6). Since w_t is a word embedding vector, Eq. (6.3) indirectly refers to the embedding matrix .

The language model can be learned by optimizing the log-loss $-\log P(s)$. However, the main bottleneck is computational: Equation (6.3) costs $|V|$ time to evaluate where $|V|$ is the vocabulary size, which can be hundreds of thousands for a big corpus. For that reason, we implemented an approximate but very fast alternative based on Noise-Contrastive Estimation [270], which reduces the time to $M \ll |V|$, where M can be as small as 100. We also run the pre-training multiple times against a validation set to choose the best model. We use *perplexity*, a common intrinsic evaluation metric based on the log-loss, as a criterion for choosing the best model and early stopping. A smaller perplexity implies a better language model. The word embedding matrix $M \in \mathbb{R}^{d\times|V|}$ (which is first randomly initialized) and the initialization for LSTM parameters are learned through this pre-training process.

6.3.2 Training Deep-SE

We have implemented the Deep-SE model in Python using Theano [271]. To simplify our model, we set the size of the memory cell in an LSTM unit and the size of a recurrent layer in RHWN to be the same as the embedding size. We tuned some important hyper-parameters (e.g. embedding size and the number of hidden layers) by conducting experiments with different values, while for some other hyper-parameters, we used the default values. This will be discussed in more details in the evaluation section.

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

Since every component in the model is differentiable, we use the popular stochastic gradient descent to perform optimization: through backpropagation, the model parameters θ are updated in the opposite direction of the gradient of the loss function $L(\theta)$. In this search, a learning rate η is used to control how large of a step we take to reach a (local) minimum. We use RMSprop, an adaptive stochastic gradient method (unpublished note by Geoffrey Hinton), which is known to work best for recurrent models. We tuned RMSprop by partitioning the data into mutually exclusive training, validation, and test sets and running the training multiple times. Specifically, the training set is used to learn a useful model. After each training epoch, the learned model was evaluated on the validation set and its performance was used to assess against hyperparameters (e.g. learning rate in gradient searches). Note that the validation set was *not* used to learn any of the model's parameters. The best performing model in the validation set was chosen to be evaluated on the test set. We also employed the early stopping strategy (see Section 6.5.3), i.e. monitoring the model's performance during the validation phase and stopping when the performance got worse. If the log-loss does not improve for ten consecutive runs, we than terminate the training.

To prevent overfitting in our neural network, we have implemented an effective solution called *dropout* in our model [58], where the elements of input and output states are randomly set to zeros during training. During testing, parameter averaging is used. In effect, dropout implicitly trains many models in parallel, and all of them share the same parameter set. The final model parameters represent the average of the parameters across these models. Typically, the dropout rate is set at 0.5.

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

6.4 Dataset

In this section, we describe how data were collected for our study and experiments.

6.4.1 Data collecting

To collect data for our dataset, we looked for issues that were estimated with story points. JIRA is one of the few widely-used issue tracking systems that support agile development (and thus story point estimation) with its JIRA Agile plugin. Hence, we selected a diverse collection of nine major open source repositories that use the JIRA issue tracking system: Apache, Appcelerator, DuraSpace, Atlassian, Moodle, Lsstcorp, MuleSoft, Spring, and Talendforge. We then used the Representational State Transfer (REST) API provided by JIRA to query and collected those issue reports. We collected all the issues which were assigned a story point measure from the nine open source repositories up until August

Table 6.1: Descriptive statistics of our story point dataset

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

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

8, 2016. We then extracted the story point, title and description from the collected issue reports.

6.4.2 Data preprocessing

Each repository contains a number of projects, and we chose to include in our dataset only projects that had more than 300 issues with story points. Issues that were assigned a story point of zero (e.g., a non-reproducible bug), as well as issues with a negative, or unrealistically large story point (e.g. greater than 100) were filtered out. Ultimately, about 2.66% of the collected issues were filtered out in this fashion. In total, our dataset has 23,313 issues with story points from 16 different projects: Apache Mesos (ME), Apache Usergrid (UG), Appcelerator Studio (AS), Aptana Studio (AP), Titanum SDK/CLI (TI), DuraCloud (DC), Bamboo (BB), Clover (CV), JIRA Software (JI), Moodle (MD), Data Management (DM), Mule (MU), Mule Studio (MS), Spring XD (XD), Talend Data Quality (TD), and Talend ESB (TE). Table 6.1 summarizes the descriptive statistics of all the projects in terms of the minimum, maximum, mean, median, mode, variance, and standard deviations of story points assigned used and the average length of the title and description of issues in each project. These sixteen projects bring diversity to our dataset in terms of both application domains and project's characteristics. Specifically, they are different in the following aspects: number of observation (from 352 to 4,667 issues), technical characteristics (different programming languages and different application domains), sizes (from 88 KLOC to 18 millions LOC), and team characteristics (different team structures and participants from different regions).

Since story points rate the relative effort of work between user stories, they are usually measured on a certain scale (e.g. 1, 2, 4, 8, etc.) to facilitate comparison (e.g. a user story is double the effort of the other) [30]. The story points used in planning poker typically follow a Fibonacci scale, i.e. 1, 2, 3, 5, 8, 13, 21, and so on [125]. Among the projects we studied, only seven of them (i.e. Usergrid, Talend ESB, Talend Data Quality, Mule Studio, Mule, Appcelerator Studio, and Aptana Studio followed the Fibonacci scale, while the other nine projects did not use any scale. When our prediction system give an estimate, we did not round it to the nearest story point value on the Fibonacci scale. An alternative approach (for those project which follow a Fibonacci scale) is treating this as a classification problem: each value on the Fibonacci scale represents a class. The limitations of this approach is that the number of classes must be pre-determined and that it is not applicable to projects that do not follow this scale.

6.5 Evaluation

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

- **RQ1. Sanity Check**: *Is the proposed approach suitable for estimating story points?* This sanity check requires us to compare our Deep-SE prediction model with the three common baseline benchmarks used in the context of effort estimation: Random Guessing, Mean Effort, and Median Effort. Random guessing is a naive benchmark used to assess if an estimation model is useful [168]. Random guessing performs random sampling (with equal probability) over the set of issues with known story points, chooses randomly one issue from the sample, and uses the story point value of that issue as the estimate of the target issue. Random guessing does not use any information associated with the target issue. Thus any useful estimation model should outperform random guessing. Mean and Median Effort estimations are commonly used as baseline benchmarks for effort estimation [7]. They use the mean or median story points of the past issues to estimate the story points of the target issue. Note that the samples used for all the naive baselines (i.e. Random Guessing, Mean Effort, and Median Effort) were from the training set.

- **RQ2. Benefits of deep representation**: *Does the use of Recurrent Highway Nets provide more accurate story point estimates than using a traditional regression technique?*
To answer this question, we replaced the Recurrent Highway Net component with a regressor for immediate prediction. Here, we compare our approach against four common regressors: Random Forests (RF), Support Vector Machine (SVM), Automatically Transformed Linear Model (ATLM), and Linear Regression (LR). We choose RF over other baselines since ensemble methods like RF, which combine the estimates from multiple estimators, are an effective method for effort estimation [8]. RF achieves a significant improvement over the decision tree approach by generating many classification and regression trees, each of which is built on a random resampling of the data, with a random subset of variables at each node split. Tree predictions are then aggregated through averaging. We used the issues in the validation set to fine-tune parameters (i.e. the number of tress, the maximum depth of the tree, and The minimum number of samples). For SVM, it has been widely use in software analytics (e.g. defect prediction) and document classification (e.g. sentiment analysis) [272]. SVM is known as Support Vector Regression (SVR) for regression problems. We also used the issues in the validation set to find

the kernel type (e.g. linear, polynomial) for testing. We used the Automatically Transformed Linear Model (ATLM) [273] recently proposed as the baseline model for software effort estimation. Although ATLM is simple and requires no parameter tuning, it performs well over a range of various project types in the traditional effort estimation [273]. Since LR is the top layer of our approach, we also used LR as the immediate regressor after LSTM layers to assess whether RHWN improves the predictive performance. We then compare the performance of these alternatives, namely LSTM+RF, LSTM+SVM, LSTM+ATLM, and LSTM+LR against our Deep-SE model.

- **RQ3. Benefits of LSTM document representation**: *Does the use of LSTM for modeling issue reports provide more accurate results than the traditional Doc2Vec and Bag-of-Words (BoW) approach?*
  The most popular text representation is Bag-of-Words (BoW) [136], where a text is represented as a vector of word counts. For example, the title and description of issue `XD-2970` in Figure 6.1 would be converted into a sparse binary vector of vocabulary size, whose elements are mostly zeros, except for those at the positions designated to "standardize", "XD", "logging" and so on. However, BoW has two major weaknesses: they lose the sequence of the words and they also ignore semantics of the words. For example, "Python", "Java", and "logging " are equally distant, while semantically "Python" should be closer to "Java" than "logging". To address this issue, Doc2vec [274] (i.e. alternatively known as paragraph2vec) is an unsupervised algorithm that learns fixed-length feature representations from texts (e.g. title and description of issues). Each document is represented in a dense vector which is trained to predict next words in the document.

  Both BoW and Doc2vec representations however effectively destroys the sequential nature of text. This question aims to explore whether LSTM with its capability of modeling this sequential structure would improve the story point estimation. To answer this question, we feed three different feature vectors: one learned by LSTM and the other two derived from BoW technique and Doc2vec to the same Random Forrests regressor, and compare the predictive performance of the former (i.e. LSTM+RF) against that of the latter (i.e. BoW+RF and Doc2vec+RF).We used Gensim[c], a well-known implementation for Doc2vec in our experiments.

- **RQ4. Cross-project estimation**: *Is the proposed approach suitable for cross-project estimation?*

[c]`https://radimrehurek.com/gensim/models/doc2vec.html`

Story point estimation in new projects is often difficult due to lack of training data. One common technique to address this issue is training a model using data from a (source) project and applying it to the new (target) project. Since our approach requires only the title and description of issues in the source and target projects, it is readily applicable to both within-project estimation and cross-project estimation. In practice, story point estimation is however known to be specific to teams and projects. Hence, this question aims to investigate whether our approach is suitable for cross-project estimation. We have implemented Analogy-based estimation called ABE0, which were proposed in previous work [275]–[278] for cross-project estimation, and used it as a benchmark. The ABE0 estimation bases on the distances between individual issues. Specifically, the story point of issues in the target project is the mean of story points of k-nearest issues from the source project. We used the Euclidean distance as a distance measure, Bag-of-Words of the title and the description as the features of an issue, and $k = 3$.

- **RQ5. Normalizing/adjusting story points**: *Does our approach still perform well with normalized/adjusted story points?*

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

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

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

- **RQ6. Compare against the existing approach:** *How does our approach perform against existing approaches in story point estimation?*

  Recently, Porru et. al. [279] also proposed an estimation model for story points. Their approach uses the type of an issue, the component(s) assigned to it, and the TF-IDF derived from its summary and description as features representing the issue. They also performed univariate feature selection to choose a subset of features for building a classifier. By contrast, our approach automatically learns semantic features which represent the actual meaning of the issue's report, thus potentially providing more accurate estimates. To answer this research question, we ran Deep-SE on the dataset used in Porru et. al, re-implemented their approach, and performed a comparison on the results produced by the two approaches.

6.5.1 Experimental setting

We performed experiments on the sixteen projects in our dataset – see Table 6.1 for their details. To mimic a real deployment scenario that prediction on a current issue is made by using knowledge from estimations of the past issues, the issues in each project were split into training set (60% of the issues), development/validation set (i.e. 20%), and test set (i.e. 20%) based on their creation time. The issues in the training set and the validation set were created before the issues in the test set, and the issues in the training set were also created before the issues in the validation set.

6.5.2 Performance measures

There are a range of measures used in evaluating the accuracy of an effort estimation model. Most of them are based on the Absolute Error, (i.e. $|ActualSP - EstimatedSP|$). where $AcutalSP$ is the real story points assigned to an issue and $EstimatedSP$ is the outcome given by an estimation model. Mean of Magnitude of Relative Error (MRE) or Mean Percentage Error and Prediction at level l [150], i.e. Pred(l), have also been used in effort estimation. However, a number of studies [151]–[154] have found that those measures bias towards underestimation and are not stable when comparing effort estimation models. Thus, the *Mean Absolute Error (MAE)*, *Median Absolute Error (MdAE)*, and the *Standardized Accuracy (SA)* have recently been recommended to compare the performance of effort estimation models [7], [280]. MAE is defined as:

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

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

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

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

where $1 \leq i \leq N$.

SA is based on MAE and it is defined as:

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

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

We assess the story point estimates produced by the estimation models using MAE, MdAE and SA. To compare the performance of two estimation models, we tested the statistical significance of the absolute errors achieved with the two models using the Wilcoxon Signed Rank Test [156]. The Wilcoxon test is a safe test since it makes no assumptions about underlying data distributions. The null hypothesis here is: "the absolute errors provided by an estimation model are not different to those provided by another estimation model". We set the confidence limit at 0.05 and also applied Bonferroni correction [281] (0.05/K, where K is the number of statistical tests) when multiple testing were performed.

In addition, we also employed a non-parametric effect size measure, the correlated samples case of the Vargha and Delaney's $\hat{A}_{XY}$ statistic [282] to assess whether the effect size is interesting. The $\hat{A}_{XY}$ measure is chosen since it is agnostic to the underlying distribution of the data, and is suitable for assessing randomized algorithms in software engineering generally [160] and effort estimation in particular [7]. Specifically, given a performance measure (e.g. the Absolute Error from each estimation in our case), the $\hat{A}_{XY}$

measures the probability that estimation model X achieves better results (with respect to the performance measure) than estimation model Y. We note that this falls into the correlated samples case of the Vargha and Delaney [282] where the Absolute Error is derived by applying different estimation methods on the same data (i.e. same issues). We thus use the following formula to calculate the stochastic superiority value between two estimation methods:

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

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

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

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

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

We focused on tuning two important hyper-parameters: the number of word embedding dimensions and the number of hidden layers in the recurrent highway net component of our model. To do so, we fixed one parameter and varied the other to observe the MAE performance. We chose to test with four different embedding sizes: 10, 50, 100, and 200, and twelve variations of the number of hidden layers from 2 to 200. The embedding size is the number of dimensions of the vector which represents a word. This word embedding is a low dimensional vector representation of words in the vocabulary. This tuning was done using the validation set. Figure 6.4 shows the results from experimenting with Apache Mesos. As can be seen, the setting where the number of embeddings is 50 and the number of hidden layers is 10 gives the lowest MAE, and thus was chosen.

For both pre-training we trained with 100 runs and the batch size is 50. The initial learning rate in pre-training was set to 0.02, adaptation rate was 0.99, and smoothing factor was 10^{-7}. For the main Deep-SE model we used 1,000 epoches and the batch size

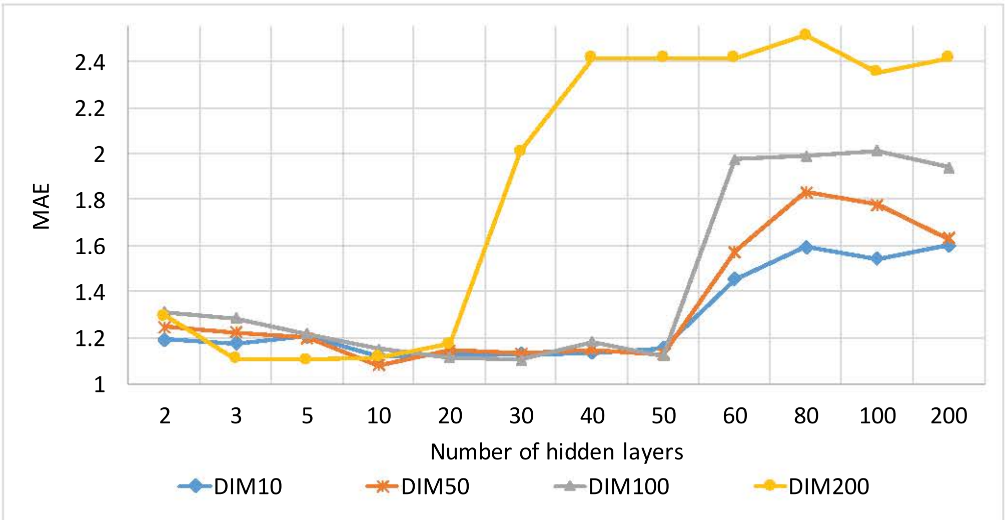


Figure 6.4: Story point estimation performance with different parameter.

wass set to 100. The initial learning rate in the main model was set to 0.01, adaptation rate was 0.9, and smoothing factor was 10^{-6}. Dropout rates for the RHWN and LSTM layers were set to 0.5 and 0.2 respectively. The maximum sequence length used by the LSTM is 100 words, which is the average length of issue description.

6.5.4 Pre-training

In most repositories, we used around 50,000 issues without story points (i.e. without labels) for pre-training, except the Mulesoft repository which has much smaller number of issues (only 8,036 issues) available for pre-training. Figure 6.5 show the top-500 frequent words used in Apache. They are divided into 9 clusters (using K-means clustering) based on their embedding which was learned through the pre-training process. We used t-distributed stochastic neighbor embedding (t-SNE) [283] to display high-dimensional vectors in two dimensions.

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

The pre-training step is known to effectively deal with limited labelled data [284]–[286]. Here, pre-training does not require story-point labels since it is trained by predicting the next words. Hence the number of data points equals to the number of words. Since

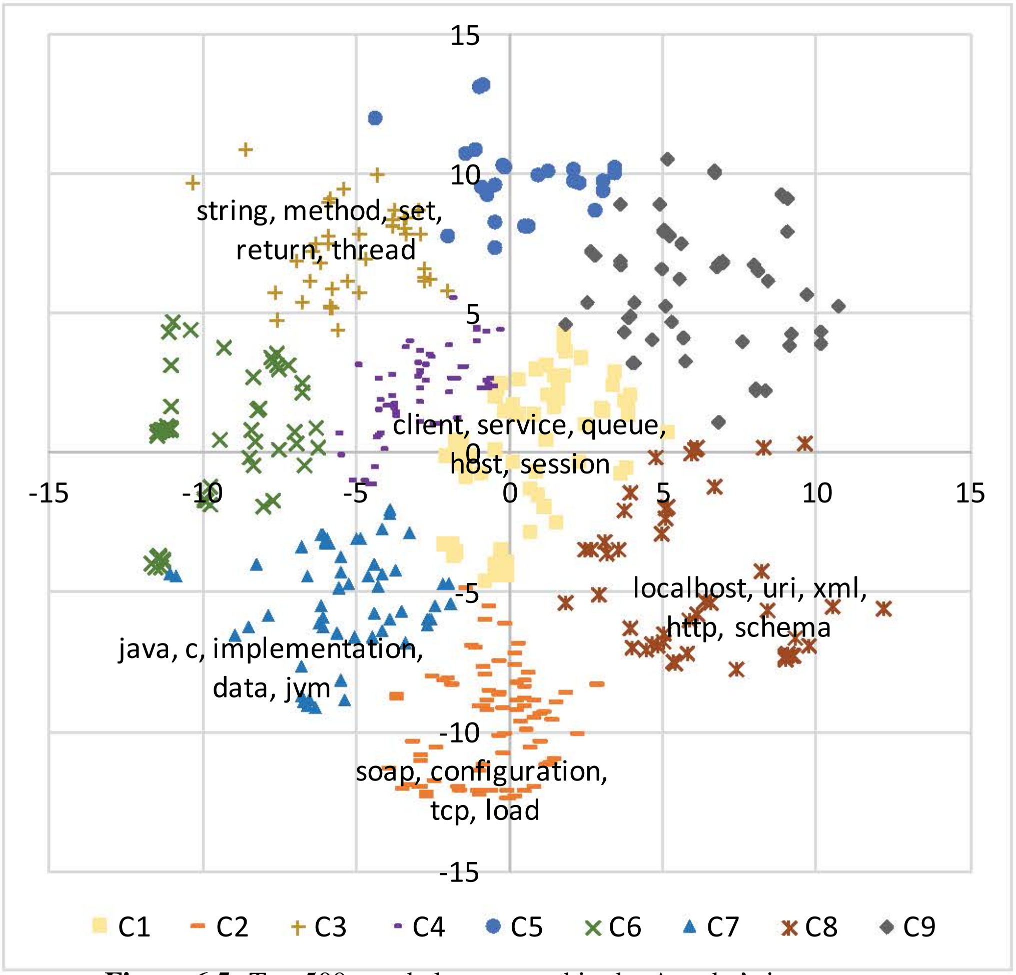


Figure 6.5: Top-500 word clusters used in the Apache's issue reports

for each project repository we used 50,000 issues for pre-training, we had approximately 5 million data points per repository for pre-training.

6.5.5 The correlation between the story points and the development time

Identifying the actual effort required for completing an issue is very challenging (especially in open source projects) since in most cases the actual effort was not tracked and recorded. We were however able to extract the development time which was the duration between when the issue's status was set to "in-progress" and when it was set to "resolved". Thus, we have explicitly excluded the waiting time for being assigned to a developer or being put on hold. The development time is the closest to the actual effort of completing the issue that we were able to extract from the data. We then performed two widely-used statistical tests (Spearman's rank and Pearson rank correlation) [287] for all the issues in

Table 6.2: The coefficient and p-value of the Spearman's rank and Pearson rank correlation on the story points against the development time

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

our dataset. Table 6.2 shows the Spearman's rank and Pearson rank correlation coefficient and p-value for all projects. We have found that there is a significantly ($p < 0.05$) positive correlation between the story points and the development time across all 16 project we studied. In some projects (e.g. Moodle) there was a strong correlation with the coefficients was around 0.8. This positive correlation demonstrates that the higher story point, the longer development time, which suggests that a correlation between an issue's story points and its actual effort.

6.5.6 Results

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

RQ1: Sanity check

Table 6.3 shows the results achieved from Deep-SE, and two baseline methods: Mean and Median method (See Appendix A.1 for the distribution of the Absolute Error). The analysis of MAE, MdAE, and SA suggests that the estimations obtained with our approach, Deep-SE, are better than those achieved by using Mean, Median, and Ran-

Table 6.3: Evaluation results of Deep-SE, the Mean and Median method (the best results are highlighted in bold). MAE and MdAE - the lower the better, SA - the higher the better.

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

dom estimates. Deep-SE consistently outperforms all these three baselines in all sixteen projects.

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

We note that the results achieved by the estimation models vary between different projects. For example, our Deep-SE achieved 0.64 MAE in the Talend ESB project (TE), while it achieved 5.97 MAE in Moodle (MD) project. The distribution of story points may be the cause of this variation: the standard deviation of story points in TE is only 1.50, while that in MD is 21.65 (see Table 6.1).

Table 6.4 shows the results of the Wilcoxon test (together with the corresponding $\hat{A}_{XY}$ effect size) to measure the statistical significance and effect size (in brackets) of the improved accuracy achieved by Deep-SE over the baselines: Mean Effort, Median Effort, and Random Guessing. In 45/48 cases, our Deep-SE significantly outperforms the baselines after applying Bonferroni correction with effect sizes greater than 0.5. Moreover, the average of the stochastic superiority (A_{iu}) of our approach against the baselines is greater than 0.7 in the most cases. The highest A_{iu} achieving in the Talend Data Quality project (TD) is 0.86 which can be considered as large effect size ($\hat{A}_{XY} > 0.8$).

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

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

RQ2: Benefits of deep representation

Table 6.5 shows MAE, MdAE, and SA achieved from Deep-SE using Recurrent Highway Networks (RHWN) for deep representation of issue reports against using Random Forests, Support Vector Machine, Automatically Transformed Linear Model, and

Table 6.4: Comparison on the effort estimation benchmarks using Wilcoxon test and $\hat{A}_{XY}$ effect size (in brackets)

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

Linear Regression Model coupled with LSTM (i.e. LSTM+RF, LSTM+SVM, LSTM+ATLM, and LSTM+LR). The distribution of the Absolute Error is reported in Appendix A.2. When we use MAE, MdAE, and SA as evaluation criteria, Deep-SE is still the best approach, consistently outperforming LSTM+RF, LSTM+SVM, LSTM+ATLM, and LSTM+LR across all sixteen projects.

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

In addition, the results for the Wilcoxon test to compare our approach (Deep-SE) against LSTM+RF, LSTM+SVM, LSTM+ATLM, and LSTM+LR is shown in Table 6.6. The improvement of our approach over LSTM+RF, LSTM+SVM, and LSTM+ATLM is still significant after applying p-value correction with the effect size greater than 0.5 in 59/64 cases. In most cases, when comparing the proposed model against LSTM+RF,

Table 6.5: Evaluation results of Deep-SE, LSTM+RF, LSTM+SVM, LSTM+ATLM, and LSTM+LR (the best results are highlighted in bold). MAE and MdAE - the lower the better, SA - the higher the better.

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

Table 6.6: Comparison between the Recurrent Highway Net against Random Forests, Support Vector Machine, Automatically Transformed Linear Model, and Linear Regression using Wilcoxon test and $\hat{A}_{12}$ effect size (in brackets)

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

LSTM+SVM, LSTM+ATLM, and LSTM+LR, the effect sizes are small (between 0.5 and 0.6). A major part of those improvement were brought by our use of the deep learning LSTM architecture to model the textual description of an issue. The use of highway recurrent networks (on top of LSTM) has also improved the predictive performance, but not as large effects as the LSTM itself (especially for those projects which have very small number of issues). However, our approach, Deep-SE, achieved A_{iu} greater than 0.6 in the most cases.

Answer to RQ2: The proposed approach of using Recurrent Highway Networks is effective in building a deep representation of issue reports and consequently improving story point estimation.

RQ3: Benefits of LSTM document representation

To study the benefits of using LSTM in representing issue reports, we compared the improved accuracy achieved by Random Forest using the features derived from LSTM against that using the features derived from BoW and Doc2vec. For a fair comparison we used Random Forests as the regressor in all settings and the result is reported in Table 6.7 (see the distribution of the Absolute Error in Appendix A.3). LSTM performs better than BoW and Doc2vec with respect to the MAE, MdAE, and SA measures in twelve projects

Table 6.7: Evaluation results of LSTM+RF, BoW+RF, and Doc2vec+RF (the best results are highlighted in bold). MAE and MdAE - the lower the better, SA - the higher the better.

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

Table 6.8: Comparison of Random Forest with LSTM, Random Forests with BoW, and Random Forests with Doc2vec using Wilcoxon test and $\hat{A}_{XY}$ effect size (in brackets)

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

(e.g. ME, UG, and AS) from sixteen projects. LSTM improved 4.16% and 11.05% in MAE over Doc2vec and BoW, respectively, averaging across all projects.

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

We acknowledge that BoW and Doc2vec perform better than LSTM in some cases. For example, in the Moodle project (MD), D2V+RF performed better than LSTM+RF in MAE and SA – it achieved 8.02 MAE and 33.19 SA. This could reflect that the combination between LSTM and RHWN significantly improves the accuracy of the estimations.

The improvement of LSTM over BoW and Doc2vec is significant after applying Bonferroni correction with effect size greater than 0.5 in 24/32 cases and A_{iu} being greater than 0.5 in all projects (see Table 6.8).

Answer to RQ3: The proposed LSTM-based approach is effective in automatically learning semantic features representing issue description, which improves story-point estimation.

Table 6.9: Mean Absolute Error (MAE) on cross-project estimation and comparison of Deep-SE and ABE0 using Wilcoxon test and $\hat{A}_{XY}$ effect size (in brackets)

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

RQ4: Cross-project estimation

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

Table 6.9 shows the performance of our Deep-SE model and ABE0 for cross-project estimation (see the distribution of the Absolute Error in Appendix A.4). We also used a benchmark of within-project estimation where older issues of the target project were used for training (see Table 6.3). In all cases, the proposed approach when used for cross-project estimation performed worse than when used for within-project estimation (e.g. on average 20.75% reduction in performance for within-repository and 97.92% for cross-repository). However, our approach outperformed the cross-project baseline (i.e. ABE0) in all cases – it achieved 2.33 and 3.82 MAE in within and cross repository, while ABE0 achieved 2.60 and 4.55 MAE. The improvement of our approach over ABE0 is still significant after applying p-value correction with the effect size greater than 0.5 in 14/16 cases.

These results confirm a universal understanding [30] in agile development that story point estimation is specific to teams and projects. Since story points are relatively measured, it is not uncommon that two different same-sized teams could give different estimates for the same user story. For example, team A may estimate 5 story points for user story UC_1 while team B gives 10 story points. However, it does not necessarily mean that team B would do more work for completing UC_1 than team A. It more likely means that team'B baselines are twice bigger than team A's, i.e. for "baseline" user story which requires 5 times less the effort than UC_1 takes, team A would give it 1 story point while team B gives 2 story points. Hence, historical estimates are more valuable for within-project estimation, which is demonstrated by this result.

Answer to RQ4: Given the specificity of story points to teams and projects, our proposed approach is more effective for within-project estimation.

RQ5: Adjusted/normalized story points

Table 6.10 shows the results of our Deep-SE and the other baseline methods in predicting the normalized story points. Deep-SE performs well across all projects. Deep-SE improved MAE between 2.13% to 93.40% over the Mean method, 9.45% to 93.27% over the Median method, 7.02% to 53.33% over LSTM+LR, 1.20% to 61.96% over LSTM+ATLM, 1.20% to 53.33% over LSTM+SVM, 4.00% to 30.00% over Doc2vec+RF, 2.04% to 36.36% over BoW+RF, and 0.86% to 25.80% over LSTM+RF. The best result is obtained in the Usergrid project (UG), it is 0.07 MAE, 0.01 MdAE, and 93.50 SA. We however note that the adjusted story points benefits all methods since it narrows the gap between minimum and maximum value and the distribution of the story points.

Answer to RQ5: Our proposed approach still outperformed other techniques in estimating the new adjusted story points.

Table 6.10: Evaluation results on the adjusted story points (the best results are highlighted in bold). MAE and MdAE - the lower the better, SA - the higher the better.

Proj	Method	MAE	MdAE	SA	Proj	Method	MAE	MdAE	SA
ME	**Deep-SE**	**0.27**	**0.03**	**76.58**	JI	**Deep-SE**	**0.60**	**0.51**	**63.20**
	lstm+rf	0.34	0.15	70.43		lstm+rf	0.74	0.79	54.42
	bow+rf	0.36	0.16	68.82		bow+rf	0.66	0.53	58.99
	d2v+rf	0.35	0.15	69.87		d2v+rf	0.70	0.53	56.99
	lstm+svm	0.33	0.10	71.20		lstm+svm	0.94	0.89	41.97
	lstm+atlm	0.33	0.14	70.97		lstm+atlm	0.89	0.89	45.18
	lstm+lr	0.37	0.21	67.68		lstm+lr	0.89	0.89	45.18
	mean	1.12	1.07	3.06		mean	1.31	1.71	18.95
	median	1.05	1.00	8.87		median	1.60	2.00	1.29
UG	**Deep-SE**	**0.07**	0.01	**93.50**	MD	**Deep-SE**	**2.56**	**2.29**	**31.83**
	lstm+rf	0.08	**0.00**	92.59		lstm+rf	3.45	3.55	8.24
	bow+rf	0.11	0.01	90.31		bow+rf	3.32	3.27	11.54
	d2v+rf	0.10	0.01	91.22		d2v+rf	3.39	3.48	9.70
	lstm+svm	0.15	0.10	86.38		lstm+svm	3.12	3.07	16.94
	lstm+atlm	0.15	0.08	86.25		lstm+atlm	3.48	3.49	7.41
	lstm+lr	0.15	0.08	86.25		lstm+lr	3.57	3.28	4.98
	mean	1.04	0.98	4.79		mean	3.60	3.67	4.18
	median	1.06	1.00	2.64		median	2.95	3.00	21.48
AS	**Deep-SE**	**0.53**	**0.20**	**69.16**	DM	**Deep-SE**	**2.30**	**1.43**	**31.99**
	lstm+rf	0.56	0.45	67.49		lstm+rf	2.83	2.59	16.23
	bow+rf	0.56	0.49	67.39		bow+rf	2.83	2.63	16.33
	d2v+rf	0.56	0.46	67.37		d2v+rf	2.92	2.80	13.80
	lstm+svm	0.55	0.32	68.34		lstm+svm	2.45	1.78	27.56
	lstm+atlm	0.57	0.46	66.87		lstm+atlm	2.83	2.57	16.28
	lstm+lr	0.57	0.49	67.12		lstm+lr	2.83	2.57	16.28
	mean	1.18	0.79	31.89		mean	3.27	3.41	3.25
	median	1.35	1.00	21.54		median	2.61	2.00	22.94
AP	**Deep-SE**	**0.92**	**0.86**	**21.95**	MU	**Deep-SE**	**0.68**	0.59	**63.83**
	lstm+rf	0.99	0.87	16.23		lstm+rf	0.70	**0.55**	63.01
	bow+rf	1.00	0.87	15.33		bow+rf	0.70	0.57	62.79
	d2v+rf	0.99	0.86	15.94		d2v+rf	0.71	0.57	62.17
	lstm+svm	1.12	0.92	5.26		lstm+svm	0.70	0.62	62.62

Continued on next page

Table 6.10 – continued from previous page

Proj	Method	MAE	MdAE	SA	Proj	Method	MAE	MdAE	SA
	lstm+atlm	1.03	0.84	12.63		lstm+atlm	0.93	0.74	50.77
	lstm+lr	1.17	1.05	1.14		lstm+lr	0.79	0.61	58.00
	mean	1.15	0.64	2.49		mean	1.21	1.51	35.86
	median	0.94	1.00	20.29		median	1.64	2.00	12.80
TI	**Deep-SE**	**0.59**	**0.17**	**56.53**	MS	**Deep-SE**	**0.86**	**0.65**	**56.82**
	lstm+rf	0.72	0.56	46.22		lstm+rf	0.91	0.76	54.37
	bow+rf	0.73	0.58	46.10		bow+rf	0.89	0.93	55.48
	d2v+rf	0.72	0.56	46.17		d2v+rf	0.90	0.69	54.66
	lstm+svm	0.73	0.62	45.74		lstm+svm	0.94	0.78	52.91
	lstm+atlm	0.73	0.57	45.86		lstm+atlm	0.99	0.87	50.45
	lstm+lr	0.73	0.56	45.77		lstm+lr	0.99	0.87	50.45
	mean	1.32	1.56	1.57		mean	1.23	0.62	38.49
	median	0.86	1.00	36.04		median	1.44	1.00	27.83
DC	**Deep-SE**	**0.48**	**0.48**	**55.77**	XD	**Deep-SE**	**0.35**	**0.08**	**80.66**
	lstm+rf	0.49	0.49	55.02		lstm+rf	0.44	0.37	75.78
	bow+rf	0.49	**0.48**	54.76		bow+rf	0.45	0.38	75.33
	d2v+rf	0.50	0.50	53.59		d2v+rf	0.45	0.32	75.31
	lstm+svm	0.49	0.43	55.24		lstm+svm	0.38	0.20	79.16
	lstm+atlm	0.53	0.47	51.02		lstm+atlm	0.92	0.76	49.05
	lstm+lr	0.53	0.47	51.02		lstm+lr	0.45	0.40	75.33
	mean	1.07	1.49	1.29		mean	1.03	1.28	43.06
	median	0.58	1.00	46.76		median	0.75	1.00	58.74
BB	**Deep-SE**	**0.41**	**0.12**	**72.00**	TD	**Deep-SE**	**0.82**	**0.64**	**53.36**
	lstm+rf	0.43	0.38	70.37		lstm+rf	0.84	0.68	52.65
	bow+rf	0.45	0.40	69.33		bow+rf	0.88	0.65	50.30
	d2v+rf	0.49	0.45	66.34		d2v+rf	0.86	0.70	51.46
	lstm+svm	0.42	0.21	71.21		lstm+svm	0.83	0.62	53.24
	lstm+atlm	0.47	0.41	67.53		lstm+atlm	0.83	0.58	52.82
	lstm+lr	0.47	0.41	67.53		lstm+lr	0.90	0.74	48.88
	mean	1.15	0.76	20.92		mean	1.29	1.42	27.20
	median	1.39	1.00	4.50		median	0.99	1.00	44.17
CV	**Deep-SE**	**1.15**	**0.79**	**23.29**	TE	**Deep-SE**	**0.40**	**0.05**	**74.58**
	lstm+rf	1.16	1.05	22.55		lstm+rf	0.47	0.46	70.39

Continued on next page

Table 6.11: Mean Absolute Error (MAE) and comparison of Deep-SE and the Porru's approach using Wilcoxon test and $\hat{A}_{XY}$ effect size (in brackets)

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

Table 6.10 – continued from previous page

Proj	Method	MAE	MdAE	SA	Proj	Method	MAE	MdAE	SA
	bow+rf	1.22	1.10	18.95		bow+rf	0.48	0.48	69.52
	d2v+rf	1.20	1.09	20.30		d2v+rf	0.48	0.48	69.41
	lstm+svm	1.22	1.15	18.77		lstm+svm	0.45	0.41	71.77
	lstm+atlm	1.47	1.28	2.22		lstm+atlm	0.49	0.48	69.14
	lstm+lr	1.47	1.28	2.22		lstm+lr	0.49	0.48	69.14
	mean	1.27	1.11	15.18		mean	0.99	0.60	37.28
	median	1.29	1.00	13.92		median	1.39	1.00	12.09

RQ6: Compare Deep-SE against the existing approach

We applied our approach, Deep-SE, and the Porru et. al.'s approach on their dataset consisted of eight projects. Table 6.11 shows the evaluation results in MAE and the comparison of Deep-SE and the Porru et. al.'s approach. The distribution of the Absolute Error is reported in Appendix A.5. Deep-SE outperforms the existing approach in all cases. Deep-SE improved between 18.18% (in TIMOB) to 56.48% (in DNN) in terms of MAE. In addition, the improvement of our approach over the Porru et. al.'s approach is still significant after applying p-value correction with the effect size greater than 0.5 in all cases. Especially, the large effect size ($\hat{A}_{XY} > 0.7$) of the improvement is obtained in the DNN project.

Answer to RQ6: Our proposed approach outperformed the existing technique using TF-IDF in estimating the story points.

Table 6.12: The pre-training, training, and testing time at 50 embedding dimensions of our Deep-SE model

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

6.5.7 Training/testing time

Deep learning models are known for taking a long time for training. This is an important factor in considering adopting our approach, especially in an agile development setting. If training time takes longer than the duration of a sprint (e.g. one or two weeks), the prediction system would not be useful in practice. We have found that the training time of our model was very small, ranging from 13 minutes to 40 minutes with an average of 22 minutes across the 16 projects (see Table 6.12). Pre-training time took much longer time, but it was done only once across a repository and took just below 7 hours at the maximum. Once the model was trained, getting an estimation from the model was very fast. As can be seen from Table 6.12, the time it took for testing all issues in the test sets was in the order of seconds. Hence, for a given new issue, it would take less than a second for the machinery to come back with an story point estimation. All the experiments were run on a MacOS laptop with 2.4 GHz Intel Core i5 and 8 GB of RAM and the embedding dimensions of 50. Hence, this result suggests that using our proposed approach to estimate story points is applicable in practice.

6.5.8 Threats to validity

We tried to mitigate threats to construct validity by using real world data from issues recorded in large open source projects. We collected the title and description provided with these issue reports and the actual story points that were assigned to them. We are aware that those story points were estimated by human teams, and thus may contain biases and in some cases may not be accurate. We have mitigated this threats by performing two set of experiments: one on the orgianal story points and the other on the adjusted normalized story points. We further note that for story points, the raw values are not as important as the relative values [288]. A user story that is assigned 6 story points should be three times as much as a user story that is assigned 2 story points. Hence, when engineers determine an estimate for a new issue, they need to compare the issue to other issues in the past in order to make the estimation consistently. The problem is thus suitable for a machine learner. The trained prediction system works in a similar manner as human engineers: using past estimates as baselines for new estimation. The prediction system tries to reproduce an estimate that human engineers would arrive at.

However, since we aim to mimic the team's capability in effort estimation, the current set of ground-truths sufficiently serves this purpose. When other sets of ground-truths become available, our model can be easily retrained. To minimize threats to conclusion validity, we carefully selected unbiased error measures, applied a number of statistical tests, and applied multiple statistical testing correction to verify our assumptions [157]. Our study was performed on datasets of different sizes. In addition, we carefully followed recent best practices in evaluating effort estimation models [160], [168], [273] to decrease conclusion instability [289].

The original implementation of Porru et. al.'s method [279] was not released, thus we have re-implemented our own version of their approach. We strictly followed the described provided in their work, however we acknowledge that our implementation may not reflect all the implementation details in their approach. To mitigate this threat, we have tested our implementation using the dataset provided in their work. We have found that our results were consistent with the results reported in their work.

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

6.5.9 Implications

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

What do the results mean for the research on effort estimation? Existing work on effort estimation mainly focus on estimating the whole project with a small number of data points (see the datasets in the PROMISE repository [82] for example). The fast emergence of agile development demand more research on estimation at the issue or user story level. Our work opens a new research area for the use of software analytics in estimating story points. The assertion demonstrated by our results is that our current method works and no other methods has been demonstrated to work at this scale of above 23,000 data points. Existing work in software effort estimation have dealt with a much smaller number of observations (i.e. data points) than our work did. For example, the China dataset has only 499 data points, Desharnais has 77, and Finish has 38 (see the datasets for effort estimation on the PROMISE repository) – these are commonly used in existing effort estimation work (e.g. [7], [290]). By contrast, in this work we deal with the scale of thousands of data points. Since we make our dataset publicly available, further research (e.g. modeling the codebase and adding team-specific features into the estimation model) can be advanced in this topic, and our current results can serve as the baseline.

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

In our context of story point estimation, if the number of new words is large, transfer learning is needed, e.g. by using the existing model as a strong prior for the new model.

However, this can be mitigated by pre-training on a large corpus so that most of the terms are covered. After pre-training, our model is able to automatically learn semantic relations between words. For example, words related to networking like "soap", "configuration", "tcp", and "load" are in one cluster (see Figure 7). Hence, even when a user story has several unique terms (but already pre-trained), retraining the main model is not necessary. Pre-training may however take time and effort. One potential research direction is therefore building up a community for sharing pre-trained networks, which can be used for initialization, thus reducing training times. As the first step towards this direction, we make our pre-trained models publicly available for the research community.

We acknowledged that the explainability of a model is important for full adoption of machine learning techniques. This is not a unique problem only for recurrent networks (RNN), but also for many powerful modern machine learning techniques (e.g. Random Forests and SVM). However, RNN is not entirely a black-box as it seems (e.g. see [291]). For example, word importance can be credited using various techniques (e.g., using gradient with respect to word value). Alternatively, there are model agnostic technique to explain any prediction [292]. Even with partly interpretable RNN, if the prediction is accurate, then we can still expect a high level of adoption.

What do the results mean for project managers and developers?

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

6.6 Related work

Existing estimation methods can generally be classified into three major groups: expert-based, model-based, and hybrid approaches. Expert-based methods rely on human expertise to make estimations, and are the most popular technique in practice [293], [294]. Expert-based estimation however tends to require large overheads and the availability of experts each time the estimation needs to be made. Model-based approaches use data from past projects but they are also varied in terms of building customized models or using fixed models. The well-known construction cost (COCOMO) model [249] is an example of a fixed model where factors and their relationships are already defined. Such estimation models were built based on data from a range of past projects. Hence, they tend to be suitable only for a certain kinds of project that were used to build the model. The customized model building approach requires context-specific data and uses various methods such as regression (e.g. [98], [99]), Neural Network (e.g. [95], [97]), Fuzzy Logic (e.g. [96]), Bayesian Belief Networks (e.g.[250]), analogy-based (e.g. [251], [252]), and multi-objective evolutionary approaches (e.g. [7]). It is however likely that no single method will be the best performer for all project types [8], [248], [295]. Hence, some recent work (e.g. [8]) proposes to combine the estimates from multiple estimators. Hybrid approaches (e.g. [253], [254]) combine expert judgements with the available data – similarly to the notions of our proposal.

While most existing work focuses on estimating a whole project, little work has been done in building models specifically for agile projects. Today's agile, dynamic and change-driven projects require different approaches to planning and estimating [125]. Some recent approaches leverage machine learning techniques to support effort estimation for agile projects. Recently, the work in [279] proposed an approach which extracts TF-IDF features from issue description to develop an story-point estimation model. The univariate feature selection technique are then applied on the extracted features and fed into classifiers (e.g. SVM). In addition, the work in [296] applied Cosmic Function Points (CFP) [297] to estimate the effort for completing an agile project. The work in [170] developed an effort prediction model for iterative software development setting using regression models and neural networks. Differing from traditional effort estimation models, this model is built after each iteration (rather than at the end of a project) to estimate effort for the next iteration. The work in [173] built a Bayesian network model for effort prediction in software projects which adhere to the agile Extreme Programming method. Their model however relies on several parameters (e.g. process effectiveness and process improvement) that require learning and extensive fine tuning. Bayesian networks are also used in [175] to model dependencies between different factors (e.g. sprint progress and sprint planning quality influence product quality) in Scrum-based software development project in order to detect problems in the project. Our work specifically focuses on estimating issues with story points using deep learning techniques to automatically learn semantic features representing the actual meaning of issue descriptions, which is the key difference from previous work. Previous research (e.g. [130], [135], [176], [177]) has also been done in predicting the elapsed time for fixing a bug or the delay risk of resolving an issue. However, effort estimation using story points is a more preferable practice in agile development.

LSTM has shown successes in many applications such as language models [68], speech recognition [69] and video analysis [70]. Our Deep-SE is a generic in which it maps text to a numerical score or a class, and can be used for other tasks, e.g. mapping a movie review to a score, or assigning scores to essays, or sentiment analysis. Deep learning has recently attracted increasing interests in software engineering. Our previous work [298] proposed a generic deep learning framework based on LSTM for modeling software and its development process. White *et. al.* [299] has employed recurrent neural networks (RNN) to build a language model for source code. Their later work [300] extended these RNN models for detecting code clones. The work in [301] also used RNNs to build a statistical model for code completion. Our recent work [302] used LSTM to build a language model for code and demonstrated the improvement of this model compared to the one using RNNs. Gu *et. al.* [303] used a special RNN Encoder–Decoder, which consists of an encoder RNN to process the input sequence and a decoder RNN with attention to generate the output sequence. This model takes as input a given API-related natural language query and returns API usage sequences. The work in [304] also uses RNN Encoder–Decoder but for fixing common errors in C programs. Deep Belief Network [305] is another common deep learning model, which has been used in software engineering, e.g. for building defection prediction models [19], [91].

6.7 Chapter summary

In this chapter, we have proposed a deep learning-based, *fully end-to-end* prediction system for estimating story points, removing the users from manually designing features from the textual description of issues. A key novelty of our approach is the combination of two powerful deep learning architectures: Long Short-Term Memory (to learn a vector representation for issue reports) and Recurrent Highway Network (for building a deep representation). The proposed approach has consistently outperformed three common baselines and four alternatives according to our evaluation results. Compared against the Mean and Median techniques, the proposed approach has improved 34.06% and 26.77% respectively in MAE averaging across 16 projects we studied. Compared against the BoW and Doc2Vec techniques, our approach has improved 23.68% and 17.90% in MAE. These are significant results in the literature of effort estimation. A major part of those improvement were brought by our use of the deep learning LSTM architecture to model the textual description of an issue. The use of highway recurrent networks (on top of LSTM) has also improved the predictive performance, but not as significantly as the LSTM itself (especially for those project which have very small number of issues). We have discussed (informally) our work with several software developers who has been practising agile and estimating story points. They all agreed that our prediction system could be useful in practice. However, to make such a claim, we need to implement it into a tool and perform a user study. Hence, we would like to evaluate empirically the impact of our prediction system for story point estimation in practice by project managers and/or software developers. This would involve developing the model into a tool (e.g. a JIRA plugin) and then organising trial use in practice. This is an important part of our future work to confirm the ultimate benefits of our approach in general.

Chapter 7

Conclusions and future work

THIS chapter concludes the work that has been carried out in this thesis and provides some pointers to possible future lines of research.

Cost and schedule overruns are common in software projects. This is one of the major problems, regardless of which development process is employed, due to inherent dynamic nature of software development project. It is highly challenging to manage software projects to satisfy those cost and time constraints. Thus, there is increasing need for project managers and decision makers to make reliable predictions as the project progresses.

As software analytics approaches emerge since software development becomes data-rich activities, an increasing number of the applications of software analytics have been developed. These approaches focus on leverage different source of data using several techniques in machine learning to provide supports in software development. Therefore, it is important to support project managers and decision makers with insightful and actionable information.

However, there has been very little work that directly supports project management, especially at the fine-grained level of iteration and issues. Our work aims to fill that gap. In this thesis, we have empirically studied the issue reports from well-known open source projects to build predictive models using machine learning techniques (including deep learning). Our predictive analytics suite provides support in three aspects in software project management: predicting whether an issue is at risk of being delayed against its deadline, predicting delivery capability for an ongoing iteration, and estimating the effort of resolving issues. In the remainder of this chapter, we summarize our findings and contributions of this thesis and discuss future work.

7.1 Thesis contributions

We presented three predictive models to support project management tasks and decision making in software projects. The results from each extensive evaluation demonstrated the effectiveness of our predictive models. We summarize our contributions as follows:

- **Predicting delivery capability (Chapter 3)**:
  We used a data-driven approach to provide automated support for project managers and other decision makers in delivery-related risk prediction in iterative development settings. Our approach is able to predict delivery capability as to whether the target amount of work will be delivered at the end of an iteration. To build the predictive models, we extracted 15 features associated with an iteration and 12 features associated with an issue. To build a feature vector representing an iteration, we introduced three novel feature aggregation techniques: Statistical aggregation, Bag-of-words feature aggregation, and Graph-based feature aggregation to aggregate features of issues associated to an iteration. Our predictive models are built based on three state-of-the-art randomized ensemble methods: Random Forests, Stochastic Gradient Boosting Machines, and Deep Neural Networks with Dropouts. The evaluation results on five open source projects demonstrated that the proposed feature aggregation techniques can improve predictive performance. The best combination of aggregated features varies between projects. Our approach also outperforms the baseline methods: Random Guessing, Mean Effort, and Median Effort.

- **Predicting delays (Chapter 4)**:
  We have proposed the predictive model to predict if an issue will be at risk of being delayed. Our approach is also able to predict the likelihood of the risk occurring and the risk impact in terms of the extension of the delay. We have performed the study in eight well known open source projects (e.g. Apache) and extracted a comprehensive set of 19 features. We have developed a sparse logistic regression model and performed feature selection on those risk factors to choose those with good discriminative power. In feature selection, we compared two different feature selection techniques: ℓ_1-penalized logistic regression model and using p-value from logistic regression model. The outcomes from feature selection techniques also confirmed the diversity of projects where the good discriminative features are different between projects. We have however seen some common patterns that in our eight case studies the discussion time and the percentage of delayed issues that a developer is involved with have positive correlation with the outcome (e.g., the

longer discussion time, the higher probability of issues being delay). In addition, the developer's workload, the discussion time, and the percentage of delayed issues that a developer is involved with are the top-three highest discrimination power features across all projects. The evaluation results on eight open source projects demonstrate that our predictive models have a strong predictive performance.

- **Predicting delays using networked classification (Chapter 5)**:
  We have presented an approach to enhance the predictive model which predicts if an issue will be at risk of being delayed. Our approach exploits not only features specific to individual issues but also the relationships between the issues (i.e. networked data). In addition, we have proposed the techniques to extract the explicit and implicit relationships between issues in software projects to build an issue network. The explicit relationships can determine the order of issues, while the implicit relationships reflect other aspects of issue relations such as issues assigned to the same developer. We extracted 3 aspects of implicit relations: Resource-based relationship (they are assigned to the same developer), Attribute-based relationship (they affect the same component), and Content-based relationship (they descriptions share the common topics). The evaluation results demonstrate a strong predictive performance of our networked classification techniques compared to traditional approaches: achieving 49% improvement in precision, 28% improvement in recall, 39% improvement in F-measure, and 16% improvement in Area Under the ROC Curve. In particular, the stacked graphical learning approach consistently outperformed the other techniques across the five projects we studied. The results from our experiments indicate that considering the relationships between issues has an impact on the predictive performance.

- **Estimating story points (Chapter 6)**:
  We have developed a story point estimation model using deep learning techniques. Our model can estimate story points of issue reports from their textual description which requires no feature engineering. The estimation model is end-to-end trainable from raw input data (i.e. issue's textual description) which consists of two deep learning architectures: long short-term memory (LSTM) and recurrent highway network (RHWN). LSTM produces the feature vector representation of the description which can capture semantic and sequential meaning of the text, and RWHN enables the model to learn different scales of story points (e.g. Fibonacci scale) using multiple non-linear layers. We collected over 200,000 issues from 16 open source projects for training the model. We also found the positive correlation between the story points and the development time which demonstrates that the

higher story point, the longer development time. Our story point estimation model has consistently outperformed three common baselines in effort estimation: Random, Mean, and Median method and the alternatives according to our evaluation results. Our model would help maintain the consistency in effort estimation in agile planning.

7.2 Future work

In this section, we discuss possible extensions of the research work presented in this thesis.

- **Commercial setting**: We empirically studied software data generated from open source projects. We, however, cannot claim that the study covers all kinds of software projects. Future work would involve expanding the study to commercial software projects and other large open source projects to assess our proposed approach further. This replication of the study is important to address the threat to conclusion validity which focuses on the generalizability of the proposed approach. For example, important risk factors related to delays in commercial projects might be different from open source projects. Furthermore, the comparisons of our proposed approaches with similar methods is needed. For example, the proposed delay prediction model could be compared against some existing issue resolving time prediction models (e.g. [178]).

- **Industry graded tool**: The proposed approach can be developed as a tool which is integrated into an existing project management platform (e.g. JIRA software). Moreover, visualization techniques (e.g. predicting dashboard) can be applied to improve user experiences. For example, the predictive suite can be embedded into an issue tracking system to notify the users (e.g. decision maker) when issues are at risk of being delays. Furthermore, the actions of users who responded to the notified issues can be recorded for further analysis.

- **User evaluation testing**: The study focuses on measuring the predictive performance of the proposed approaches. However, a user study to demonstrate the usefulness of our models is required in the deployment phase. The evaluation from the users (e.g. project manager and decision maker) is essential to the success of deployment of our predictive models in terms of gaining the user's trust. Eliciting user requirements to better integrate the models into existing issue tracking systems is also part of the user study for future work.

- **Different sources of data**: The proposed predictive models leverage the data (i.e. issue and iteration reports) collected from issue tracking systems. There are different types of data available in software repositories which can be taken into account to build various predictive models. For example, the delay prediction model can be enhanced with the information from version control systems (e.g. GitHub).
- **Risk mitigation plan**: The ability to make accurate forecasting which we focus in this thesis is just only the first part of the activities in risk management which are risk identification and assessment. The next step involves risk mitigation where project managers come up with the plan to mitigate risks (e.g. adjusting the iteration plan to prevent the risk of under achieving iterations). Further work would involve building those prescriptive models, i.e., giving recommendation plans to deal with under/over achieving iterations or issue delays. There is a possibility of using collaborative filtering techniques to derive risk mitigation actions since open source software projects often involve large teams of contributors.

Bibliography

[1] B. Michael, S. Blumberg, and J. Laartz, "Delivering large-scale IT projects on time, on budget, and on value," Tech. Rep., 2012.

[2] P. Bright, *What windows as a service and a free upgrade mean at home and at work*, https://arstechnica.com/information-technology/2015/07/what-windows-as-a-service-and-a-free-upgrade-mean-at-home-and-at-work/, 2015.

[3] L. Williams, "What agile teams think of agile principles," *Communications of the ACM*, vol. 55, no. 4, p. 71, 2012.

[4] U. Abelein and B. Paech, "Understanding the Influence of User Participation and Involvement on System Success - a Systematic Mapping Study," *Empirical Software Engineering*, pp. 1–54, 2013.

[5] A. Mockus, D. Weiss, and P. Z. P. Zhang, "Understanding and predicting effort in software projects," in *Proceedings of the 25th International Conference on Software Engineering (ICSE)*, vol. 6, IEEE, 2003, pp. 274–284.

[6] B. Stewart, "Predicting project delivery rates using the Naive-Bayes classifier," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 14, no. 3, pp. 161–179, May 2002.

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

[8] 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.

[9] L. Huang, D. Port, L. Wang, T. Xie, and T. Menzies, "Text mining in supporting software systems risk assurance," in *Proceedings of the IEEE/ACM international conference on Automated software engineering (ASE)*, New York, USA: ACM Press, 2010, pp. 163–167.

[10] Z. Xu, B. Yang, and P. Guo, “Software Risk Prediction Based on the Hybrid Algorithm of Genetic Algorithm and Decision Tree,” in *Proceedings of International Conference on Intelligent Computing (ICIC)*, 2007, pp. 266–274.

[11] Y. Hu, X. Zhang, E. Ngai, R. Cai, and M. Liu, “Software project risk analysis using Bayesian networks with causality constraints,” *Decision Support Systems*, vol. 56, pp. 439–449, 2013.

[12] C. Fang and F. Marle, “A simulation-based risk network model for decision support in project risk management,” *Decision Support Systems*, vol. 52, no. 3, pp. 635–644, 2012.

[13] J. Zhou, H. Zhang, and D. Lo, “Where Should the Bugs Be Fixed ?” *Proceedings of the 34th International Conference on Software Engineering (ICSE)*, pp. 14–24, 2012.

[14] S. Wang, D. Lo, and J. Lawall, “Compositional vector space models for improved bug localization,” in *Proceedings of the 30th International Conference on Software Maintenance and Evolution (ICSME)*, 2014, pp. 171−180.

[15] A. N. Lam, A. T. Nguyen, H. A. Nguyen, and T. N. Nguyen, “Bug Localization with Combination of Deep Learning and Information Retrieval,” in *Proceedings of the 25th IEEE/ACM International Conference on Program Comprehension (ICPC)*, 2017, pp. 218–229.

[16] A. N. Lam, A. T. Nguyen, H. A. Nguyen, and T. N. Nguyen, “Combining deep learning with information retrieval to localize buggy files for bug reports,” in *Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2016, pp. 476–481.

[17] A. T. Nguyen, T. T. Nguyen, J. Al-Kofahi, H. V. Nguyen, and T. N. Nguyen, “A topic-based approach for narrowing the search space of buggy files from a bug report,” in *Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2011, pp. 263–272.

[18] F. Zhang, A. Mockus, I. Keivanloo, and Y. Zou, “Towards building a universal defect prediction model with rank transformed predictors,” *Empirical Software Engineering*, vol. 21, no. 5, pp. 2107–2145, Aug. 2015.

[19] X. Yang, D. Lo, X. Xia, Y. Zhang, and J. Sun, “Deep Learning for Just-in-Time Defect Prediction,” in *Proceedings of the IEEE International Conference on Software Quality, Reliability and Security (QRS)*, 2015, pp. 17–26.

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

[21] A. F. Otoom, D. Al-shdaifat, M. Hammad, and E. E. Abdallah, "Severity Prediction of Software Bugs," in *Proceedings of the 7th International Conference on Information and Communication Systems (ICICS)*, 2016, pp. 92–95.

[22] T. Menzies and A. Marcus, "Automated severity assessment of software defect reports," in *Proceedings of the International Conference on Software Maintenance (ICSM)*, IEEE, 2008, pp. 346–355.

[23] A. Lamkanfi, S. Demeyer, E. Giger, and B. Goethals, "Predicting the severity of a reported bug," in *Proceedings of the 7th IEEE Working Conference on Mining Software Repositories (MSR)*, IEEE, 2010, pp. 1–10.

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

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

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

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

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

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

[30] M. Usman, E. Mendes, F. Weidt, and R. Britto, "Effort Estimation in Agile Software Development: A Systematic Literature Review," in *Proceedings of the 10th International Conference on Predictive Models in Software Engineering (PROMISE)*, 2014, pp. 82–91.

[31] C. E. Brodley, U. Rebbapragada, K. Small, and B. Wallace, "Challenges and Opportunities in Applied Machine Learning," *AI Magazine*, vol. 33, no. 1, pp. 11–24, 2012.

[32] E. Blanzieri and A. Bryl, "A survey of learning-based techniques of email spam filtering," *Artificial Intelligence Review*, vol. 29, no. 1, pp. 63–92, 2008.

[33] K. Rieck, P. Trinius, C. Willems, and T. Holz, "Automatic analysis of malware behavior using machine learning," *Journal of Computer Security*, vol. 19, no. 4, pp. 639–668, 2011.

[34] O. Chapelle, P. Haffner, and V. N. Vapnik, "Support vector machines for histogram-based image classification," *IEEE transactions on Neural Networks*, vol. 10, no. 5, pp. 1055–1064, 1999.

[35] C. M. Bishop, *Pattern Recognition and Machine Learning*, 9. 2013, vol. 53, pp. 1689–1699. arXiv: `arXiv:1011.1669v3`.

[36] K. Christensen, S. Nørskov, L. Frederiksen, and J. Scholderer, "In search of new product ideas: Identifying ideas in online communities by machine learning and text mining," *Creativity and Innovation Management*, vol. 26, no. 1, pp. 17–30, 2017.

[37] K. Siau and Y. Yang, "Impact of Artificial Intelligence, Robotics, and Machine Learning on Sales and Marketing," in *Twelve Annual Midwest Association for Information Systems Conference (MWAIS 2017)*, 2017, pp. 18–19.

[38] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016.

[39] J. R. Quinlan, *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., Mar. 1993.

[40] R. Conforti, M. de Leoni, M. La Rosa, W. M. van der Aalst, and A. H. ter Hofstede, "A recommendation system for predicting risks across multiple business process instances," *Decision Support Systems*, vol. 69, pp. 1–19, Jan. 2015.

[41] W. M. Ibrahim, N. Bettenburg, E. Shihab, B. Adams, and A. E. Hassan, "Should I contribute to this discussion?" In *Proceedings of the 7th IEEE Working Conference on Mining Software Repositories (MSR)*, Ieee, May 2010, pp. 181–190.

[42] L. Breiman, "Random forests," *Machine learning*, pp. 5–32, 2001.

[43] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings," *IEEE Transactions on Software Engineering*, vol. 34, no. 4, pp. 485–496, Jul. 2008.

[44] Y. Kamei, S. Matsumoto, A. Monden, K.-i. Matsumoto, B. Adams, and A. E. Hassan, "Revisiting common bug prediction findings using effort-aware models," English, in *Proceedings of the IEEE International Conference on Software Maintenance (ICSM)*, IEEE, Sep. 2010, pp. 1–10.

[45] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Annals of Statistics*, vol. 29, no. 5, pp. 1189–1232, 2001.

[46] J. H. Friedman, "Stochastic gradient boosting," *Computational Statistics & Data Analysis*, vol. 38, no. 4, pp. 367–378, 2002.

[47] K. Crammer and Y. Singer, "On the Algorithmic Implementation of Multiclass Kernel-based Vector Machines," *Journal of Machine Learning Research*, vol. 2, pp. 265–292, 2001.

[48] D. D. Lewis, "Naive (Bayes) at forty: The independence assumption in information retrieval," in *European conference on machine learning*, Springer, 1998, pp. 4–15.

[49] D. M. Grether, "Bayes rule as a descriptive model: The representativeness heuristic," *The Quarterly Journal of Economics*, vol. 95, no. 3, pp. 537–557, 1980.

[50] J. Wolfson, S. Bandyopadhyay, M. Elidrisi, G. Vazquez-Benitez, D. Musgrove, G. Adomavicius, P. Johnson, and P. O'Connor, "A Naive Bayes machine learning approach to risk prediction using censored, time-to-event data," *Statistics in medicine*, pp. 21–42, Apr. 2014.

[51] W. Abdelmoez, M. Kholief, and F. M. Elsalmy, "Bug Fix-Time Prediction Model Using Naïve Bayes Classifier," in *Proceedings of the 22nd International Conference on Computer Theory and Applications (ICCTA)*, 2012, pp. 13–15.

[52] R. Kohavi, "Scaling Up the Accuracy of Naive-Bayes Classifiers: A Decision-Tree Hybrid," in *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD)*, 1996, pp. 202–207.

[53] Y. Bengio, "Artificial Neural Networks and Their Application to Sequence Recognition," PhD thesis, Montreal, Que., Canada, Canada, 1991.

[54] D. F. Specht, "Probabilistic neural networks," *Neural networks*, vol. 3, no. 1, pp. 109–118, 1990.

[55] Y. Hu, J. Huang, J. Chen, M. Liu, K. Xie, and S. Yat-sen, "Software Project Risk Management Modeling with Neural Network and Support Vector Machine Approaches," in *Proceedings of the 3rd International Conference on Natural Computation (ICNC)*, vol. 3, 2007, pp. 358–362.

[56] D. Neumann, "An enhanced neural network technique for software risk analysis," *IEEE Transactions on Software Engineering*, vol. 28, no. 9, pp. 904–912, Sep. 2002.

[57] Q. Wang, J. Zhu, and B. Yu, "Combining Classifiers in Software Quality Prediction : A Neural Network Approach," in *Proceedings of the 2nd International Symposium on Neural Networks*, Springer Berlin Heidelberg, 2005, pp. 921–926.

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

[59] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[60] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[61] L. Deng and D. Yu, "Deep Learning: Methods and Applications," Tech. Rep., May 2014.

[62] Y. LeCun, Y. Bengio, *et al.*, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, vol. 3361, no. 10, pp. 255–258, 1998.

[63] M. Ranzato and G. E. Hinton, "Modeling pixel means and covariances using factorized third-order boltzmann machines," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, IEEE, IEEE, Jun. 2010, pp. 2551–2558.

[64] J. F. Kolen and S. C. Kremer, "Gradient Flow in Recurrent Nets: The Difficulty of Learning LongTerm Dependencies," in *A Field Guide to Dynamical Recurrent Networks*, Wiley-IEEE Press, 2009, pp. 237–243. arXiv: `arXiv:1011.1669v3`.

[65] X. He and L. Deng, "Speech recognition, machine translation, and speech translation-a unified discriminative learning paradigm," *IEEE Signal Processing Magazine*, vol. 28, no. 5, pp. 126–133, 2011.

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

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

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

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

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

[71] Project Management Institute Inc, *A guide to the project management body of knowledge (PMBOK guide)*. 2000, p. 211. arXiv: `978-1-933890-51-7`.

[72] C. Jones, "Software Project Management Practices : Failure Versus Success," *CrossTalk: The Journal of Defense Software Engineering*, vol. 17, no. 10, pp. 5–9, 2004.

[73] A. L. Lederer and J. Prasad, "Nine management guidelines for better cost estimating," *Communications of the ACM*, vol. 35, no. 2, pp. 51–59, 1992.

[74] H. Kerzner and H. R. Kerzner, *Project management: a systems approach to planning, scheduling, and controlling*. John Wiley & Sons, 2017.

[75] G. Antoniol, K. Ayari, M. Di Penta, F. Khomh, and Y.-G. Guéhéneuc, "Is It a Bug or an Enhancement?: A Text-based Approach to Classify Change Requests," in *Proceedings of the Conference of the Center for Advanced Studies on Collaborative Research: Meeting of Minds*, ACM, 2008, pp. 304–318.

[76] D. Alencar, S. L. Abebe, S. Mcintosh, D. Alencar da Costa, S. L. Abebe, S. Mcintosh, U. Kulesza, and A. E. Hassan, "An Empirical Study of Delays in the Integration of Addressed Issues," in *Proceedings of the International Conference on Software Maintenance and Evolution (ICSME)*, IEEE, 2014, pp. 281–290.

[77] G. Murphy and D. Čubranić, "Automatic bug triage using text categorization," in *Proceedings of the 16th International Conference on Software Engineering & Knowledge Engineering (SEKE)*, 2004, pp. 92–97.

[78] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?" In *Proceedings of the 28th International Conference on Software engineering (ICSE)*, New York, USA: ACM Press, May 2006, pp. 361–370.

[79] Y. Tian, D. Lo, X. Xia, and C. Sun, "Automated prediction of bug report priority using multi-factor analysis," *Empirical Software Engineering*, vol. 20, no. 5, pp. 1354–1383, 2015.

[80] P. Bourque and R. E. Fairley, *Guide to the Software Engineering - Body of Knowledge.* 2014, p. 346. arXiv: `arXiv:1210.1833v2`.

[81] T. Menzies and T. Zimmermann, "Software Analytics: So What?" *IEEE Software*, vol. 30, no. 4, pp. 31–37, Jul. 2013.

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

[83] R. P. L. Buse and T. Zimmermann, "Information needs for software development analytics," in *Proceedings of the 34th International Conference on Software Engineering (ICSE)*, IEEE, Jun. 2012, pp. 987–996.

[84] O. Baysal, R. Holmes, and M. W. Godfrey, "Developer dashboards: The need for qualitative analytics," *IEEE Software*, vol. 30, no. 4, pp. 46–52, 2013.

[85] A. E. Hassan, A. Hindle, M. Shepperd, C. O. Brian, K. Enjoy, and T. Menzies, "Roundtable: What's Next in Software Analytics," *IEEE Software*, vol. 30, no. 4, pp. 53–56, 2013.

[86] D. Zhang, Y. Dang, J.-G. Lou, S. Han, H. Zhang, and T. Xie, "Software analytics as a learning case in practice," in *Proceedings of the International Workshop on Machine Learning Technologies in Software Engineering (MALETS)*, 2011, pp. 55–58.

[87] R. P. Buse and T. Zimmermann, “Analytics for software development,” in *Proceedings of the FSE/SDP workshop on Future of software engineering research (FoSER)*, 2010, pp. 77–81.

[88] D. Zhang, S. Han, Y. Dang, J.-G. Lou, H. Zhang, and T. Xie, “Software Analytics in Practice,” *IEEE Software*, vol. 30, no. 5, pp. 30–37, 2013.

[89] N. R. Darwish, A. A. Mohamed, and A. S. Abdelghany, “A Hybrid Machine Learning Model for Selecting Suitable Requirements Elicitation Techniques A Hybrid Machine Learning Model for Selecting Suitable Requirements Elicitation Techniques,” *International Journal of Computer Science and Information Security*, vol. 14, no. 6, pp. 380–391, 2016.

[90] G. Grieco, G. L. Grinblat, L. Uzal, S. Rawat, J. Feist, and L. Mounier, “Toward Large-Scale Vulnerability Discovery using Machine Learning,” in *Proceedings of the 6th ACM Conference on Data and Application Security and Privacy (CODASPY)*, 2016, pp. 85–96.

[91] S. Wang, T. Liu, and L. Tan, “Automatically learning semantic features for defect prediction,” in *Proceedings of the International Conference on Software Engineering (ICSE)*, vol. 14-22, 2016, pp. 297–308.

[92] X. Xia, D. Lo, Y. Ding, J. M. Al-Kofahi, T. N. Nguyen, and X. Wang, “Improving Automated Bug Triaging with Specialized Topic Model,” *IEEE Transactions on Software Engineering*, vol. 43, no. 3, pp. 272–297, 2016.

[93] H. K. Dam, T. Pham, S. W. Ng, T. Tran, J. Grundy, A. Ghose, T. Kim, and C.-J. Kim, “A deep tree-based model for software defect prediction,” *arXiv:1802.00921*, no. February, 2018. arXiv: `1802.00921`.

[94] T. Menzies, E. Kocagüneli, L. Minku, F. Peters, and B. Turhan, “Effort Estimation,” in *Sharing Data and Models in Software Engineering*, Elsevier, 2015, pp. 47–50.

[95] S. Kanmani, J. Kathiravan, S. S. Kumar, and M. Shanmugam, “Neural Network Based Effort Estimation Using Class Points for OO Systems,” in *Proceedings of the International Conference on Computing: Theory and Applications (ICCTA)*, IEEE, Mar. 2007, pp. 261–266.

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

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

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

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

[100] B. Boehm, "Software risk management: principles and practices," *Software, IEEE*, vol. 8, no. 1, pp. 32–41, 1991.

[101] A. Pika, W. M. van der Aalst, C. J. Fidge, A. H. ter Hofstede, M. T. Wynn, and W. V. D. Aalst, "Profiling event logs to configure risk indicators for process delays," in *Proceedings of the 25th International Conference on Advanced Information Systems Engineering (CAiSE)*, Springer, Jul. 2013, pp. 465–481.

[102] C.-P. Chang, "Mining software repositories to acquire software risk knowledge," in *Proceedings of the 14th International Conference on Information Reuse & Integration (IRI)*, IEEE, Aug. 2013, pp. 489–496.

[103] P. S. Kochhar, F. Thung, and D. Lo, "Automatic fine-grained issue report reclassification," in *Proceedings of the IEEE International Conference on Engineering of Complex Computer Systems (ICECCS)*, 2014, pp. 126–135.

[104] K. Somasundaram and G. C. Murphy, "Automatic categorization of bug reports using latent Dirichlet allocation," in *Proceedings of the 5th India Software Engineering Conference (ISEC)*, 2012, pp. 125–130.

[105] A. Sureka, "Learning to Classify Bug Reports into Components," in *Proceedings of the 50th international conference on Objects, Models, Components, Patterns*, Springer, 2012, pp. 288–303.

[106] S. Vargas-Baldrich, M. Linares-Vásquez, and D. Poshyvanyk, "Automated tagging of software projects using bytecode and dependencies," in *Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2016, pp. 289–294.

[107] A. Lamkanfi, S. Demeyer, Q. D. Soetens, and T. Verdonckz, "Comparing mining algorithms for predicting the severity of a reported bug," in *Proceedings of the European Conference on Software Maintenance and Reengineering (CSMR)*, 2011, pp. 249–258.

[108] R. Jindal, R. Malhotra, and A. Jain, "Prediction of defect severity by mining software project reports," *International Journal of System Assurance Engineering and Management*, vol. 8, no. 2, pp. 334–351, 2017.

[109] J. Xuan, H. Jiang, Z. Ren, and W. Zou, "Developer prioritization in bug repositories," in *Proceedings of the 34th International Conference on Software Engineering (ICSE)*, 2012, pp. 25–35.

[110] R. Robbes and D. Rothlisberger, "Using developer interaction data to compare expertise metrics," in *Proceedings of the 10th Working Conference on Mining Software Repositories (MSR)*, IEEE, May 2013, pp. 297–300.

[111] T. Zimmermann, N. Nagappan, P. J. Guo, and B. Murphy, "Characterizing and predicting which bugs get reopened," in *Proceedings of the 34th International Conference on Software Engineering (ICSE)*, IEEE Press, Jun. 2012, pp. 1074–1083.

[112] X. Xia, D. Lo, E. Shihab, X. Wang, and B. Zhou, "Automatic, high accuracy prediction of reopened bugs," *Automated Software Engineering*, vol. 22, no. 1, pp. 75–109, 2014.

[113] E. Shihab, A. Ihara, Y. Kamei, W. M. Ibrahim, M. Ohira, B. Adams, A. E. Hassan, and K.-i. Matsumoto, "Studying re-opened bugs in open source software," *Empirical Software Engineering*, vol. 18, no. 5, pp. 1005–1042, Sep. 2012.

[114] W. Maalej, R. Tiarks, T. Roehm, and R. Koschke, "On the Comprehension of Program Comprehension," *ACM Transactions on Software Engineering and Methodology*, vol. 23, no. 4, pp. 1–37, 2014.

[115] A. E. Hassan, "The road ahead for Mining Software Repositories," in *Frontiers of Software Maintenance*, IEEE, Sep. 2008, pp. 48–57.

[116] D. Čubranić, G. C. Murphy, J. Singer, and K. S. Booth, "Hipikat: A project memory for software development," *IEEE Transactions on Software Engineering*, vol. 31, no. 6, pp. 446–465, 2005.

[117] A. E. Hassan and R. C. Holt, "Using development history sticky notes to understand software architecture," in *Proceedings of the 12th IEEE International Workshop on Program Comprehension*, Jun. 2004, pp. 183–192.

[118] C. Bird, A. Gourley, P. Devanbu, M. Gertz, and A. Swaminathan, "Mining email social networks," in *Proceedings of the International Workshop on Mining Software Repositories (MSR)*, ACM, 2006, pp. 137–143.

[119] P. C. Rigby and A. E. Hassan, "What Can OSS Mailing Lists Tell Us? A Preliminary Psychometric Text Analysis of the Apache Developer Mailing List," in *Proceedings of the 4th International Workshop on Mining Software Repositories (MSR)*, May 2007, pp. 23–30.

[120] M. Harman, Y. Jia, and Y. Zhang, "App store mining and analysis: MSR for app stores," in *Proceedings of the International Working Conference on Mining Software Repositories (MSR)*, 2012, pp. 108–111.

[121] I. J. Mojica, M. Nagappan, B. Adams, and A. E. Hassan, "Understanding Reuse in the Android Market," in *Proceedings of the 20th IEEE International Conference on Program Comprehension (ICPC)*, 2012, pp. 113–122.

[122] A. S. Sayyad, T. Menzies, and H. Ammar, "On the value of user preferences in search-based software engineering: A case study in software product lines," in *Proceedings of the 35th International Conference on Software Engineering (ICSE)*, 2013, pp. 492–501.

[123] F. Palma, H. Farzin, Y. G. Guéhéneuc, and N. Moha, "Recommendation system for design patterns in software development: An DPR overview," in *Proceedings of the 3rd International Workshop on Recommendation Systems for Software Engineering (RSSE)*, 2012, pp. 1–5.

[124] A. Hindle, "Green mining: a methodology of relating software change and configuration to power consumption," *Empirical Software Engineering*, vol. 20, no. 2, pp. 374–409, 2015.

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

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

[127] M. Choetkiertikul, H. K. Dam, T. Tran, and A. Ghose, "Characterization and prediction of issue-related risks in software projects," in *Proceedings of the 12th Working Conference on Mining Software Repositories (MSR)*, IEEE, 2015, pp. 280–291.

[128] S. Kaufman and C. Perlich, "Leakage in Data Mining : Formulation , Detection , and Avoidance," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 6(4), no. 15, pp. 556–563, 2012.

[129] D. R. McCallum and J. L. Peterson, "Computer-based readability indexes," in *Proceedings of the ACM Conference*, ACM, 1982, pp. 44–48.

[130] L. D. Panjer, "Predicting Eclipse Bug Lifetimes," in *Proceedings of the 4th International Workshop on Mining Software Repositories (MSR)*, 2007, pp. 29–32.

[131] X. Xia, D. Lo, E. Shihab, X. Wang, and X. Yang, "ELBlocker: Predicting blocking bugs with ensemble imbalance learning," *Information and Software Technology*, vol. 61, pp. 93–106, 2015.

[132] M. Choetkiertikul, H. K. Dam, T. Tran, and A. Ghose, "Predicting delays in software projects using networked classification," in *Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2015, pp. 353–364.

[133] H. Valdivia Garcia, E. Shihab, and H. V. Garcia, "Characterizing and predicting blocking bugs in open source projects," in *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR)*, ACM Press, 2014, pp. 72–81.

[134] X. Xia, D. Lo, M. Wen, E. Shihab, and B. Zhou, "An empirical study of bug report field reassignment," in *Proceedings of the Conference on Software Maintenance, Reengineering, and Reverse Engineering*, 2014, pp. 174–183.

[135] P. Hooimeijer and W. Weimer, "Modeling bug report quality," in *Proceedings of the 22 IEEE/ACM international conference on Automated software engineering (ASE)*, ACM Press, Nov. 2007, pp. 34–44.

[136] P. Tirilly, V. Claveau, and P. Gros, "Language modeling for bag-of-visual words image categorization," in *Proceedings of the International Conference on Content-based Image and Video Retrieval*, 2008, pp. 249–258.

[137] E. Zegura, K. Calvert, and S. Bhattacharjee, "How to model an internetwork," in *Proceedings of the 15th Annual Joint Conference of the IEEE Computer Societies Networking the Next Generation (INFOCOM)*, vol. 2, 1996, pp. 594–602.

[138] P. Bhattacharya, M. Iliofotou, I. Neamtiu, and M. Faloutsos, "Graph-based Analysis and Prediction for Software Evolution," in *Proceedings of the 34th International Conference on Software Engineering (ICSE)*, IEEE Press, 2012, pp. 419–429.

[139] L. Rokach, "Taxonomy for characterizing ensemble methods in classification tasks: A review and annotated bibliography," *Computational Statistics and Data Analysis*, vol. 53, no. 12, pp. 4046–4072, 2009.

[140] D. O. Maclin and R., "Popular Ensemble Methods: An Empirical Study," *Journal of Artificial Intelligence Research*, vol. 11, pp. 169–198, 1999.

[141] T. Dietterich, "Ensemble methods in machine learning," in *Proceedings of the 1st International Workshop on Multiple Classifier Systems*, vol. 1857, Springer, 2000, pp. 1–15.

[142] T. G. Dietterich, "An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization," *Machine learning*, vol. 40, no. 2, pp. 139–157, 2000.

[143] M. Galar, A. Fernandez, E. Barrenechea, H. Bustince, and F. Herrera, "A review on ensembles for the class imbalance problem: Bagging-, boosting-, and hybrid-based approaches," *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews*, vol. 42, no. 4, pp. 463–484, 2012.

[144] S. M. Halawani, I. A. Albidewi, and A. Ahmad, "A Novel Ensemble Method for Regression via Classification Problems," *Journal of Computer Science*, vol. 7, no. 3, pp. 387–393, 2011.

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

[146] J. H. Friedman, “Greedy function approximation: A gradient boosting machine,” *Annals of Statistics*, vol. 29, no. 5, pp. 1189–1232, 2001.

[147] X. Wang and A. McCallum, “Topics over time: A non-markov continuous-time model of topical trends,” in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, 2006, pp. 424–433.

[148] M. Fernández-Delgado, E. Cernadas, S. Barro, D. Amorim, and D. Amorim Fernández-Delgado, “Do we Need Hundreds of Classifiers to Solve Real World Classification Problems?” *Journal of Machine Learning Research*, vol. 15, pp. 3133–3181, 2014.

[149] K. H. Esbensen and P. Geladi, “Principles of proper validation: Use and abuse of re-sampling for validation,” *Journal of Chemometrics*, vol. 24, pp. 168–187, 2010.

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

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

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

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

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

[155] K. Goldberg, T. Roeder, D. Gupta, and C. Perkins, “Eigentaste: A Constant Time Collaborative Filtering Algorithm,” *Information Retrieval*, vol. 4, no. 2, pp. 133–151, 2001. arXiv: `0005074v1 [arXiv:astro-ph]`.

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

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

[158] S. Baccianella, A. Esuli, and F. Sebastiani, "Evaluation measures for ordinal regression," in *Proceedings of the 9th International Conference on Intelligent Systems Design and Applications (ISDA)*, IEEE, 2009, pp. 283–287.

[159] D. M. W. Powers, "Evaluation: From Precision, Recall and F-Factor to ROC, Informedness, Markedness & Correlation David," *Journal of Machine Learning Technologies*, vol. 2, no. 1, pp. 37–63, 2011.

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

[161] R. M. Everson and J. E. Fieldsend, "Multi-class ROC analysis from a multi-objective optimisation perspective," *Pattern Recognition Letters*, vol. 27, no. 8, pp. 918–927, 2006.

[162] B. W. Matthews, "Comparison of the predicted and observed secondary structure of T4 phage lysozyme," *Biochimica et Biophysica Acta (BBA)-Protein Structure*, vol. 405, no. 2, pp. 442–451, 1975.

[163] G. Jurman, S. Riccadonna, and C. Furlanello, "A comparison of MCC and CEN error measures in multi-class prediction," *PLoS ONE*, vol. 7, no. 8, pp. 1–8, 2012. arXiv: `1008.2908`.

[164] R. Genuer, J.-M. Poggi, and C. Tuleau-Malot, "Variable selection using random forests," *Pattern Recognition Letters*, vol. 31, no. 14, pp. 2225–2236, 2010.

[165] K. Tumer and N. C. Oza, "Input decimated ensembles," *Pattern Analysis and Applications*, vol. 6, no. 1, pp. 65–77, 2003.

[166] K. Tumer and J. Ghosh, "Error Correlation and Error Reduction in Ensemble Classifiers," *Connection Science*, vol. 8, no. 3-4, pp. 385–404, 1996.

[167] E. Tuv, "Feature Selection with Ensembles , Artificial Variables , and Redundancy Elimination," *Journal of Machine Learning Research*, vol. 10, pp. 1341–1366, 2009.

[168] M. Shepperd and S. MacDonell, "Evaluating prediction systems in software project estimation," *Information and Software Technology*, vol. 54, no. 8, pp. 820–827, 2012.

[169] M. Usman, E. Mendes, and J. Börstler, "Effort estimation in agile software development: A survey on the state of the practice," in *Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering*, Nanjing, China, 2015, pp. 1–10.

[170] P. Abrahamsson, R. Moser, W. Pedrycz, A. Sillitti, and G. Succi, "Effort prediction in iterative software development processes – incremental versus global prediction models," in *Proceeding of the 1st International Symposium on Empir-*

ical Software Engineering and Measurement (ESEM), IEEE Computer Society, 2007, pp. 344–353.

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

[172] O. Benediktsson, D. Dalcher, K. Reed, and M. Woodman, "COCOMO-Based Effort Estimation for Iterative and Incremental Software Development," *Software Quality Journal*, vol. 11, pp. 265–281, 2003.

[173] P. Hearty, N. Fenton, D. Marquez, and M. Neil, "Predicting Project Velocity in XP Using a Learning Dynamic Bayesian Network Model," *IEEE Transactions on Software Engineering*, vol. 35, no. 1, pp. 124–137, 2009.

[174] R. Torkar, N. M. Awan, A. K. Alvi, and W. Afzal, "Predicting software test effort in iterative development using a dynamic Bayesian network," in *Proceedings of the 21st IEEE International Symposium on Software Reliability Engineering – (Industry Practice Track)*, IEEE, 2010.

[175] M. Perkusich, H. De Almeida, and A. Perkusich, "A model to detect problems on scrum-based software development projects," *The ACM Symposium on Applied Computing*, pp. 1037–1042, 2013.

[176] P. Bhattacharya and I. Neamtiu, "Bug-fix time prediction models: can we do better?" In *Proceedings of the 8th working conference on Mining software repositories (MSR)*, ACM, 2011, pp. 207–210.

[177] E. Giger, M. Pinzger, and H. Gall, "Predicting the fix time of bugs," in *Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering (RSSE)*, ACM, 2010, pp. 52–56.

[178] L. Marks, Y. Zou, and A. E. Hassan, "Studying the fix-time for bugs in large open source projects," in *Proceedings of the 7th International Conference on Predictive Models in Software Engineering (Promise)*, ACM Press, 2011, pp. 1–8.

[179] C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller, "How Long Will It Take to Fix This Bug?" In *Proceedings of the 4th International Workshop on Mining Software Repositories (MSR)*, 2007, pp. 1–8.

[180] N. Bettenburg and A. E. Hassan, "Studying the impact of dependency network measures on software quality," in *Proceedings of the International Conference on Software Maintenance (ICSM)*, 2012, pp. 1–10.

[181] T. Zimmermann and N. Nagappan, "Predicting defects using network analysis on dependency graphs," in *Proceedings of the 13th international conference on Software engineering (ICSE)*, 2008, pp. 531–540.

[182] J. Anvik and G. C. Murphy, "Reducing the effort of bug report triage," *ACM Transactions on Software Engineering and Methodology*, vol. 20, no. 3, pp. 1–35, 2011.

[183] M. M. Rahman, G. Ruhe, and T. Zimmermann, "Optimized assignment of developers for fixing bugs an initial evaluation for eclipse projects," in *Proceedings of the 3rd International Symposium on Empirical Software Engineering and Measurement (ESEM)*, IEEE, Oct. 2009, pp. 439–442.

[184] P. Runeson, M. Alexandersson, and O. Nyholm, "Detection of Duplicate Defect Reports Using Natural Language Processing," in *Proceedings of the 29th International Conference on Software Engineering (ICSE)*, IEEE, 2007, pp. 499–510.

[185] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun, "An approach to detecting duplicate bug reports using natural language and execution information," in *Proceedings of the 30th International Conference on Software Engineering (ICSE)*, 2008, pp. 461–470.

[186] N. Bettenburg, R. Premraj, and T. Zimmermann, "Duplicate bug reports considered harmful . . . really?" In *Proceedings of the International Conference on Software Maintenance (ICSM)*, 2008, pp. 337–345.

[187] C. Sun, D. Lo, S. C. Khoo, and J. Jiang, "Towards more accurate retrieval of duplicate bug reports," in *Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, IEEE, 2011, pp. 253–262.

[188] N. Jalbert and W. Weimer, "Automated duplicate detection for bug tracking systems," in *Proceedings of the International Conference on Dependable Systems and Networks With FTCS and DCC (DSN)*, IEEE, 2008, pp. 52–61.

[189] A. T. Nguyen, T. T. T. N. Nguyen, D. Lo, and C. Sun, "Duplicate bug report detection with a combination of information retrieval and topic modeling," in *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2012, pp. 70–79.

[190] F. Palomba, G. Bavota, M. Di Penta, R. Oliveto, A. De Lucia, and D. Poshyvanyk, "Detecting bad smells in source code using change history information," in *Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2013, pp. 268–278.

[191] T. Jiang, L. Tan, and S. Kim, "Personalized defect prediction," in *Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2013, pp. 279–289.

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

[193] S. Group, “CHAOS Report,” West Yarmouth, Massachusetts: Standish Group, Tech. Rep., 2004.

[194] M. J. Carr and S. L. Konda, “Taxonomy-Based Risk Identification,” Software Engineering Institute, Carnegie Mellon University, Tech. Rep. June, 1993.

[195] P. J. Guo, T. Zimmermann, N. Nagappan, and B. Murphy, “Characterizing and predicting which bugs get fixed: an empirical study of Microsoft Windows,” *Proceedings of the 32nd International Conference on Software Engineering (ICSE)*, pp. 495–504, 2010.

[196] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann, “What makes a good bug report?” In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, New York, USA: ACM Press, Nov. 2008, pp. 308–318.

[197] D. M. Blei, A. Y. Ng, and M. I. Jordan, “Latent Dirichlet Allocation,” *Journal of Machine Learning Research*, vol. 3, no. 4-5, pp. 993–1022, 2012.

[198] W.-M. Han and S.-J. Huang, “An empirical analysis of risk components and performance on software projects,” *Journal of Systems and Software*, vol. 80, no. 1, pp. 42–50, Jan. 2007.

[199] A. A. Porter, H. P. Siy, and L. G. Votta, “Understanding the effects of developer activities on inspection interval,” in *Proceedings of the 19th international conference on Software engineering (ICSE)*, ACM Press, May 1997, pp. 128–138.

[200] L. Wallace and M. Keil, “Software project risks and their effect on outcomes,” *Communications of the ACM*, vol. 47, no. 4, pp. 68–73, Apr. 2004.

[201] S. Kim, T. Zimmermann, K. Pan, and E. Jr. Whitehead, “Automatic Identification of Bug-Introducing Changes,” in *Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering (ASE)*, IEEE, Sep. 2006, pp. 81–90.

[202] I. Guyon and A. Elisseeff, “An Introduction to Variable and Feature Selection,” *The Journal of Machine Learning Research*, vol. 3, pp. 1157–1182, 2003.

[203] S. Menard, *Applied logistic regression analysis, 2nd edition*. SAGE University paper, 2002, vol. 106.

[204] D. A. Belsley, E. Kuh, and R. E. Welsch, *Regression diagnostics: Identifying influential data and sources of collinearity*. John Wiley & Sons, 2005, vol. 571.

[205] A. K. Cline and I. S. Dhillon, *Computation of the Singular Value Decomposition*. 2006, 45(1)–45(14).

[206] S.-I. Lee, H. Lee, P. Abbeel, and A. Y. Ng, “Efficient l1 regularized logistic regression,” in *Proceedings of the National Conference on Artificial Intelligence*, Menlo

Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, vol. 21, 2006, pp. 401–409.

[207] B. Zadrozny and C. Elkan, "Transforming classifier scores into accurate multi-class probability estimates," in *Proceedings of the 8th ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, 2002, pp. 694–699.

[208] N. Chawla and D. Cieslak, "Evaluating probability estimates from decision trees," in *American Association for Artificial Intelligence (AAAI)*, 2006, pp. 1–6.

[209] A. Garg and D. Roth, "Understanding Probabilistic Classifiers," in *Machine Learning: Lecture Notes in Computer Science*, ser. Lecture Notes in Computer Science, vol. 2167, 2001, pp. 179–191.

[210] L.-M. Wang, X.-L. Li, C.-H. Cao, and S.-M. Yuan, "Combining decision tree and Naive Bayes for classification," *Knowledge-Based Systems*, vol. 19, no. 7, pp. 511–515, Nov. 2006.

[211] B. Boehm, *Software risk management*. Springer, 1989, pp. 1–19.

[212] A. Iqbal, "Understanding Contributor to Developer Turnover Patterns in OSS Projects : A Case Study of Apache Projects," *ISRN Software Engineering*, vol. 2014, pp. 10–20, 2014.

[213] V. J. Hodge and J. Austin, "A Survey of Outlier Detection Methodoligies," *Artificial Intelligence Review*, vol. 22, no. 1969, pp. 85–126, 2004.

[214] D. H. Jr and S. Lemeshow, *Applied logistic regression, 3rd edition*. Wiley, 2004.

[215] X. Lam, T. Vu, and T. Le, "Addressing cold-start problem in recommendation systems," in *Proceedings of the 2nd international conference on Ubiquitous information management and communication*, 2008, pp. 208–211.

[216] X. Qin, M. Salter-Townshend, and P. Cunningham, "Exploring the Relationship between Membership Turnover and Productivity in Online Communities," in *Proceedings of the 8th International AAAI Conference on Weblogs and Social Media*, 2014, pp. 406–415.

[217] Xu Ruzhi, Q. leqiu, and Jing Xinhai, "CMM-based software risk control optimization," in *Proceedings of the 5th IEEE Workshop on Mobile Computing Systems and Applications*, IEEE, 2003, pp. 499–503.

[218] E. Letier, D. Stefan, and E. T. Barr, "Uncertainty, risk, and information value in software requirements and architecture," in *Proceedings of the 36th International Conference on Software Engineering (ICSE)*, New York, USA: ACM Press, May 2014, pp. 883–894.

[219] F. Thung, D. Lo, and L. Jiang, "Automatic defect categorization," in *Proceedings of the Working Conference on Reverse Engineering (WCRE)*, 2012, pp. 205–214.

[220] D. A. da Costa, S. L. Abebe, S. Mcintosh, U. Kulesza, and A. E. Hassan, “An Empirical Study of Delays in the Integration of Addressed Issues,” in *Proceedings of the International Conference on Software Maintenance and Evolution (ICSME)*, 2014, pp. 281–290.

[221] M. M. Rahman, G. Ruhe, and T. Zimmermann, “Optimized assignment of developers for fixing bugs an initial evaluation for eclipse projects,” in *Proceedings of the 3rd International Symposium on Empirical Software Engineering and Measurement*, IEEE, Oct. 2009, pp. 439–442.

[222] M. Zanoni, F. Perin, F. A. Fontana, and G. Viscusi, “Pattern detection for conceptual schema recovery in data-intensive systems,” *Journal of Software: Evolution and Process*, vol. 26, no. 12, pp. 1172–1192, 2014. arXiv: `1408.1293`.

[223] J. Neville and D. Jensen, “Collective Classification with Relational Dependency Networks,” in *Proceedings of the 2nd International Workshop on Multi-Relational Data Mining*, 2003, pp. 77–91.

[224] B. Taskar, V. Chatalbashev, and D. Koller, “Learning Associative Markov Networks,” in *Proceedings of the 21st International Conference on Machine Learning*, 2004, pp. 102–110.

[225] L. Getoor, “Link-based classification,” in *Advanced Methods for Knowledge Discovery from Complex Data*, vol. 3, 2005, pp. 189–207.

[226] E. Segal, R. Yelensky, and D. Koller, “Genome-wide discovery of transcriptional modules from DNA sequence and gene expression,” *Bioinformatics*, vol. 19, pp. 273–282, 2003.

[227] C. Science, R. Holloway, and L. Egham, “Support Vector Machines for Multi-Class Pattern Recognition,” in *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, vol. 99, 1999, pp. 219–224.

[228] D. Böhning, “Multinomial logistic regression algorithm,” *Annals of the Institute of Statistical Mathematics*, vol. 44, no. 1, pp. 197–200, 1992.

[229] S. a. Macskassy and F. Provost, “Classification in Networked Data: A toolkit and a univariate case study,” *Journal of Machine Learning Research*, vol. 8, no. December 2004, pp. 1–41, 2005.

[230] S. L. Lauritzen, *Graphical models*. Oxford University Press, 1996.

[231] S. a. Macskassy and F. Provost, “A simple relational classifier,” in *Proceedings of the 2nd Workshop on Multi-Relational Data Mining (MRDM)*, 2003, pp. 64–76.

[232] Z. Kou and W. W. Cohen, “Stacked Graphical Models for Efficient Inference in Markov Random Fields,” in *Proceedings of the SIAM International Conference*

on Data Mining. Society for Industrial and Applied Mathematics, 2007, pp. 533–538.

[233] S. a. Macskassy, “Relational classifiers in a non-relational world: Using homophily to create relations,” in *Proceedings of the 10th International Conference on Machine Learning and Applications (ICMLA)*, vol. 1, 2011, pp. 406–411.

[234] M. McPherson, L. Smith-Lovin, and J. M. Cook, “Birds of a Feather: Homophily in Social Networks,” *Annual Review of Sociology*, vol. 27, no. 1, pp. 415–444, 2001.

[235] P. Vojtek and M. Bieliková, “Homophily of neighborhood in graph relational classifier,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 5901 LNCS, pp. 721–730, 2010.

[236] B. Golub and M. O. Jackson, “How homophily affects the speed of learning and best-response dynamics,” *Quarterly Journal of Economics*, vol. 127, no. 3, pp. 1287–1338, 2012. arXiv: `1004.0858`.

[237] R. a. Hummel and S. W. Zucker, “On the foundations of relaxation labeling processes.,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 5, no. 3, pp. 267–287, 1983.

[238] A. Podgurski and L. A. Clarke, “A Formal Model of Program Dependeces and Its Implications for Software Testing, Debugging and Maintenance,” *IEEE Transactions on Software Engineering*, vol. Vol. 16, no. 9, pp. 965–979, 1990.

[239] B. Korel, “The Program Dependence Graph in Static Program Testing,” *Information Processing Letters*, vol. 24, no. 2, pp. 103–108, Jan. 1987.

[240] J. Ferrante, K. J. Ottenstein, and J. D. Warren, “The Program Dependence Graph and Its Use in Optimization,” *ACM Transactions on Programming Languages and Systems*, vol. 9, no. 3, pp. 319–349, Jul. 1987.

[241] A. Orso, S. Sinha, and M. J. Harrold, “Classifying Data Dependences in the Presence of Pointers for Program Comprehension, Testing, and Debugging,” *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 13, no. 2, pp. 199–239, Apr. 2004.

[242] L. Lopez-Fernandez, “Applying social network analysis to the information in CVS repositories,” in *Proceedings of the International Workshop on Mining Software Repositories (MSR)*, 2004, pp. 101–105.

[243] S.-K. Huang and K.-m. Liu, “Mining version histories to verify the learning process of Legitimate Peripheral Participants,” *ACM SIGSOFT Software Engineering Notes*, vol. 30, no. 4, p. 1, 2005.

[244] T. Wolf, A. Schröter, D. Damian, and T. Nguyen, “Predicting build failures using social network analysis on developer communication,” in *Proceedings of the 31st International Conference on Software Engineering (ICSE)*, 2009, pp. 1–11.

[245] A. Meneely, L. Williams, W. Snipes, and J. Osborne, “Predicting failures with developer networks and social network analysis,” in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE)*, 2008, p. 13.

[246] W. Hu and K. Wong, “Using citation influence to predict software defects,” in *Proceedings of the 10th Working Conference on Mining Software Repositories (MSR)*, IEEE, May 2013, pp. 419–428.

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

[248] M. Jorgensen and M. Shepperd, “A systematic review of software development cost estimation studies,” *IEEE Transactions on Software Engineering*, vol. 33, no. 1, pp. 33–53, 2007.

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

[250] S. Bibi, I. Stamelos, and L. Angelis, “Software cost prediction with predefined interval estimates,” in *Proceedings of the 1st Software Measurement European Forum*, Rome, Italy, 2004, pp. 237–246.

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

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

[253] R. Valerdi, “Convergence of Expert Opinion via the Wideband Delphi Method: An Application in Cost Estimation Models,” in *Proceedings of the 21st Annual International Symposium of the International Council on Systems Engineering (INCOSE)*, vol. 21, Jun. 2011, pp. 1238–1251.

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

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

[256] T. Pham, T. Tran, D. Phung, and S. Venkatesh, “Faster training of very deep networks via p-norm gates,” in *Proceedings of the 23rd International Conference on Pattern Recognition (ICPR)*, 2016, pp. 3542–3547.

[257] ——, "Predicting healthcare trajectories from medical records: A deep learning approach," *Journal of Biomedical Informatics*, vol. 69, pp. 218–229, 2017.

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

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

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

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

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

[263] S. Liang and R. Srikant, "Why Deep Neural Networks for Function Approximation?" In *arXiv:1610.04161*, 2016, pp. 1–13.

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

[265] M. Raghu, B. Poole, J. Kleinberg, S. Ganguli, and J. Sohl-Dickstein, "On the expressive power of deep neural networks," in *Proceedings of the 34th International Conference on Machine Learning (ICML)*, 2017, pp. 2847–2854.

[266] G. F. Montufar, R. Pascanu, K. Cho, and Y. Bengio, "On the number of linear regions of deep neural networks," in *Proceedings of the 27th International Conference on Neural Information Processing Systems*, 2014, pp. 2924–2932.

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

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

[269] J. Schmidhuber, "Deep Learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015. arXiv: 1404.7828.

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

[271] T. D. Team, “Theano: A {Python} framework for fast computation of mathematical expressions,” *arXiv e-prints*, vol. abs/1605.0, 2016.

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

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

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

[275] E. Kocaguneli, S. Member, and T. Menzies, “Exploiting the Essential Assumptions of Analogy-Based Effort Estimation,” *IEEE Transactions on Software Engineering*, vol. 38, no. 2, pp. 425–438, 2012.

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

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

[278] Y. F. Li, M. Xie, and T. N. Goh, “A Study of Project Selection and Feature Weighting for Analogy Based Software Cost Estimation,” *Journal of Systems and Software*, vol. 82, no. 2, pp. 241–252, Feb. 2009.

[279] S. Porru, A. Murgia, S. Demeyer, M. Marchesi, and R. Tonelli, “Estimating Story Points from Issue Reports,” in *Proceedings of the 12th International Conference on Predictive Models and Data Analytics in Software Engineering (PROMISE)*, 2016, pp. 1–10.

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

[281] H. H. Abdi, “The Bonferonni and Sidak Corrections for Multiple Comparisons,” *Encyclopedia of Measurement and Statistics.*, vol. 1, pp. 1–9, 2007. arXiv: `arXiv:1011.1669v3`.

[282] A. Vargha and H. D. Delaney, “A Critique and Improvement of the CL Common Language Effect Size Statistics of McGraw and Wong,” *Journal of Educational and Behavioral Statistics*, vol. 25, no. 2, pp. 101–132, 2000.

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

[284] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio, "Why does unsupervised pre-training help deep learning?" *Journal of Machine Learning Research*, vol. 11, pp. 625–660, Mar. 2010.

[285] J. Weston, F. Ratle, and R. Collobert, "Deep learning via semi-supervised embedding," in *Proceedings of the 25th International Conference on Machine Learning (ICML)*, Helsinki, Finland: ACM, 2008, pp. 1168–1175.

[286] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," *Journal of Machine Learning Research*, vol. 12, pp. 2493–2537, Nov. 2011.

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

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

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

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

[291] A. Karpathy, J. Johnson, and L. Fei-Fei, "Visualizing and Understanding Recurrent Networks," in *arXiv preprint arXiv:1506.02078*, 2015, pp. 1–12. arXiv: `1506.02078`.

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

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

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

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

[296] R. D. A. Christophe Commeyne, Alain Abran, *Effort Estimation with Story Points and COSMIC Function Points - An Industry Case Study*, 1. 2008, vol. 21, pp. 25–36.

[297] ISO/IEC JTC 1/SC 7, *INTERNATIONAL STANDARD ISO/IEC Software engineering COSMIC: a functional size measurement method*. 2011, vol. 2011, pp. 1–14.

[298] H. K. Dam, T. Tran, J. Grundy, and A. Ghose, “DeepSoft: A vision for a deep model of software,” in *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE)*, Seatle, WA, USA: ACM, To Appear., 2016.

[299] M. White, C. Vendome, M. Linares-v, and D. Poshyvanyk, “Toward Deep Learning Software Repositories,” in *Proceedings of the 12th Working Conference on Mining Software Repositories (MSR)*, 2015, pp. 334–345.

[300] M. White, M. Tufano, C. Vendome, and D. Poshyvanyk, “Deep learning code fragments for code clone detection,” in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Singapore, Singapore: ACM, 2016, pp. 87–98.

[301] V. Raychev, M. Vechev, and E. Yahav, “Code completion with statistical language models,” in *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2014, pp. 419–428.

[302] H. Dam, T. Tran, and T. Pham, “A deep language model for software code,” *arXiv preprint arXiv:1608.02715*, no. August, pp. 1–4, 2016. arXiv: `1608.02715`.

[303] *Deep learning*, http://deeplearning.net/.

[304] R. Gupta, S. Pal, A. Kanade, and S. Shevade, “Deepfix: Fixing common C language errors by deep learning,” in *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, AAAI Press, 2017, pp. 1345–1351.

[305] G. Hinton and R. Salakhutdinov, “Reducing the dimensionality of data with neural networks,” *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

Appendix A

The distribution of the Absolute Error achieved by Deep-SE

A.1 RQ1. Sanity Check

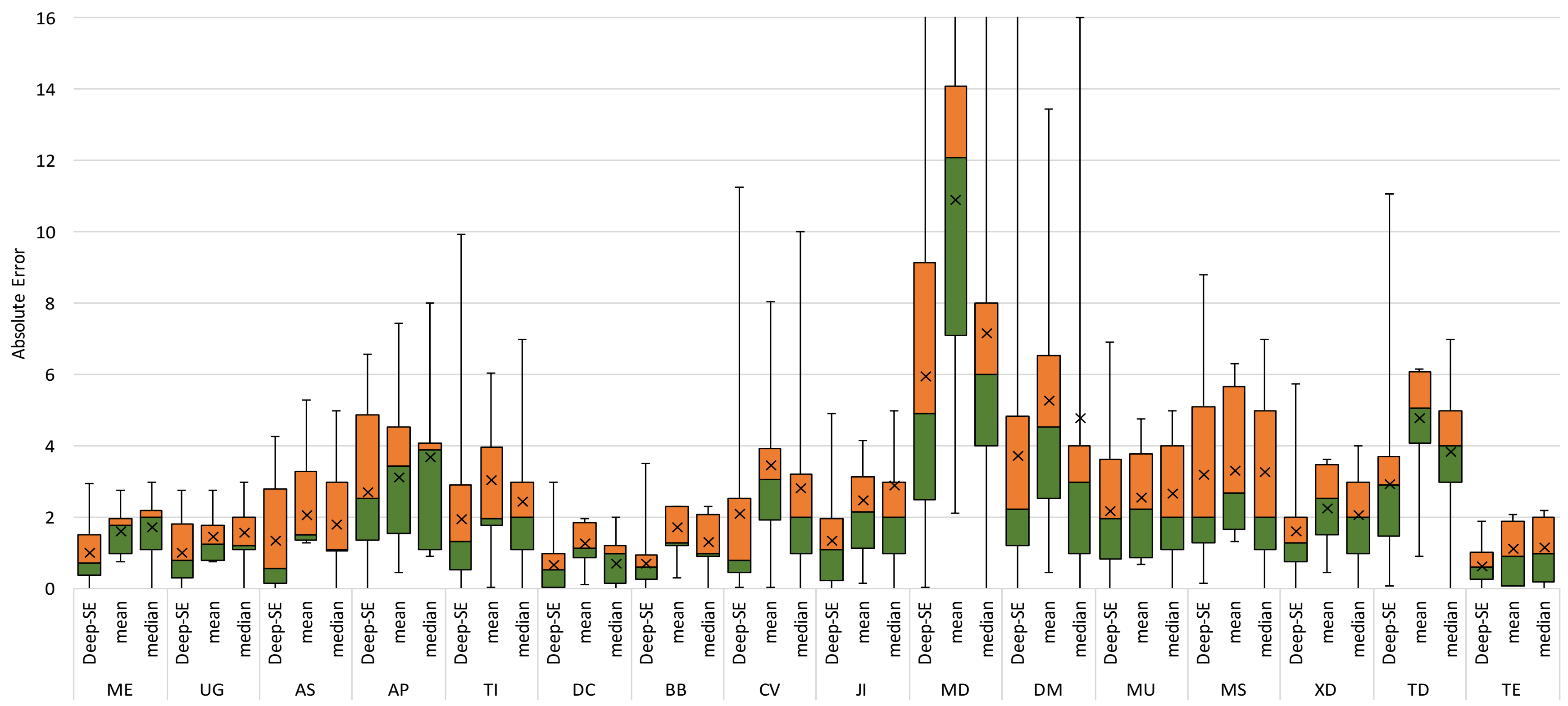


Figure A.1: The distribution of the Absolute Error achieved by Deep-SE, mean and median method in each project. X is a mean of the Absolute Error averaging across all issues in test set (the lower the better).

A.2 RQ2. Benefits of deep representation

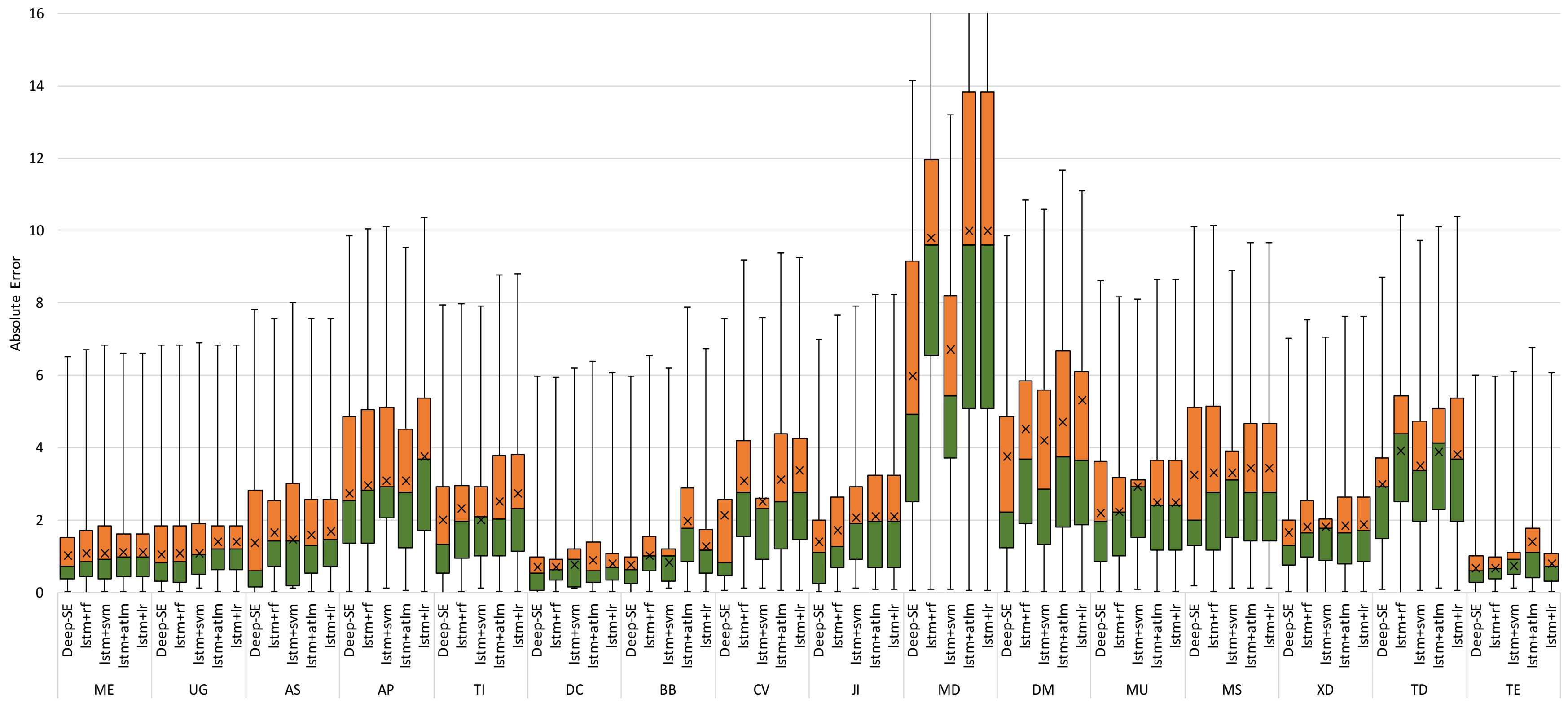


Figure A.2: The distribution of the Absolute Error achieved by Deep-SE, LSTM+RF, LSTM+SVM, LSTM+ATLM, and LSTM+LR in each project. X is a mean of the Absolute Error averaging across all issues in test set (the lower the better).

A.3 RQ3. Benefits of LSTM document representation

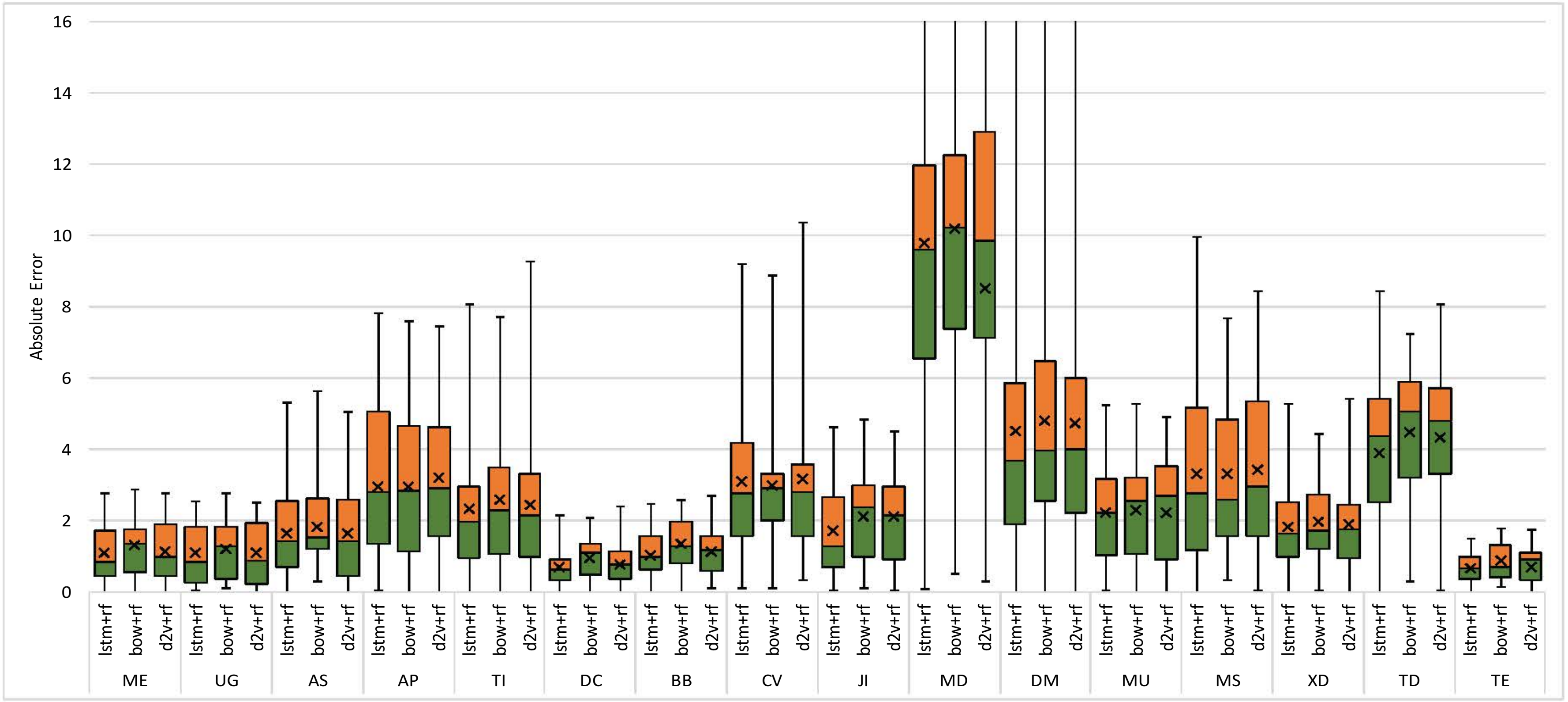


Figure A.3: The distribution of the Absolute Error achieved by LSTM+RF, BoW+RF, and Doc2vec+RF in each project. X is a mean of the Absolute Error averaging across all issues in test set (the lower the better).

A.4 RQ4. Cross-project estimation

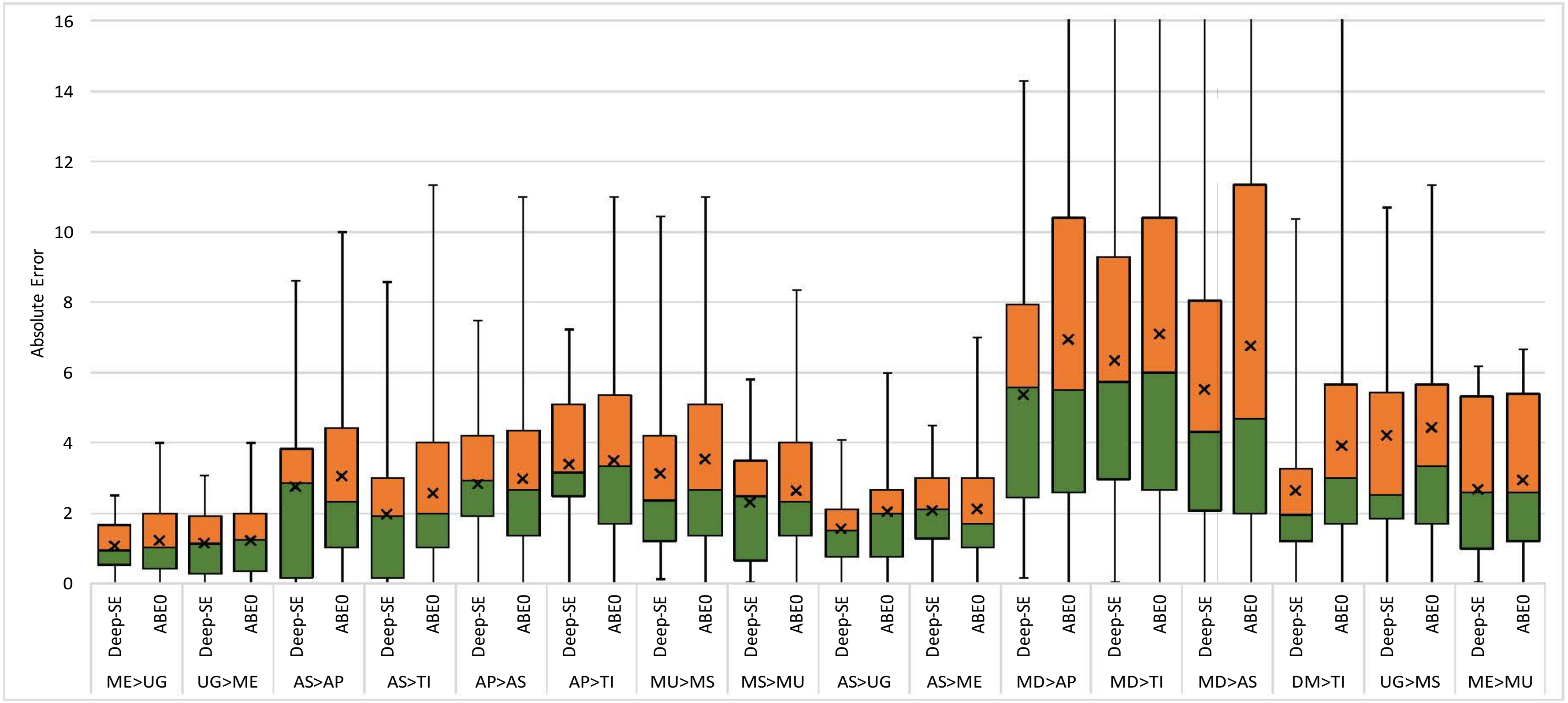


Figure A.4: The distribution of the Absolute Error achieved by Deep-SE and ABE0 in cross-project estimation. X is a mean of the Absolute Error averaging across all issues in test set (the lower the better).

A.5 RQ6. Compare Deep-SE against the existing approach

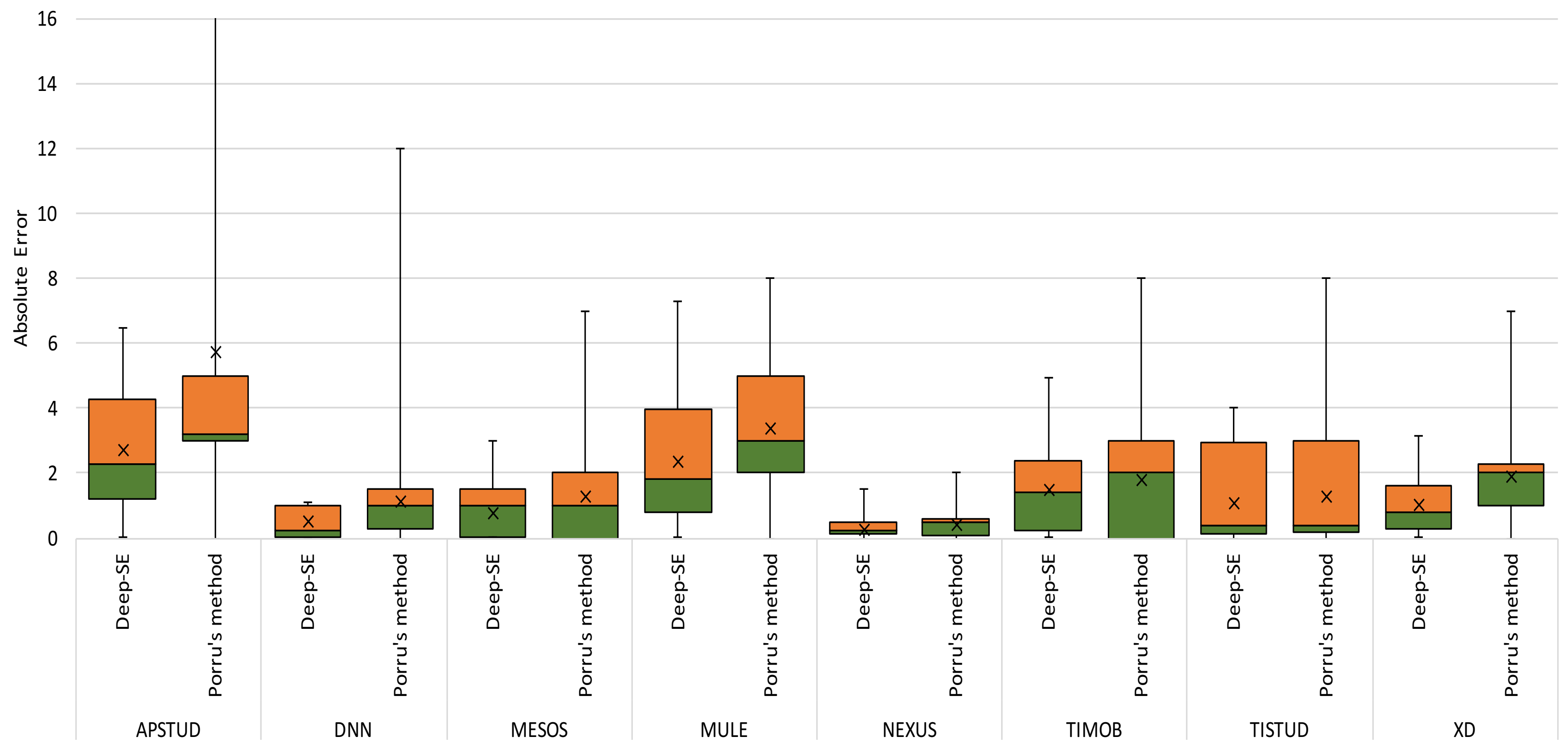


Figure A.5: The distribution of the Absolute Error achieved by Deep-SE and the Porru's approach on the Porru's dataset. X is a mean of the Absolute Error averaging across all issues in test set (the lower the better).