Empirical Software Engineering manuscript No.
(will be inserted by the editor)

Predicting the delay of issues with due dates in software projects

Morakot Choetkiertikul · Hoa Khanh Dam · Truyen Tran · Aditya Ghose

Received: date / Accepted: date

Abstract 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 paper 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 (hereafter called *risk factors*) characterizing delayed issues were extracted from eight open source projects: Apache, Duraspace, Java.net, JBoss, JIRA, Moodle, Mulesoft, and WSO2. Risk factors with good discriminative power were selected to build predictive models to predict if the resolution of an issue will be at risk of being delayed. Our predictive models are able to predict both the the extend of the delay and the likelihood of the delay occurrence. The evaluation results demonstrate the effectiveness of our predictive models, achieving on average 79% precision, 61% recall, 68% F-measure, and 83% Area Under the ROC Curve. Our predictive models also have low error rates: on average 0.66 for Macro-averaged Mean Cost-Error and 0.72 Macro-averaged Mean Absolute Error.

M. Choetkiertikul · H. K. Dam · A. Ghose
School of Computing and Information Technology,
Faculty of Engineering and Information Sciences,
University of Wollongong, Australia
E-mail: mc650@uowmail.edu.au

H. K. Dam
E-mail: hoa@uow.edu.au

A. Ghose
E-mail: aditya@uow.edu.au

Truyen Tran
School of Information Technology,
Deakin University, Australia
E-mail: truyen.tran@deakin.edu.au

Keywords Empirical software engineering · Project management · Mining software engineering repositories

1 Introduction

Software projects tend to overrun cost and schedule. In fact, a study (Michael et al 2012) 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. In the well-known CHAOS report, Standish Group found that 82% of software projects missed their schedules (Group 2004). To alleviate such problems, today's software development is trending toward agility and continuous delivery like DevOps. Even large software systems, e.g. Microsoft Windows (Bright 2015), 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 resolved issues.

An issue could represent a software bug, a new requirement or a user story, or even a project task. A range of recent work in software analytics have moved to focusing on issues, e.g. predicting its resolving time (e.g. Weiss et al (2007); Panjer (2007); Marks et al (2011); Giger et al (2010), which will be discussed in details in Section 7. 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. 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. However, in practice project will never execute exactly as it was planned due to various reasons. Hence, this work aims to provide automated support for project managers and other decision makers in predicting whether an issue 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.

In this paper, 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. 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 (hereafter called *risk factors*) characterizing delayed issues, which are then used to predict if an ongoing issue has a delay risk.

The paper presents two main contributions:

- **Characterization of the issues that constitute a risk of delay.** We extracted a comprehensive set of *19* risk factors (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 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.

This work extends our previous work (Choetkiertikul et al 2015a) in the following aspects:

- We have introduced a number of additional features (e.g. changing of fix versions). In addition, we have introduced a topic model for the description of issues using Latent Dirichlet Allocation (LDA) and used the extracted topics as an additional feature. This addition reflects the nature of an issue which may have implications to its delay status.
- A new feature selection technique using *p-value* has also been introduced, in addition to the ℓ_1-penalized logistic regression model. The former considers the influence of each risk factor to delay the outcome independently, whereas the latter examines all of them at the same time to take in account the possible correlation between them. We have also added an investigation to understand the predictive performance with and without using feature selection.
- We have employed two additional state-of-the-arts *randomized ensemble methods*: Stochastic Gradient Boosting Machines and Deep Neural Net-

works with Dropouts for building our predictive models. This allows us to observe how a wide range of classifiers perform in predicting delays for software projects.
- The evaluation has also been extended using a sliding window setting: first, the issue reports are sorted based on the time they are resolved; next, 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.
- Four additional projects, Java.net, JIRA, Mulesoft and WSO2, have been added to our dataset, increasing the number of issues to more than 60,000 across eight large open source projects.
- We have also added an additional risk class into our prediction models which now can classify an issue into: major delayed, medium delayed, minor delayed, and the non-delayed class.
- We have extended our empirical study with making delay predictions at three different points in time: at the creation time of an issue, at the time when a deadline (e.g. due date) was assigned to an issue, and the other at the end of discussion time, and compared their predictive performances. This investigation is to explore how the time when the prediction is made has implications to its accuracy and usefulness. Different prediction times lead to different set of features are extracted since some are only available at a certain time, e.g. developers are not assigned at the time an issue is created.
- The experimental evaluation has been redone entirely due to the above significant extensions, and new results are reported in this paper.

The remainder of this paper is organized as follows. Section 2 presents a comprehensive set of risk factors that are potentially associated with delayed issues. Section 3 describes how those risk factors are selected to develop our predictive models (discussed in Section 4). Section 5 reports the experimental evaluations of our approach. Threats to validity of our study are discussed in Section 6. Related work is discussed in Section 7 before we conclude and outline future work in Section 8.

2 Risk Factor Identification and Extraction

In this section, we describe how data were collected for our study and the risk factors extracted from the data.

2.1 Data collecting and preprocessing

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. Currently, there are more than fifty sub-projects under Apache community umbrella (i.e. Hadoop, Apache CouchDB, and Apache Spark). There are more than three hundred

core members and ten-thousand peripheral members contributing to the project (Iqbal 2014). All issues in Apache were recorded in Apache's issue tracking system[1]. The Duraspace project[2] supports the 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. Java.net[3] 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). JBoss[4] 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 with more than one-thousand contributors. JIRA[5] 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[6] 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[7] 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. 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 and their development progress are recorded in the WSO2's repository[8]. Those seven 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.

All the eight projects use JIRA[9], 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. Table 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 Duras-

[1] `https://issues.apache.org/jira`
[2] `https://jira.duraspace.org`
[3] `https://java.net/jira/`
[4] `https://issues.jboss.org`
[5] `https://jira.atlassian.com`
[6] `https://tracker.moodle.org`
[7] `https://www.mulesoft.org/jira/`
[8] `https://wso2.org/jira/`
[9] `https://www.atlassian.com/software/jira`

pace and WSO2, the resolved or closed issues between *November 9, 2006* to *February 03, 2016* were collected. For Moodle, more than 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 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

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 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/?selectedTab=com.atlassian.jira.jira-projects-plugin:version-summary-panel 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.

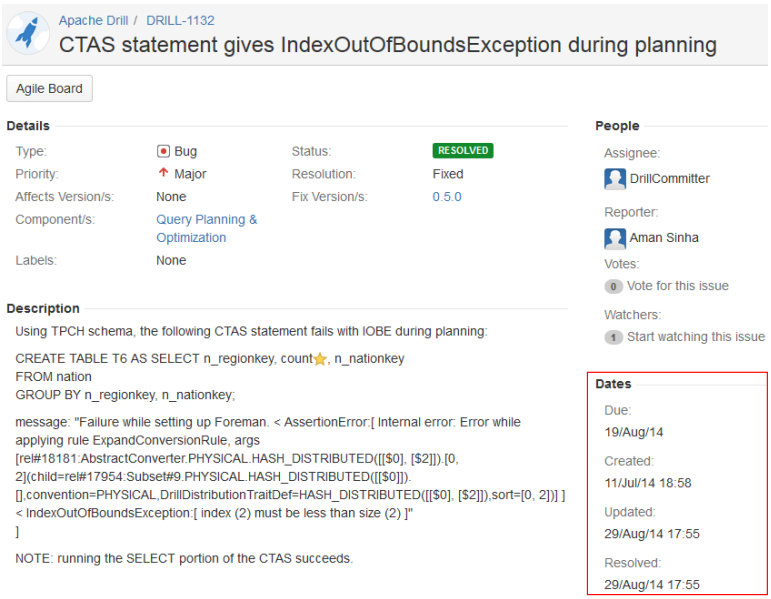

Fig. 1: An example of an issue assigned with a due date

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 (Hodge and Austin 2004) in identifying those outlier cases in each project, we

Table 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

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 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. We have made all the data publicly available at `http://www.dsl.uow.edu.au/sasite/index.php/public-datasets-2/`.

2.2 Potential risk factors for software issues

One of our objectives is to characterize risk factors that lead to a delayed issue. These factors form risk indicators (or features) which are then used to predict if an issue will be delayed. In our model, 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 and testing 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 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 risk factors extracted and selected are historically relevant with respect to this prediction time. Doing this is to prevent information leakage in prediction (Kaufman and Perlich 2012). 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 risk factors related to an issue causing a delay, and then used feature selection techniques (see Section 3) to remove weak and redundant factors. We now describe each of those risk factors in detail.

– 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 2 shows the distribution of the discussion time in each project. Teams tend to spend between 10 to 90 days for the discussion time.

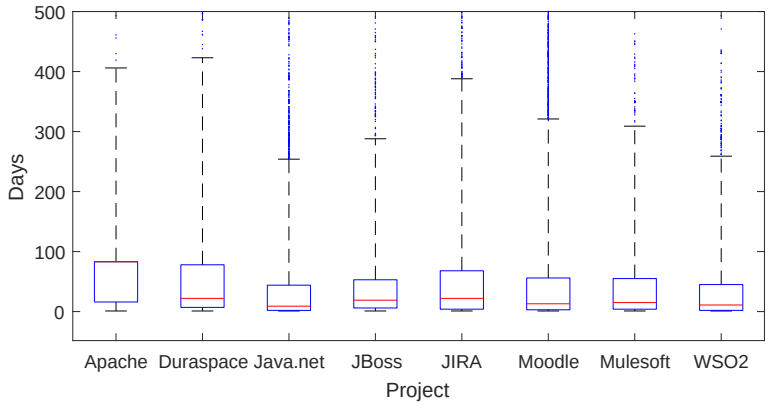

Fig. 2: The distribution of the *discussion time* in each project

– Type

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., Zimmermann et al (2012), Guo et al (2010), Shihab et al (2012), Xia et al (2014a), 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 (Xia et al 2014b). In addition, Valdivia et al. use the changing of priority as a feature to predict blocking bug (Valdivia Garcia et al 2014). 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 (Bettenburg et al 2008a). 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 (Panjer 2007). 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 (Xia et al 2014b). 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 (Choetkiertikul et al 2015b). 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 (Xia et al 2015). 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 (Valdivia Garcia et al 2014).

– 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) (Blei et al 2012). 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 5.

– 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., Bhattacharya and Neamtiu (2011); Hooimeijer and Weimer (2007); Zimmermann et al (2012). For example, Zimmermann et. al. found that bugs reported by low reputation people are less likely to be reopened (Zimmermann et al 2012). Hooimeijer et al. used bug opener's reputation to predict whether a new bug report will receive immediate attention or not (Hooimeijer and Weimer 2007). Bhattacharya et al. studied bug fix time prediction models using submitter's reputations (Bhattacharya and Neamtiu 2011). 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 (Guo et al 2010). We use Hooimeijer's submitter reputation (Hooimeijer and Weimer 2007) 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 3 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.

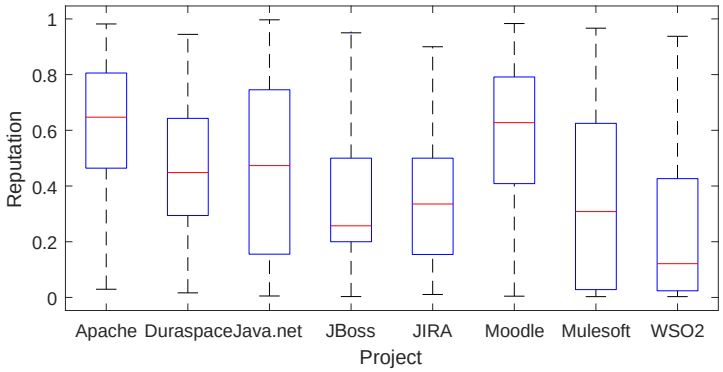

Fig. 3: The distribution of the *reporter reputation* in each project

– 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 (Han and Huang 2007), and developer workload may have significant impact on the progress of a project (Porter et al 1997; Pika et al 2013). 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 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 (Wallace and Keil 2004). Incompetent developers tend to not complete their

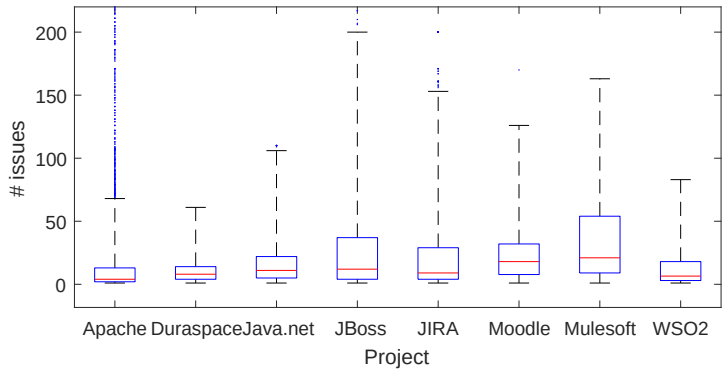

Fig. 4: The distribution of the *developer's workload* in each project

tasks in time (Pika et al 2013). Boehm (Boehm 1991) 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 they often choose or were given the most complex tasks (Kim et al 2006). 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

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

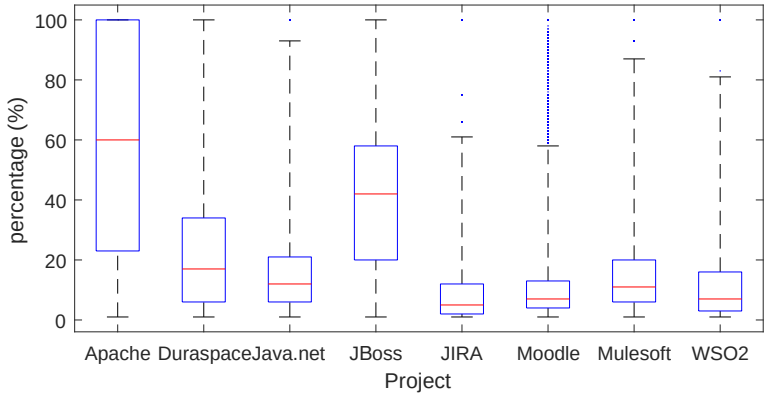

Fig. 5: The distribution of the *percentage of delayed issues that a developer involved with* in each project

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.

3 Risk Factor Selection

Sparse models – those with a small number of features – are desirable since they are easier to interpret and acted upon by model users. They are also critical to prevent over-fitting when training data is limited compared to the number of possible attributes. Hence, the second phase of our approach involves selecting a compact subset of risk factors (described in Section 2) that provide a good predictive performance. This process is known as *feature selection* in machine learning (Guyon and Elisseeff 2003).

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

3.1 Using p-value from logistic regression model

Logistic regression model (Menard 2002) 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 risk factor x as follows:

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

We built logistic regression models based on the risk factors described in Section 2. This technique examines the contribution of each risk factor to the classification outcome independently. The coefficient β_1 indicates how strong the risk factor contributes to the delay. Its p-value measures the probability that the risk factor's contribution is due to random chance. A low p-value ($p < 0.05$) indicates that a risk factor is likely to associated with the outcome. A larger p-value, in contrast, suggests that the changes of a factor's value are not statistically associated with the changes of the delayed class. In our study, we select risk factors with $p < 0.05$. However, assessing p-values of an individual risk factor ignores correlations among them which can cause multi-collinearity. Collinearity refers to an exact or approximate linear relationship between two or more risk factors 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 risk factors which exhibit multi-collinearity using the Belsley collinearity diagnostics (Belsley et al 2005) by selecting only one risk factor with the lowest p-value from those highly correlated risk factors.

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 risk factor in absence of other factors. When all factors are simultaneously considered, the individual contributions tend to be dampened due to correlation between factors. Thus it might be better to assess the significance of all factors at the same time. To that end, we developed a ℓ_1-penalized logistic regression model (Lee et al 2006) for selecting risk factors. The following is the detail of the ℓ_1-penalized logistic regression model.

Let $\left\{(x^i, y^i)\right\}_{i=1}^n$ be the training set, where $x \in \mathbb{R}^p$ be the factor 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 factor 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 factors by shrinking their weights toward zeros. It does so by creating competition among the factors 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 risk factors were then used to build a predictive model which we describe in the next section.

The source code in Matlab of both feature selection techniques: using p-value from logistic regression model and ℓ_1-penalized logistic regression model have been published online at `http://www.dsl.uow.edu.au/sasite/index.php/public-datasets-2/`.

4 Predictive Models

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 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 that we selected are used to train a diverse set of seven classifiers: Random Forests, Neural Networks, Decision Tree, Naive Bayes, NBTree, Deep Neural Networks with Dropouts, and Gradient Boosting Machines. We briefly describe each classifier as follows.

- Neural Networks (aNN) - aNN 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 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., Hu et al (2007) ,Neumann (2002) ,Wang et al (2005), which yields good results.
- Deep Neural Networks with Dropouts (Deep Nets) - Deep Nets is traditional artificial neural networks with multiple hidden layers. Recent advances include dropout (Srivastava et al 2014), 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.
- Decision Tree (C4.5) - C4.5 is a decision tree algorithm that generates decision nodes based on the information gain using the value of each factors (Quinlan 1993). The significant benefit of decision tree classifier is that it offers an explainable model. Thus, most of the work that focuses on interpreting models, e.g.,Conforti et al (2015) ,Ibrahim et al (2010), selected this technique.
- Random Forests (RF) - RF is a significant improvements of the decision tree approach by generating many classification trees, each of which is

built on a random resampling of the data, with random subset of variables at each node split. Tree predictions are then aggregated through voting (Breiman 2001). Previous research, Lessmann et al (2008) ,Kamei et al (2010) has shown that RF is one of the best for a wide range of problems.
- Gradient Boosting Machines (GBMs) - GBMs are a boosting method that combines multiple weak learners in an additive manner (Friedman 2001, 2002). 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 RF, a weak learner can be any function approximator. In our implementation, regression trees are used as weak learners. Each tree is trained on 80% randomly sampled data, and 30% of risk factors. This creates a stochastic ensemble which is known to improve generalizability.
- Naive Bayes (NB) - NB is based on the assumption that risk factors, when the outcome is known, are conditionally independent. Despite of this naive assumption, NB has been found to be effective as a classifier. NB offers a natural way to estimate the probability of delay. Thus, studies that aim to measure the degree of uncertainty naturally apply NB to build models, e.g., Wolfson et al (2014) ,Abdelmoez et al (2012).
- NBTree - NBtree is a hybrid algorithm between Naive Bayes classifier and C4.5 Decision Tree classification. The decision tree nodes contain univariate splits as regular decision trees, but the leaf nodes accommodate with Naive Bayes classifiers (Kohavi 1996).

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.

4.1 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 (Zadrozny and Elkan 2002). 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.

4.1.1 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.

4.1.2 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 (Chawla and Cieslak 2006).

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.

4.1.3 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 risk factors in Section 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 (Garg and Roth 2001).

4.1.4 Combining decision tree and Naive Bayes

As mentioned earlier, NBTree is a hybrid of decision tree (C4.5) and Naive Bayes. 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. 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 (Wang et al 2006).

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 (Boehm 1989). Risk exposure supports project managers to establish risk priorities (Boehm 1991). 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.

5 Evaluation

5.1 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 2.1) 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 (Weiss et al 2007). Figure 11 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. This would allow us to investigate several different training and test sets.

Table 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.)

Table 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".

5.2 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 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 5 shows the indication of collinearity in terms of Variance-Decomposition Proportions (VDP) from applying Belsley collinearity diagnostics (Belsley et al 2005). 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 $VDP > 0.5$ 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 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

Table 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

an issue causing delays (Jr and Lemeshow 2004). 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

Table 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

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 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

5.3 Performance Measure

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 num-

ber of predictive performance measures suitable for ordinal risk classes for evaluation described as below.

5.3.1 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). The values stored in the confusion matrix are used to compute the widely-used Precision, Recall, and F-measure for the delayed issues to evaluate the performance of the predictive models:

- Precision (Prec): The ratio of correctly predicted delayed issue over all the issues predicted as delayed issue. It is calculated as:

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

- Recall (Re): The ratio of correctly predicted delayed issue over all of the actually issue delay. 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.

5.3.2 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 $\delta\left[.\right]$ is the indicator function.

The Macro-averaged Mean Cost-Error[10] is defined as below:

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

where $\bar{RE}$ is the predicted risk exposure computed in Section 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 4.2 for how a risk exposure is calculated). Hence, the MMCE error between actual and predicted risk exposure for this issue is 1.1.

5.3.3 Macro-averaged Mean Absolute Error (MMAE)

We also used another metric called Macro-averaged Mean Absolute Error (MMAE) (Baccianella et al 2009) to assess the distance between actual and predicted classes. MMAE is suitable for ordered classes like those defined in this paper. 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 acknowledge that other strategies can be used in practice. Again the normalization against the class size handles the class imbalance.

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

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.

5.4 Results

5.4.1 Comparison of different projects

Figure 6 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 demonstrate the effectiveness of our predictive models across the eight projects, achieving 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).

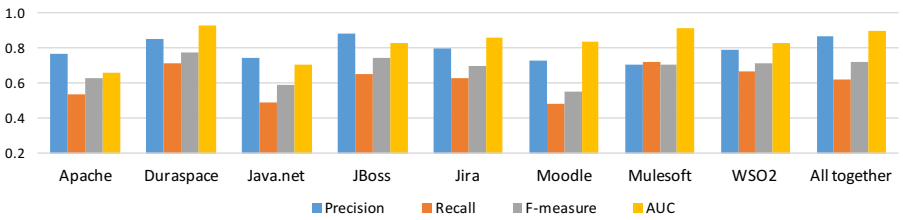

Fig. 6: Evaluation results for different projects

5.4.2 Comparison of different classifiers

Table 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

Table 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 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.

5.4.3 Comparison of different feature selection approaches

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 7, the ℓ_1-penalized logistic regression model produced the best performance across different measures: 0.79 precision, 0.63 recall, 0.69 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 features (i.e. *ℓ_1-penalized logistic regression model*) effectively contributes to the predictive performance.

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 7, 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.

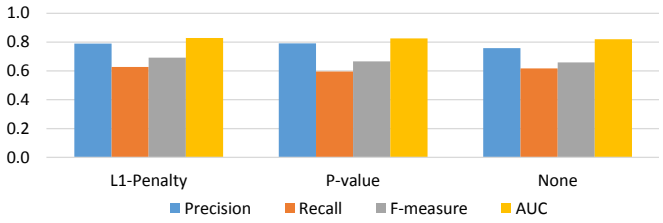

Fig. 7: Evaluation results for different feature selection approaches

5.4.4 Comparison of different prediction times

The above results were obtained from the predictions made at the end of discussion time (see Section 2). We also performed 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. As can be seen in Figure 8, 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. This result confirms our hypothesis that the time when the prediction is made has implications to its accuracy – the later we predict, the more accuracy we could gain since more information has become available. However, predicting at later times may be less useful since the outcome may become obvious or it is too late to change the outcome.

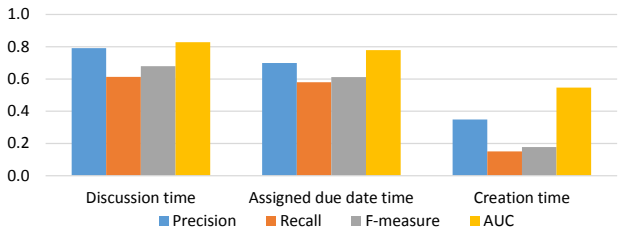

Fig. 8: Evaluation results for different prediction times

5.4.5 Comparison of different numbers of topics

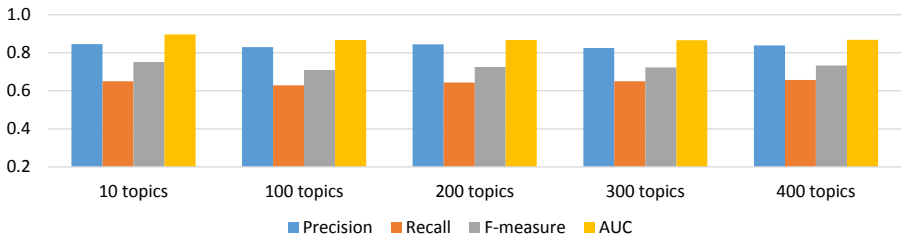

Fig. 9: Evaluation results for different numbers of topics

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 9 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 9, 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 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.

Table 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

5.4.6 MMAE and MMCE as performance measures

MMAE and MMCE are used to assess the performance of our models in terms of predicting risk exposure. Figure 10 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.

5.4.7 Sliding window approach

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 to predict delayed issues in the next window.

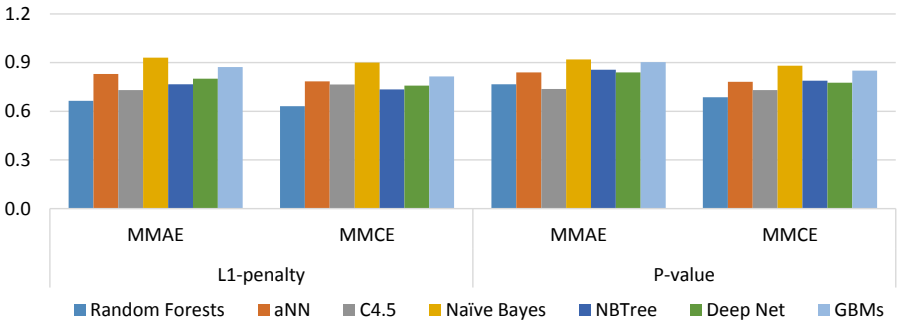

Fig. 10: MMAE and MMCE (the lower the better)

Figure 11 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" (Lam et al 2008) 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.

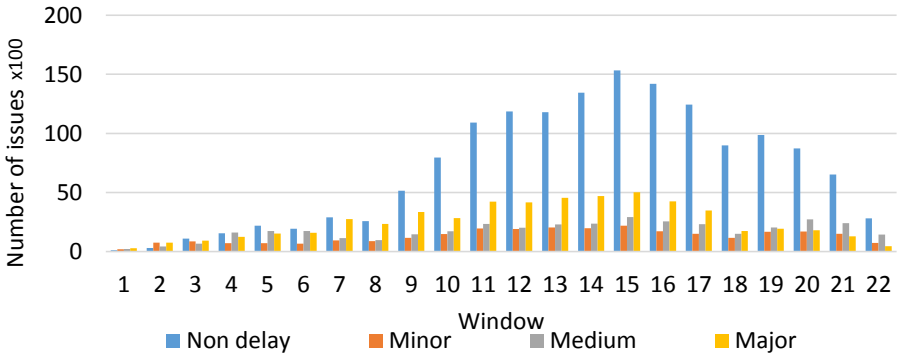

Fig. 11: Number of issues in each window (6-month window)

Figure 12 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 6^{th} window achieves 0.66 precision, 0.79 recall, 0.72 F-measure, and

0.83 AUC. As can be observed from Figure 12, The predictive performance in terms of F-measure then decreases after the 8^{th} 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 8^{th} window. The predictive performance measures (i.e. precision, recall, and F-measure) then converges at the 16^{th} window and slightly increases after the 18^{th} window.

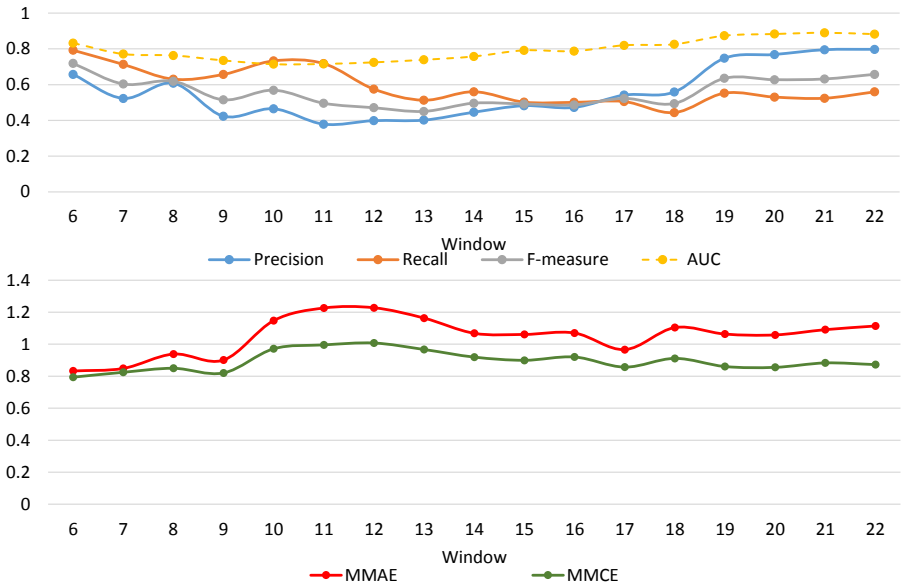

Fig. 12: Evaluation results using the 6-month sliding window setting

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 13 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. It could shows that the predictive model trained from the *recent* issues can increase the predictive performance (e.g. prevent the overfitting problem)

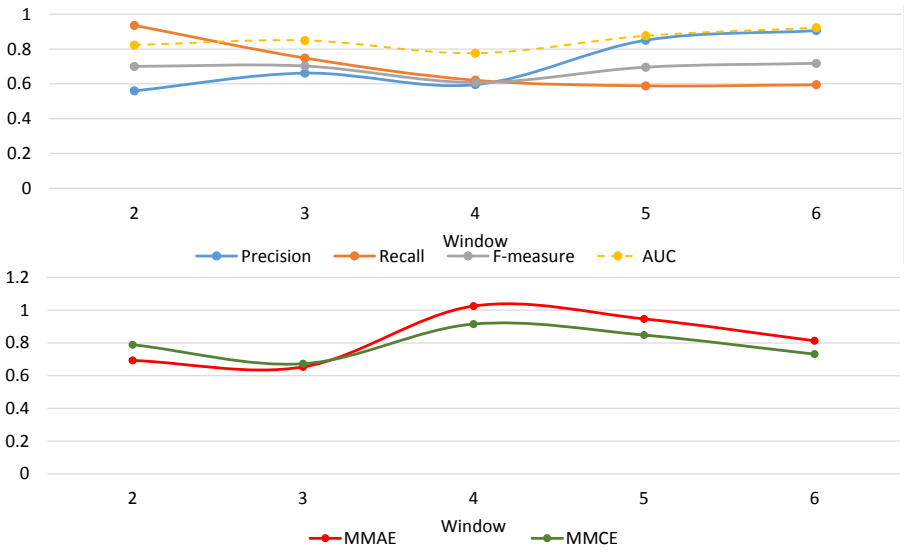

Fig. 13: Evaluation results using the 2-year sliding window setting

5.4.8 What did we learn from the risk factors

Outcomes from the risk factor selection process (see Section 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, 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 Mule-

Table 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

soft 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 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 (Genuer et al 2010). 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.

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 se-

lection 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 5.1. 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 paper 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 (Qin et al 2014). 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.

7 Related Work

Software risk management has attracted great attention since Boehm's seminal work, e.g., Boehm (1989, 1991), 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 (Boehm 1991; Xu Ruzhi et al 2003). Risk control aims to develop, engage, and monitor risk mitigation plans (Boehm 1991).

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 (Letier et al 2014). 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 (Pika et al 2013). For example, the work in Hu et al (2013) 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., Weiss et al (2007), Giger et al (2010), Panjer (2007), Marks et al (2011), Bhattacharya and Neamtiu (2011), blocking bug prediction, e.g., Valdivia Garcia et al (2014), Xia et al (2015), re-opened bug prediction, e.g., Zimmermann et al (2012), Shihab et al (2012), Xia et al (2014a), severity/priority prediction, e.g., Lamkanfi et al (2010), Menzies and Marcus (2008), Tian et al (2015), automatic bug categorization, e.g., Thung et al (2012), delays in the integration of a resolved issue to a release, e.g., da Costa et al (2014), bug triaging, e.g., Anvik et al (2006), Anvik and Murphy (2011), Rahman et al (2009), Murphy and Čubranić (2004), bug report field reassignment, e.g., Xia et al (2014b), Kochhar et al (2014), bug resolver prediction, e.g., Zanoni et al (2014) and duplicate bug detection (Runeson et al 2007), Wang et al (2008), Bettenburg et al (2008b), Sun et al (2011), Jalbert and Weimer (2008). 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 Weiss et al (2007) 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 Panjer (2007) 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 Marks et al (2011) proposed to use Random Forrest to predict bug's fixing time using three features: location, reporter and description. The work in Giger et al (2010) 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 Marks et al (2011). 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.

8 Conclusions and Future Work

In this paper, 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 future work would involve expanding our study to other large open source projects and commercial software projects to further assess our predictive models. We also plan to explore additional risk factors such as the discussions between developers and project leaders. Since there are interdependencies between risk in practice (e.g. one risk can trigger another), an important part of our future work is to develop a model (e.g. using Bayesian network) to represent dependencies and probabilistic relationships between causes and effects of the risk factors we identified here. Risk assessment which has been addressed in this paper is just only the first part of the solution. The next task is providing various actionable recommendations such as which risks should be deal with first, and which measures could be used to mitigate them.

References

Abdelmoez W, Kholief M, Elsalmy FM (2012) Bug Fix-Time Prediction Model Using Naïve Bayes Classifier. In: Proceedings of the 22nd International Conference on Computer Theory and Applications (ICCTA), October, pp 13–15

Anvik J, Murphy GC (2011) Reducing the effort of bug report triage. ACM Transactions on Software Engineering and Methodology 20(3):1–35

Anvik J, Hiew L, Murphy GC (2006) Who should fix this bug? In: Proceedings of the 28th International Conference on Software engineering (ICSE), ACM Press, New York, USA, pp 361–370

Baccianella S, Esuli A, Sebastiani F (2009) Evaluation measures for ordinal regression. In: Proceedings of the 9th International Conference on Intelligent Systems Design and Applications (ISDA), IEEE, pp 283–287

Belsley DA, Kuh E, Welsch RE (2005) Regression diagnostics: Identifying influential data and sources of collinearity, vol 571. John Wiley & Sons

Bettenburg N, Just S, Schröter A, Weiss C, Premraj R, Zimmermann T (2008a) What makes a good bug report? In: Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering, ACM Press, New York, USA, pp 308–318

Bettenburg N, Premraj R, Zimmermann T (2008b) Duplicate bug reports considered harmful ... really? In: Proceedings of the International Conference on Software Maintenance (ICSM), pp 337–345

Bhattacharya P, Neamtiu I (2011) Bug-fix time prediction models: can we do better? In: Proceedings of the 8th working conference on Mining software repositories (MSR), ACM, pp 207–210

Blei DM, Ng AY, Jordan MI (2012) Latent Dirichlet Allocation. Journal of Machine Learning Research 3(4-5):993–1022

Boehm B (1989) Software risk management. Springer

Boehm B (1991) Software risk management: principles and practices. Software, IEEE 8(1):32–41

Breiman L (2001) Random forests. Machine learning pp 5–32

Bright P (2015) What windows as a service and a free upgrade mean at home and at work. https://goo.gl/Fzwflg

Chawla N, Cieslak D (2006) Evaluating probability estimates from decision trees. American Association for Artificial Intelligence (AAAI) pp 1–6

Choetkiertikul M, Dam HK, Tran T, Ghose A (2015a) Characterization and prediction of issue-related risks in software projects. In: Proceedings of the 12th Working Conference on Mining Software Repositories (MSR), IEEE, pp 280–291

Choetkiertikul M, Dam HK, Tran T, Ghose A (2015b) Predicting delays in software projects using networked classification. In: Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp 353 – 364

Conforti R, de Leoni M, La Rosa M, van der Aalst WM, ter Hofstede AH (2015) A recommendation system for predicting risks across multiple business process instances. Decision Support Systems 69:1–19

da Costa DA, Abebe SL, Mcintosh S, Kulesza U, Hassan AE (2014) An Empirical Study of Delays in the Integration of Addressed Issues. In: Proceedings of the International Conference on Software Maintenance and Evolution (ICSME), pp 281–290

Friedman JH (2001) Greedy function approximation: a gradient boosting machine. Annals of Statistics 29(5):1189–1232

Friedman JH (2002) Stochastic gradient boosting. Computational Statistics & Data Analysis 38(4):367–378

Garg A, Roth D (2001) Understanding Probabilistic Classifiers, Lecture Notes in Computer Science, vol 2167. Springer Berlin Heidelberg, Berlin, Heidelberg

Genuer R, Poggi JM, Tuleau-Malot C (2010) Variable selection using random forests. Pattern Recognition Letters 31(14):2225–2236

Giger E, Pinzger M, Gall H (2010) Predicting the fix time of bugs. In: Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering (RSSE), ACM, pp 52–56

Group S (2004) Chaos report. Tech. rep., West Yarmouth, Massachusetts: Standish Group

Guo PJ, Zimmermann T, Nagappan N, Murphy B (2010) Characterizing and predicting which bugs get fixed: an empirical study of Microsoft Windows. Proceedings of the 32nd International Conference on Software Engineering (ICSE) 1:495–504

Guyon I, Elisseeff A (2003) An Introduction to Variable and Feature Selection. The Journal of Machine Learning Research 3:1157–1182

Han WM, Huang SJ (2007) An empirical analysis of risk components and performance on software projects. Journal of Systems and Software 80(1):42–50

Hodge VJ, Austin J (2004) A Survey of Outlier Detection Methodoligies. Artificial Intelligence Review 22(1969):85–126

Hooimeijer P, Weimer W (2007) Modeling bug report quality. In: Proceedings of the 22 IEEE/ACM international conference on Automated software engineering (ASE), ACM Press, pp 34 – 44

Hu Y, Huang J, Chen J, Liu M, Xie K, Yat-sen S (2007) Software Project Risk Management Modeling with Neural Network and Support Vector Machine Approaches. In: Proceed-

ings of the 3rd International Conference on Natural Computation (ICNC), vol 3, pp 358–362

Hu Y, Zhang X, Ngai E, Cai R, Liu M (2013) Software project risk analysis using Bayesian networks with causality constraints. Decision Support Systems 56:439–449

Ibrahim WM, Bettenburg N, Shihab E, Adams B, Hassan AE (2010) Should I contribute to this discussion? Proceedings of the 7th IEEE Working Conference on Mining Software Repositories (MSR 2010) pp 181–190

Iqbal A (2014) Understanding Contributor to Developer Turnover Patterns in OSS Projects : A Case Study of Apache Projects. ISRN Software Engineering 2014:10 – 20

Jalbert N, Weimer W (2008) Automated duplicate detection for bug tracking systems. In: Proceedings of the International Conference on Dependable Systems and Networks With FTCS and DCC (DSN), IEEE, pp 52–61

Jr DH, Lemeshow S (2004) Applied logistic regression, 3rd edition. Wiley

Kamei Y, Matsumoto S, Monden A, Matsumoto Ki, Adams B, Hassan AE (2010) Revisiting common bug prediction findings using effort-aware models. In: Proceedings of the IEEE International Conference on Software Maintenance (ICSM), IEEE, pp 1–10

Kaufman S, Perlich C (2012) Leakage in Data Mining : Formulation , Detection , and Avoidance. ACM Transactions on Knowledge Discovery from Data (TKDD) 6(4)(15):556–563

Kim S, Zimmermann T, Pan K, Jr Whitehead E (2006) Automatic Identification of Bug-Introducing Changes. In: Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering (ASE), IEEE, pp 81–90

Kochhar PS, Thung F, Lo D (2014) Automatic fine-grained issue report reclassification. Proceedings of the IEEE International Conference on Engineering of Complex Computer Systems, ICECCS pp 126–135

Kohavi R (1996) 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), pp 202 – 207

Lam X, Vu T, Le T (2008) Addressing cold-start problem in recommendation systems. Proceedings of the 2nd international conference on Ubiquitous information management and communication pp 208–211

Lamkanfi A, Demeyer S, Giger E, Goethals B (2010) Predicting the severity of a reported bug. In: Proceedings of the 7th IEEE Working Conference on Mining Software Repositories (MSR), IEEE, pp 1–10

Lee SI, Lee H, Abbeel P, Ng AY (2006) Efficient l˜ 1 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, pp 401–409

Lessmann S, Baesens B, Mues C, Pietsch S (2008) Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings. IEEE Transactions on Software Engineering 34(4):485–496

Letier E, Stefan D, Barr ET (2014) Uncertainty, risk, and information value in software requirements and architecture. In: Proceedings of the 36th International Conference on Software Engineering (ICSE), ACM Press, New York, USA, pp 883–894

Marks L, Zou Y, Hassan AE (2011) 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, pp 1–8

Menard S (2002) Applied logistic regression analysis, 2nd edition, vol 106. SAGE University paper

Menzies T, Marcus A (2008) Automated severity assessment of software defect reports. In: Proceedings of the International Conference on Software Maintenance (ICSM), IEEE, pp 346–355

Michael B, Blumberg S, Laartz J (2012) Delivering large-scale IT projects on time, on budget, and on value. Tech. rep.

Murphy G, Čubranić D (2004) Automatic bug triage using text categorization. In: Proceedings of the 16th International Conference on Software Engineering & Knowledge Engineering (SEKE), pp 92–97

Neumann D (2002) An enhanced neural network technique for software risk analysis. IEEE Transactions on Software Engineering 28(9):904–912

Panjer LD (2007) Predicting Eclipse Bug Lifetimes. In: Proceedings of the 4th International Workshop on Mining Software Repositories (MSR), pp 29–32

Pika A, van der Aalst WM, Fidge CJ, ter Hofstede AH, Wynn MT, Aalst WVD (2013) Profiling event logs to configure risk indicators for process delays. In: Proceedings of the 25th International Conference on Advanced Information Systems Engineering (CAiSE), Springer, pp 465–481

Porter AA, Siy HP, Votta LG (1997) Understanding the effects of developer activities on inspection interval. In: Proceedings of the 19th international conference on Software engineering (ICSE), ACM Press, pp 128–138

Qin X, Salter-Townshend M, Cunningham P (2014) Exploring the Relationship between Membership Turnover and Productivity in Online Communities. In: Proceedings of the 8th International AAAI Conference on Weblogs and Social Media

Quinlan JR (1993) C4.5: programs for machine learning. Morgan Kaufmann Publishers Inc.

Rahman MM, Ruhe G, Zimmermann T (2009) 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, pp 439–442

Runeson P, Alexandersson M, Nyholm O (2007) Detection of Duplicate Defect Reports Using Natural Language Processing. In: Proceedings of the 29th International Conference on Software Engineering (ICSE), IEEE, pp 499–510

Shihab E, Ihara A, Kamei Y, Ibrahim WM, Ohira M, Adams B, Hassan AE, Matsumoto Ki (2012) Studying re-opened bugs in open source software. Empirical Software Engineering 18(5):1005–1042

Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R (2014) Dropout: A simple way to prevent neural networks from overfitting. Journal of Machine Learning Research 15:1929–1958

Sun C, Lo D, Khoo SC, Jiang J (2011) Towards more accurate retrieval of duplicate bug reports. In: Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering (ASE), IEEE, pp 253–262

Thung F, Lo D, Jiang L (2012) Automatic defect categorization. Proceedings of the Working Conference on Reverse Engineering (WCRE) pp 205–214

Tian Y, Lo D, Xia X, Sun C (2015) Automated prediction of bug report priority using multi-factor analysis. Empirical Software Engineering 20(5):1354–1383

Valdivia Garcia H, Shihab E, Garcia HV (2014) Characterizing and predicting blocking bugs in open source projects. In: Proceedings of the 11th Working Conference on Mining Software Repositories (MSR), ACM Press, pp 72–81

Wallace L, Keil M (2004) Software project risks and their effect on outcomes. Communications of the ACM 47(4):68–73

Wang LM, Li XL, Cao CH, Yuan SM (2006) Combining decision tree and Naive Bayes for classification. Knowledge-Based Systems 19(7):511–515

Wang Q, Zhu J, Yu B (2005) Combining Classifiers in Software Quality Prediction : A Neural Network Approach. In: Proceedings of the 2nd International Symposium on Neural Networks, Springer Berlin Heidelberg, pp 921–926

Wang X, Zhang L, Xie T, Anvik J, Sun J (2008) An approach to detecting duplicate bug reports using natural language and execution information. In: Proceedings of the 30th International Conference on Software Engineering (ICSE), pp 461–470

Weiss C, Premraj R, Zimmermann T, Zeller A (2007) How Long Will It Take to Fix This Bug? In: Proceedings of the 4th International Workshop on Mining Software Repositories (MSR), pp 1–8

Wolfson J, Bandyopadhyay S, Elidrisi M, Vazquez-Benitez G, Musgrove D, Adomavicius G, Johnson P, O'Connor P (2014) A Naive Bayes machine learning approach to risk prediction using censored, time-to-event data. Statistics in medicine pp 21 – 42

Xia X, Lo D, Shihab E, Wang X, Zhou B (2014a) Automatic, high accuracy prediction of reopened bugs. Automated Software Engineering 22(1):75–109

Xia X, Lo D, Wen M, Shihab E, Zhou B (2014b) An empirical study of bug report field reassignment. Proceedings of the Conference on Software Maintenance, Reengineering, and Reverse Engineering pp 174–183

Xia X, Lo D, Shihab E, Wang X, Yang X (2015) ELBlocker: Predicting blocking bugs with ensemble imbalance learning. Information and Software Technology 61:93–106

Xu Ruzhi, leqiu Q, Jing Xinhai (2003) CMM-based software risk control optimization. In: Proceedings of the 5th IEEE Workshop on Mobile Computing Systems and Applications, IEEE, pp 499–503

Zadrozny B, Elkan C (2002) Transforming classifier scores into accurate multiclass probability estimates. In: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, pp 694–699

Zanoni M, Perin F, Fontana FA, Viscusi G (2014) Dual analysis for recommending developers to resolve bugs. Journal of Software: Evolution and Process 26(12):1172–1192

Zimmermann T, Nagappan N, Guo PJ, Murphy B (2012) Characterizing and predicting which bugs get reopened. In: Proceedings of the 34th International Conference on Software Engineering (ICSE), IEEE Press, pp 1074–1083