

Characterization and prediction of issue-related risks in software projects

Morakot Choetkietikul*, Hoa Khanh Dam*, Truyen Tran[†] and Aditya Ghose*

*School of Computer Science and Software Engineering

Faculty of Engineering and Information Sciences

University of Wollongong, Australia

Email: {mc650,hoa,aditya}@uow.edu.au

[†]School of Information Technology

Deakin University, Australia

Email: truyen.tran@deakin.edu.au

Abstract—Identifying risks relevant to a software project and planning measures to deal with them are critical to the success of the project. Current practices in risk assessment mostly rely on high-level, generic guidance or the subjective judgements of experts. In this paper, we propose a novel approach to risk assessment using historical data associated with a software project. Specifically, our approach identifies patterns of past events that caused project delays, and uses this knowledge to identify risks in the current state of the project. A set of risk factors characterizing “risky” software tasks (in the form of *issues*) were extracted from five open source projects: Apache, Duraspace, JBoss, Moodle, and Spring. In addition, we performed feature selection using a sparse logistic regression model to select risk factors with good discriminative power. Based on these risk factors, we built predictive models to predict if an issue will cause a project delay. Our predictive models are able to predict both the risk impact (i.e. the extend of the delay) and the likelihood of a risk occurring. The evaluation results demonstrate the effectiveness of our predictive models, achieving on average 48%–81% precision, 23%–90% recall, 29%–71% F-measure, and 70%–92% Area Under the ROC Curve. Our predictive models also have low error rates: 0.39–0.75 for Macro-averaged Mean Cost-Error and 0.7–1.2 for Macro-averaged Mean Absolute Error.

I. INTRODUCTION

Software projects tend to overrun cost and schedule. In fact, a study [1] 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 [2]. One explanation for such overruns is that project managers do not take prudent measures to assess and manage the risks. Research has shown that risk management, which involves the identification, assessment, and treatment of risks, is critical to the success of IT projects [3].

Current practices in software risk management mostly rely on high-level guidance (e.g. Boehm’s “top 10 list of software risk items” [4] or SEI’s risk management framework [5]) and expert knowledge. While such guidance and frameworks provide useful information, they are generic and making them applicable to a particular project is typically not easy. On

the other hand, expert judgements provide assessment of risks specific to a project, but they tend to be subjective.

Existing empirical work (e.g. [6–10]) has mostly considered how risk management has supported decision makers and project managers in identifying, analyzing and mitigating risks. Such studies are useful in developing a retrospective understanding of the relationships between risk factors and how they affect project outcomes. However, it would be much more valuable for project managers and decision makers to be provided with insightful and actionable information about the current existence of risks in their project at the fine-grained level of tasks (*issues*¹), allowing them to come up with concrete measures to with these issues.

The work presented in this paper aims to address these (latter) questions. We focus on a particular kind of risk that adversely affects a scheduled release date – an important, major milestone for a software project. Each release has a planned date and an actual date, and is associated with a number of issues that need to be resolved prior to release. Releases are typically delayed because of their associated issues not being resolved in time (i.e. before the release date). Those issues that cause release delays are referred to in this paper as *delayed issues* – as opposed to *non-delayed issues* which are 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 poses the risk of causing a *delay*. Our approach mines the historical data associated with a project to extract past instances of risks (i.e. delayed issues). An example of a risk pattern is a software developer being overloaded with the issues assigned to them, resulting in the risk that they may not complete some of those tasks in time for a scheduled release. This knowledge allows us to extract a set of features (*risk factors*) characterizing “risky” issues, which are then used

¹In modern software development (for both commercial and open software development contexts), software tasks are often recorded in an issue/bug tracking system (e.g. JIRA) in the form of new feature requests, bug reports, documentation tasks, etc. Hereafter, we refer to all software tasks as *issues* to reflect this practice.

to predict if an existing issue has a delay risk. Hence, our approach addresses both risk identification and risk analysis, which are arguably the most significant and challenging tasks in the risk management process. Risk identification involves determining risk that can, when present, adversely affect a project. Risk analysis aims to identify the *impact of a risk* and its *probability of occurrence*.

The paper presents two main contributions:

- **Characterization of the issues that constitute a risk of delay.**

We extracted a comprehensive set of 16 risk factors (discussion time, waiting time, issues type, number of repetition tasks, percentage of delayed issues that a developer involved with, developer’s workload, issue priority, number of comments, changing of priority, number of fix version, number of affect version, number of issue link, number of blocked issues, number of blocking issues, changing of description, and reporter reputation) from the 40,830 issues collected from five open source projects: Apache, Duraspace, JBoss, Moodle, and Spring. In addition, we performed feature selection using a *sparse 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 poses a risk of delaying a project milestone (e.g. a release). 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.) 30% chance of X not posing a risk (e.g. causing no delay), (e.g.) 50% of being a minor risk (e.g. causing minor delay), and (e.g.) 20% a major risk (e.g. major delay). The performance of our predictive models were evaluated on five different open source projects to ensure that they can be generalized. We achieved 48%–81% precision, 23%–90% recall, 29%–71% F-measure, 70%–92% Area Under the ROC Curve, and low error rates: 0.39–0.75 for Macro-averaged Mean Cost-Error and 0.7–1.2 for Macro-averaged Mean Absolute Error.

The remainder of this paper is organized as follows. Section II presents a comprehensive set of risk factors that are potentially associated with schedule overruns. Section III describes how those risk factors are selected to develop our predictive models (discussed in Section IV) . Section V reports the experimental evaluations of our approach. Threats to validity of our study are discussed in Section VI. Related work is discussed in Section VII before we conclude and outline future work in Section VIII.

II. RISK FACTOR IDENTIFICATION AND EXTRACTION

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

A. Data collecting and preprocessing

We collected the data (past issues) from five open source projects: Apache, Duraspace, JBoss, Moodle, and Spring. Apache is a web server originally designed for Unix environments. There are more than three hundred core members and ten-thousand peripheral members contributing to the project [11]. All issues in Apache were recorded in Apache’s issue tracking system². The Duraspace project³ supports the digital asset management that contains several sub-projects in their repository i.e. VIVO, Islandora, Hydra Hypatia, and Dura-Cloud. There are about two-hundred contributors including reporters, developers, testers, and reviewers working for the Duraspace community. JBoss⁴ is an application server program which supports the general enterprise software development framework. The JBoss community has been developing more two-hundred sub-projects with more than one-thousand contributors. 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 tracker⁵ is the Moodle’s issue tracking system which is used to collect issues, working items and keep track issues status in development activities. Spring⁶ is an application development framework and it has approximately one-hundred contributors. Those five projects have different sizes, number of contributors, and development processes.

All the five projects use JIRA⁷, 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. We collected both *delayed* and *non-delayed* issues. For delayed issues, we also collected how many days of delay the issues caused. For Duraspace, JBoss, Moodle and Spring, delayed and non-delayed issues were identified by comparing their due date and resolving date, e.g. delayed issues were resolved after their due date. For Moodle, the Moodle’s development life-cycle has a two-release plan: a major release every 6 months, and a minor release every 2 months. The release versions were planned based on these two milestones. We identified delayed issues by comparing a planned release date with actual resolved date of issues.

TABLE I: Dataset description

Project	Delayed issues		Non-delayed issues		Total
	Count	Percentage	Count	Percentage	
Apache	111	[2.27%]	4,771	[97.72%]	4,882
Duraspace	314	[9.75%]	2,908	[90.25%]	3,222
JBoss	2,242	[15.08%]	12,627	[84.92%]	14,869
Moodle	214	[2.24%]	9,325	[97.75%]	9,539
Spring	80	[0.96%]	8,238	[99.03%]	8,318
Total	2,961	[7.25%]	37,869	[92.74%]	40,830

²<https://issues.apache.org/jira/secure/Dashboard.jspa>

³<https://jira.duraspace.org/>

⁴<https://issues.jboss.org/secure/Dashboard.jspa>

⁵<https://tracker.moodle.org/secure/Dashboard.jspa>

⁶<https://spring.io/projects>

⁷<https://www.atlassian.com/software/jira>

Table I shows the number of delayed issues and non-delayed issues for each project. For Apache, 14,364 issues opened from *January 1, 2012* to *December 16, 2014* were collected from their issue tracking system. During the data preprocessing phase, we removed issues with a status other than resolved or closed. We also filtered out issues with empty field. Some issues reported as a test case in QA tasks, which are not related to a development task, were also filtered out. After the preprocessing step, the number of issues was reduced to 4,882 issues consisting of 111 delayed issues (2.27%) and 4,771 non-delayed issues (97.7%). For Duraspace, the resolved or closed issues between *February 9, 2007* to *August 18, 2014* were collected. For JBoss, Spring and Moodle, we retrieved issues between *January 4, 2004* to *January 27, 2015*. The same preprocessing used for the Apache was applied for the other four projects. In total, we collected 40,830 issues from all the five projects, which consist of 2,961 (7.25%) delayed and 37,869 (92.7%) non-delayed issues. We have made all the data publicly available at <http://www.uow.edu.au/~mc650/>.

B. Potential risk factors for software issues

One of our objectives is to characterize what risk factors lead to an issue causing a project delay. These factors form risk indicators (or features) which are then used to predict if an issue will cause a delay. The time when the prediction is made has implications to its accuracy and usefulness. The latter 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). In this work, we assume prediction would be made after an issue has been discussed and assigned to a developer, and thus the risk factors we extracted are historically relevant with respect to this prediction time. We however acknowledge that further studies are needed to determine when it is a good time to start predicting.

We initially extracted a broad range of risk factors related to an issue causing a delay, and then used feature selection techniques (see Section III) to remove weak and redundant factors. We now describe each of those risk factors in detail.

1) *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.

2) *Waiting time*: Waiting time indicates the time when an issue is waiting for being acted upon, e.g. waiting for the assigned developer(s) to take actions. Abnormal waiting time is an indication of an issue being delayed due to lack of team cooperation or nobody wanting to deal with the issue [12]. The waiting time of an issue is from when an assignee has been

assigned to the issue, until they take an action to resolve it such as changing the issue's status or submitting a pull request.

3) *Type*: Each issue will be assigned a type (e.g. Task, Bug, New feature, Improvement, and Documentation) 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 the issue types are defined specifically in each of the five selected projects. For example, while there is no "Document" type for issues in the Moodle project, this type exists in the issues of JBoss and Duraspace.

4) *Number of times that an issue is reopened*: Previous research (e.g. [13–15]) 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.

5) *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.

6) *Changing of priority*: The changing of an issue's priority may indicate shift of its complexity. For example, Valdivia et al. [16] use the changing of priority as a feature to predict blocking bug. 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.

7) *Number of comments*: Number of comments from developers during the discussion time may be indicative of the degree of teams collaboration [17]. Panjer [18] 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. Note that we only consider the number of comments posted before the end of the discussion time.

8) *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.

9) *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 planned 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.

10) *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 issues. There are several type of issue links (e.g. relates to, duplicate, and block). We consider all relations of issue link and use the number of those links as a risk indicator.

11) *Number of issues that are blocked by this issue*: Blocker is one of the issue linking relation 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. Thus, we deal with blocker relationship separately.

12) *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 high number of blocker issues is more difficult since all blocker issues need to be fixed beforehand. Thus, the number of blocker issues indicates the time allocated to solve an issue [16].

13) *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.

14) *Reporter reputation*: Reporter reputation has been studied in previous work in mining bug reports (e.g. [13, 19, 20]). For example, Zimmermann et. al. [13] found that bugs reported by low reputation people are less likely to be reopened. Hooimeijer et al. [20] used bug opener’s reputation to predict whether a new bug report will receive immediate attention or not. Bhattacharya et al. [19] studied bug fix time prediction models using submitter’s reputations. In the context of predicting delayed issues, reporter reputation could be one of the risk factors because issue reporters with low reputation may write poor issue reports, which may result in a longer time to resolve the issue [14]. We use the reporter reputation as defined based on Hooimeijer’s submitter reputation [20] 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.

15) *Developer’s workload*: Developer workload is a reflection of the quality of resource planning, which is crucial for project success. A lack of resource planning has implications to project failures [21], and developer workload may have significant impact on the progress of a project [12, 22]. 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.

16) *Percentage of delayed issues that a developer involved with*: Team members lack specialized skills required by the project and inexperienced team members are amongst the

major threats to schedule overruns [23]. Teams that consist of incompetent developers might be a cause of project delays [12]. Boehm [4] also stated that personnel shortfalls are 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 [24]. This phenomenon might also hold for delayed issues: best developers may get most/hardest issues and thus they take the longest time to complete it. A developer might have a large number of delayed issues because he/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.

III. 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 II) that provide a good predictive performance. This process is known as *feature selection* in machine learning [25].

We developed a ℓ_1 -penalized logistic regression model [26] for selecting risk factors. A logistic regression model aims to predict the probability of an event occurring (e.g. will this issue cause a delay?) using a combination of factors that can be numerical (e.g., the number of comments), or categorical (e.g. issue type). The ℓ_1 -penalized logistic regression model was built on training data. Our factors were chosen from those described in Section II. We built a model for the issues from each project and a model for all of the issues we collected across the five projects. We now describe this ℓ_1 -penalized logistic regression model in more detail.

Let $\{(x^i, y^i)\}_{i=1}^n$ 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 $y = -1$ if not. 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:

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

where $\lambda > 0$ is the penalty factor. The ℓ_1 -penalty suppresses weak, redundant and irrelevant factors by shrinking their

TABLE II: Descriptive ℓ_1 -penalized logistic regression model for risk probability, trained on all issues collected from the five projects

Feature	Apache	Duraspace	JBoss	Moodle	Spring	All
discussion	1.027	-1.368	-3.153	-21.353	-0.371	-7.274
waiting	0	0	0	-15.307	0	-4.495
type:Bug	0	-3.254	-0.959	-2.933	-4.763	-0.859
type:Code Task	N/A	-0.052	N/A	N/A	N/A	0.216
type:Documentation	0	0.359	1.617	N/A	N/A	1.06
type:Epic	0	1.026	0	0	0	0.925
type:Improvement	0.044	-0.745	N/A	-2.833	-4.704	0.271
type:New Feature	0.106	0.097	0	-3.011	-4.495	0.64
type:Story	0	-0.81	0	0	0	-0.342
type:Sub-task	-1.183	0.267	0.94	-2.796	0.001	0.867
type:Task	-0.37	-0.477	0.724	-2.604	-4.534	0.548
type:Technical task	0	0.35	0	0	0	0.567
repetition	-0.391	4.01	3.11	-0.427	6.358	2.878
perofdelay	3.183	4.202	3.088	2.569	22.091	3.361
workload	0.179	-0.911	1.367	-1.104	0.241	1.621

Feature	Apache	Duraspace	JBoss	Moodle	Spring	All
priority:Blocker	0.465	0.657	-0.322	-1.744	4.301	-0.827
priority:Critical	-1.585	0	-0.45	-2.929	3.716	-0.903
priority:Major	0.317	0	-0.453	-2.983	1.571	-0.921
priority:Minor	0	-0.663	-0.438	-2.801	1.778	-0.876
priority:Trivial	-0.229	-1.548	-0.95	-2.639	0	-0.783
no_comment	0	0.001	2.31	-1.43	-2.918	2.826
no_priority_change	-0.001	2.874	-0.234	1.684	1.325	1.166
no_fixversion_change	3.042	0.152	2.081	-1.352	-0.088	2.529
no_affectversion	-0.001	0	-0.903	-11.489	-0.822	-3.413
no_issuelink	0.001	0	0	-1.717	0.883	1.268
no_blocking	0	0	-0.175	0.52	0	0.216
no_blockedby	0	-0.236	0.747	2.341	0	1.785
no_des_change	0.001	0.955	0	1.251	0	1.472
reporter_rep	1.355	-0.298	-0.712	2.53	1.623	-0.498

discussion=Discussion time, wating=Wating time, type=Issue’s type, repetition=Number of repetition tasks, perofdelay=Percentage of delayed issues that a developer involved with, workload=Developer’s workload, priority=Issue’s priority, no comment=Number of comments, no priority change=Changing of priority, no fixversion=Number of fix version, no affectversion=Number of affect version, no issuelink=Number of issue link, no blocking=Number of issues that are blocked by this issue, no blockedby=Number of issues that block this issue, no des change=Changing of description, reporter rep=Reporter reputation, N/A=This feature is not available.

weights toward zeros. It does so by creating competition among factors to explain 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].

Table II 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 five projects). 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 Apache the weight of the discussion time is positive (1.027), indicating that the discussion time is positively correlated with delayed issues. By contrast, the weight of the discussion time in the other four projects is negative (e.g. -3.153 in JBoss), meaning the discussion time being negatively correlated with delayed issues. This diversity can also be observed in other risk factors (except the “percentage of delayed issues that a developer involved with” factor which is positively correlated with delayed issues in all the five projects). We also note that the magnitude of each weight approximately indicates the degree to which a factor affects the probability of an issue causing delays [27]. Note that the exponential of a weight is an *odds-ratio* which is a measure of 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 relative strength of 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, improvement type, number of repetition tasks and reporter reputation are among the risk factors we selected

for the Apache project. Note that we selected different sets of risk factors for different projects due to the project diversity. The selected factors were then used to build a predictive model which we describe in the next section.

IV. PREDICTIVE MODELS

Our predictive models are able to predict not only if an issue will cause a delay 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 I), we choose to use two risk classes: *major delayed* and *minor delayed* (and the *non-delayed* class). During the training phase for our dataset, we use a threshold to determine which risk class an issue belongs to. The threshold was chosen such that it gives a good balance between the two risk classes.

For each of our case study projects, the risk factors that we selected are used to train five classifiers: Random Forests, Neural Networks, Decision Tree, Naive Bayes, and NBTree. We briefly describe each classifier as follows.

- Random Forests (RF) - RF is a significant improvements of decision tree approach by generating many classification trees, each of which is built with random subset of variables at each node split, and aggregates into the individual results with a higher concentration of a particular class [28]. Previous research [29, 30] has shown that RF is an outstanding modeling technique.
- Neural Networks (aNN) - aNN is a model of feedforward recognition mimicking biological neurons and synapses. Its 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.,[31–33]) which yield good results.
- 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 [34]. 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.,[35,36]) selected this technique.

- 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. The main reason for using NB is to obtain the probability that a issue will be delayed as we need to determine a likelihood of the risk. Thus, studies that aim to measure the degree of uncertainty naturally applies NB to build models (e.g., [37,38]).
- 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 [39].

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 and major delay. In the following subsection, we will describe this important aspect of our predictive models.

A. Predicting the likelihood of a risk occurring

Our objective is not only predicting risk classes but also estimating the class probabilities. Of the five classifiers studied, Naive Bayes and Neural Networks 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. 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 [40]. 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.

1) *Estimating probability in Random Forests:* A Random Forests algorithm will generate many trees from random sampling of features and data instances. The predicted class is the class that hold the highest score in voting (the number of trees predicting that class) [28]. 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 30 and 10 trees predict minor delayed and non-delayed respectively. Thus, the voting result can be treat as the probability distribution, which means, the probability of the issue to be major delayed is 60%, 30% for minor delayed, and 10% for non-delayed.

2) *Neural Network classifiers with posteriori probabilities:* In order to convert Neural Networks' output to probabilities, aNN classifiers typically provide outputs which estimate Bayesian posteriori probabilities. When the estimation is accurate, aNN outputs can be treated as probabilities and the sum of probability distribution is equal to 1 [41,42]. It means that the weighted outputs that are generated from the networks can be treated as a class probability. For example, in our context, when we construct the Neural Networks, it must consist of three output nodes in the output layer to represent three predicting classes (*major delayed*, *minor delayed*, and *non-delayed*). An output node calculates the sum of weighted features that is calculated from the hidden layer. Thus, the results from the thee output nodes are the probability distribution of each class.

3) *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 [43].

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) *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, minor, and non delayed risk) and a dependent feature vector x_1 through x_n which are our risk factors in Section II, the probability of class y is:

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

Then, the instances are classified using Bayes's decision rule [44].

5) *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 [45].

B. Risk exposure prediction

Our predictive models are able to predict the exposure of a risk. Risk exposure is defined as the probability of a risk occurring times the loss if that risk occurs [46]. Risk exposure supports project managers to establish risk priorities [4]. Our predicted risk exposure \bar{RE} is computed as follows.

For an issue i , let C_1 , C_2 and C_3 be the costs associated with the issue causing no delay, minor 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:

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

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

Note that all the costs, C_1 , C_2 and C_3 , are user defined and specific to a project. For example, assume that $C_1 = 0$, $C_2 = 1$, and $C_3 = 2$, 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. $P(i, Min) = 0.4$), and 30% major delay (i.e. $P(i, Maj) = 0.3$), then the predicted risk exposure \bar{RE}_i of the issue is 1.

V. EVALUATION

This section describes our experimental setting, the performance measures and the results.

A. Experimental setting

All issues collected in each of the five case studies (see Section II-A) 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 setting was to try mimic a real deployment scenario that prediction on a current issue is made using knowledge from the past issues. We could also follow 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. This would allow us to investigate several different training and test sets per project, which we leave for future work since we believe that our findings in the current experimental setting still hold.

TABLE III: Experimental setting

Project	Training set			Testset		
	Major	Minor	Non	Major	Minor	Non
Apache	15	66	340	2	28	105
Duraspace	39	118	2,805	15	55	500
JBoss	998	795	8,965	112	337	1,794
Moodle	80	89	791	18	27	195
Spring	11	39	350	12	18	120
All together	1,143	1,107	13,251	159	465	2,714

(“All together” is an integration of issues from all the five projects.)

Table III shows the number of issues in training set and test set for each project. Since (major/minor) delayed issues are rare (only 7% 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.

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

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:

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

where C_i is the binary classification of issue i , and $P(i, Maj)$, $P(i, Min)$, and $P(i, Non)$ are the probabilities of issue i classified in the major delayed, minor delayed, and non-delayed classes respectively. Basically, this rule determines that an issue is considered as delayed if the sum probability of it being classified into the major and minor delayed classes are 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 truly caused a delay, then the classification is a true positive (tp). If the issue is classified as delayed when actually it did not cause a delay, then the classification is a false positive (fp). If the issue is classified as non-delayed when it in fact caused a delay, then the classification is a false negative (fn). Finally, if the issue is classified as non-delayed and it in fact did not cause a delay, 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**: 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**: 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.

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 $k \in \{1, 2, 3\}$ – there are 3 classes in our classification – i.e., $n_k = \sum_{i=1}^n \delta [y^i = k]$ and $n = n_1 + n_2 + n_3$. Here $\delta [\cdot]$ is the indicator function.

The Macro-averaged Mean Cost-Error⁸ is defined as below:

$$MMCE = \frac{1}{3} \sum_{k=1}^3 \frac{1}{n_k} \sum_{i=1}^n |\bar{RE}_i - C| \delta [y^i = k]$$

where \bar{RE} is the predicted risk exposure computed in Section IV-B 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, 30% in minor delayed, and 20% in non-delayed. Assume that the issue actually caused a minor delay (i.e. the true class is minor delayed) and the costs C_1 (no delay), C_2 (minor delay), and C_3 (major delay) are respectively 0, 1, and 2. The predicted risk exposure \bar{RE} is 1.3 and the actual risk exposure is 1 (see Section IV-B for how a risk exposure is calculated). Hence, the MMCE error between actual and predicted risk exposure for this issue is 0.3.

3) *Macro-averaged Mean Absolute Error (MMAE)*: We also used another metric called Macro-averaged Mean Absolute Error (MMAE) [47] 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 ($k = 3$), then an error of 2 has occurred. Here, we assume that the predicted class is the one with the highest probability, but we acknowledge that other strategies can be used in practice. Again the normalization against the class size handles the class imbalance.

⁸Here we deal with only 3 classes but the formula can be easily generalized to n classes.

Macro-averaged Mean Absolute Error:

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

For example, an issue is predicted to be 10% in major delayed, 60% in minor delayed, and 30% in non-delayed. Thus, the predicted class of this issue is minor delayed ($k = 2$). 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 1.

C. Results

Comparison of different classifiers: Figure 1 shows the precision, recall, F-measure, MMCE, and MMAE achieved by Random Forests, Neural Network, Decision Tree (C4.5), Naive Bayes, and NBTree for the collected issues in each of the five open source projects and in all the projects (shown by “All” legend). As can be seen in Figure 1(a.), Random Forests achieve the highest precision of 0.9 (averaging across five projects), while the other classifiers achieve only 0.4–0.6 precision. Random Forests also outperform the other classifiers in terms of recall and F-measure: it achieves the highest recall of 0.7 and the highest F-measure of 0.8 (averaging across five projects). It should however be noted that Naive Bayes and Decision outperform Random Forests in terms of F-measure for the Spring project. Random Forests also achieve the lowest risk exposure prediction errors with only 0.34 for MMAE and 0.41 for MMCE, while the other classifiers give 1.05–1.15 MMAE and 0.64–0.7 MMCE.

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. As can be seen in Figure 1(d.), the average of AUC across all classifiers and across all project is 0.81. All classifiers achieve more than 0.6 AUC while Random Forests is also the best performer in this aspect with 0.93 AUC.

Overall, the evaluation results demonstrate the effectiveness of our predictive models, achieving on average 48%–81% precision, 23%–90% recall, 29%–71% F-measure, and 70%–92% Area Under the ROC Curve. Our predictive models also have low error rates: 0.39–0.75 for Macro-averaged Mean Cost-Error and 0.7–1.2 for Macro-averaged Mean Absolute Error. Although Random Forests consistently perform well across different measures, this classifier does not provide easily explainable models. Easy-to-understand models such as decision trees tend to be more preferable for practitioners [16].

The variety of project nature: The predictive performance of the five classifiers vary from projects to projects. For example, from the F-measure report (Figure 1 (c.)), the predictive performance of all classifiers for the JBoss project is worst than that for the other projects. This demonstrate the diversity of open source projects in nature. For projects that our predictive models struggled, there might be some changes in the project (e.g. additional contributors joined in) between

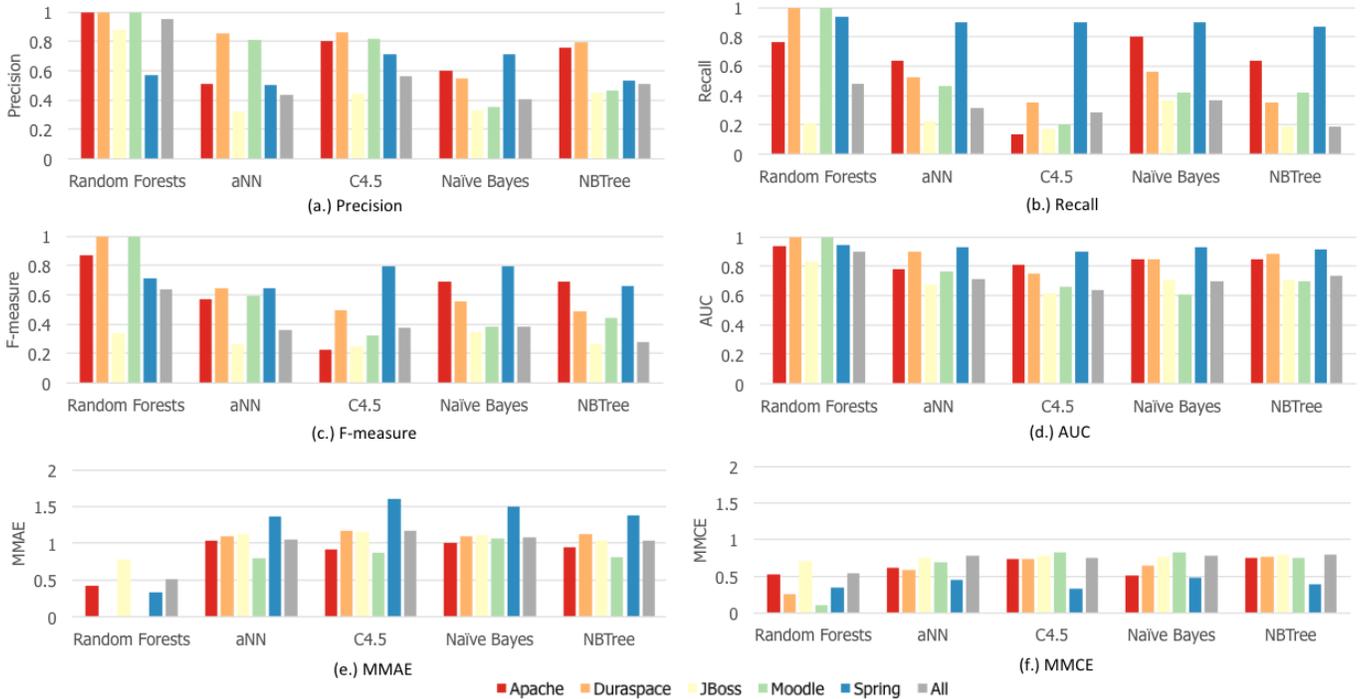


Fig. 1: Evaluation results
(Precision/Recall/F-measure/AUC, the higher the better. The MMAE/MMCE, the lower the better.)

the training time and test time, and thus patterns present at training time may not entirely repeat later on at test time.

MMAE and MMCE as performance measures: MMAE and MMCE are used to assess the performance of our models in terms of predicting risk exposure. 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.

What did we learn from the risk factor weights: Outcomes from the risk factor selection process (see Section III – Table II) 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 projects, we have also seen some common patterns, e.g. “the percentage of delayed issues that a developer involved with” is a positive factor across all the five projects. In a term of discrimination power of the risk factors, the top three highest discrimination power factors are: the percentage of delayed issues that a developer involved with, discussion time, and the number of times that issue is reopened. In addition, the issue’s type that is consistently indicative of delays is “Documentation”, which might reflect that this type of issue does not have enough attention in open source projects. Furthermore, the “Trivial” priority has less impact on causing a delay, while the “Blocker” priority has stronger impact (particularly for the Spring project). These would provide insightful and actionable information for project managers in risk management.

VI. THREATS TO VALIDITY

Internal validity: Our data set has the class imbalance problem. The majority of issues (over 90% of the total data) are non-delayed issues. This has implications to a classifier’s ability to learn to identify delayed issues. We have used stratified sampling to mitigate this problem. We also designed and used two performance measures that are insensitive to class imbalance: the MMCE and MMAE. In addition, classifiers generally make a certain assumptions about the data, e.g. Naive Bayes assumes that the factors are conditionally independent (which may not be true in our context,) or the other classifiers generally assume that training data is sufficiently large. We have used a range of different classifiers, and performed feature selection to minimize this threat. Feature selection reduces the feature space, limits the chance of variations, and thus requires less data to learn patterns out of features. Especially, our feature selection is based on maximizing predictive performance in held-out data, and consequently it helps deal with overfitting and overoptimistic estimation.

Another threat to our study is that the patterns that hold in the train 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). The sliding approach discussed in Section V-A would also minimize this threat. We have attempted to cover most important risk factors related to

an issue causing a project delay. 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 40,000 issue reports from the five 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 setting. 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 [48]. 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.

VII. RELATED WORK

Software risk management has attracted great attention since Boehm's seminal work (e.g. [4, 46]) 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 [4, 49]. Risk control aims to develop, engage, and monitor risk mitigation plans.

Statistical and machine learning techniques have been used in different aspects of risk management. For example, Letier *et al.* [50] proposed a statistical decision analysis approach to provide a statistical support on complex requirements and architecture. 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.* [12] 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. For example, the work in [8] 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. [18, 19, 51–53]), blocking bug prediction (e.g. [16]), re-opened bug prediction (e.g. [13, 15]), severity/priority prediction (e.g. [54, 55]), delays in the integration of a resolved issue to a release (e.g. [56]), bug triaging (e.g. [57–60]), and duplicate bug detection ([61–65]). 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 [51] 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 [18] 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 [53] proposed to use Random Forest to predict bug's fixing time using three features: location, reporter and description. The work in [52] 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 variable, they need to discretize the fix-time into categories (e.g. within 1 month, 1 year and more than 1 year as in [53]). 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 cause a delay. Our future work would involve investigating how to extend those techniques for delay prediction and compare them with our approach.

VIII. CONCLUSIONS AND FUTURE WORK

Schedule overruns are common and one of the major problems in software projects. Thus, there is increasing need for project managers and decision makers to identify risks causing project delays early. In this paper, we have performed a study in five major open source projects and extracted a comprehensive set of risk factors that are related to the issues (software tasks) in those projects. We have developed a sparse logistic regression model and performed feature selection on those risk factors to choose those with good discriminative power. Using those selected risk factors, we have developed accurate models to predict if an issue will cause a delay, if so to what extent 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.

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. Performing additional experiments with different training and test sets (using the sliding window approach) is also part of our future work. 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

- [1] B. Michael, S. Blumberg, and J. Laartz, "Delivering large-scale IT projects on time, on budget, and on value," 2012. [Online]. Available: http://www.mckinsey.com/insights/business_technology/delivering_large-scale_it_projects_on_time_on_budget_and_on_value
- [2] S. Group, "Chaos report," West Yarmouth, Massachusetts: Standish Group, Tech. Rep., 2004.
- [3] B. Flyvbjerg and A. Budzier, "Why Your IT Project May Be Riskier Than You Think," *Harvard Business Review*, vol. 89, no. 9, pp. 601–603, 2011.
- [4] B. W. Boehm, "Software risk management: principles and practices," *Software, IEEE*, vol. 8, no. 1, pp. 32–41, 1991.
- [5] M. J. Carr and S. L. Konda, "Taxonomy-Based Risk Identification," Software Engineering Institute, Carnegie Mellon University, Tech. Rep. June, 1993.
- [6] L. Huang, D. Port, L. Wang, T. Xie, and T. Menzies, "Text mining in supporting software systems risk assurance," in *Proceedings of the IEEE/ACM international conference on Automated software engineering - ASE '10*, ser. ASE '10. New York, New York, USA: ACM Press, 2010, p. 163.
- [7] Z. Xu, B. Yang, and P. Guo, "Software Risk Prediction Based on the Hybrid Algorithm of Genetic Algorithm and Decision Tree," in *Advanced Intelligent Computing Theories and Applications. With Aspects of Contemporary Intelligent Computing Techniques SE - 30*, ser. Communications in Computer and Information Science, D.-S. Huang, L. Heutte, and M. Loog, Eds. Springer Berlin Heidelberg, 2007, vol. 2, pp. 266–274.
- [8] Y. Hu, X. Zhang, E. Ngai, R. Cai, and M. Liu, "Software project risk analysis using Bayesian networks with causality constraints," *Decision Support Systems*, vol. 56, pp. 439–449, Dec. 2013.
- [9] C. Fang and F. Marle, "A simulation-based risk network model for decision support in project risk management," *Decision Support Systems*, vol. 52, no. 3, pp. 635–644, Feb. 2012. [Online]. Available: <http://linkinghub.elsevier.com/retrieve/pii/S0167923611002016>
- [10] M. N. Moreno García, I. R. Román, F. J. García Peñalvo, and M. T. Bonilla, "An association rule mining method for estimating the impact of project management policies on software quality, development time and effort," *Expert Systems with Applications*, vol. 34, no. 1, pp. 522–529, Jan. 2008.
- [11] A. Iqbal, "Understanding Contributor to Developer Turnover Patterns in OSS Projects : A Case Study of Apache Projects," vol. 2014, 2014.
- [12] A. Pika, W. M. van der Aalst, C. J. Fidge, A. H. ter Hofstede, M. T. Wynn, and W. V. D. Aalst, "Profiling event logs to configure risk indicators for process delays," *Advanced Information Systems Engineering (CAISE 2013)*, pp. 465–481, Jul. 2013.
- [13] T. Zimmermann, N. Nagappan, P. J. Guo, and B. Murphy, "Characterizing and predicting which bugs get reopened," in *34th International Conference on Software Engineering (ICSE), 2012*. IEEE Press, Jun. 2012, pp. 1074–1083.
- [14] P. J. Guo, T. Zimmermann, N. Nagappan, and B. Murphy, "Characterizing and predicting which bugs get fixed: an empirical study of Microsoft Windows," *2010 ACM/IEEE 32nd International Conference on Software Engineering*, vol. 1, pp. 495–504, 2010.
- [15] E. Shihab, A. Ihara, Y. Kamei, W. M. Ibrahim, M. Ohira, B. Adams, A. E. Hassan, and K.-i. Matsumoto, "Studying re-opened bugs in open source software," *Empirical Software Engineering*, vol. 18, no. 5, pp. 1005–1042, Sep. 2012.
- [16] H. Valdivia Garcia, E. Shihab, and H. V. Garcia, "Characterizing and predicting blocking bugs in open source projects," in *Proceedings of the 11th Working Conference on Mining Software Repositories - MSR 2014*. ACM Press, May 2014, pp. 72–81.
- [17] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann, "What makes a good bug report?" in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering - SIGSOFT '08/FSE-16*. New York, New York, USA: ACM Press, Nov. 2008, p. 308. [Online]. Available: <http://dl.acm.org/citation.cfm?id=1453101.1453146>
- [18] L. D. Panjer, "Predicting Eclipse Bug Lifetimes," in *Fourth International Workshop on Mining Software Repositories (MSR'07:ICSE Workshops 2007)*. IEEE, May 2007, pp. 29–29.
- [19] P. Bhattacharya and I. Neamtiu, "Bug-fix time prediction models," in *Proceeding of the 8th working conference on Mining software repositories - MSR '11*. New York, New York, USA: ACM Press, May 2011, p. 207. [Online]. Available: <http://dl.acm.org/citation.cfm?id=1985441.1985472>
- [20] P. Hooimeijer and W. Weimer, "Modeling bug report quality," in *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering - ASE '07*. ACM Press, Nov. 2007, p. 34. [Online]. Available: <http://dl.acm.org/citation.cfm?id=1321631.1321639>
- [21] W.-M. Han and S.-J. Huang, "An empirical analysis of risk components and performance on software projects," *Journal of Systems and Software*, vol. 80, no. 1, pp. 42–50, Jan. 2007.
- [22] A. A. Porter, H. P. Siy, and L. G. Votta, "Understanding the effects of developer activities on inspection interval," in *Proceedings of the 19th international conference on Software engineering - ICSE '97*. ACM Press, May 1997, pp. 128–138.
- [23] L. Wallace and M. Keil, "Software project risks and their effect on outcomes," *Communications of the ACM*, vol. 47, no. 4, pp. 68–73, Apr. 2004.
- [24] S. Kim, T. Zimmermann, K. Pan, and E. Jr. Whitehead, "Automatic Identification of Bug-Introducing Changes," in *21st IEEE/ACM International Conference on Automated Software Engineering (ASE'06)*. IEEE, Sep. 2006, pp. 81–90. [Online]. Available: <http://dl.acm.org/citation.cfm?id=1169218.1169308>
- [25] I. Guyon and A. Elisseeff, "An Introduction to Variable and Feature Selection," *The Journal of Machine Learning Research*, vol. 3, pp. 1157–1182, Mar. 2003. [Online]. Available: <http://dl.acm.org/citation.cfm?id=944919.944968>
- [26] S.-I. Lee, H. Lee, P. Abbeel, and A. Y. Ng, "Efficient l_1 regularized logistic regression," in *Proceedings of the National Conference on Artificial Intelligence*, vol. 21, no. 1. Menlo Park, CA; Cambridge, MA; London; AAI Press; MIT Press; 1999, 2006, p. 401.
- [27] D. H. Jr and S. Lemeshow, *Applied logistic regression*, 2nd ed. Wiley-Interscience Publication, 2004.
- [28] L. Breiman, "Random forests," *Machine learning*, pp. 5–32, 2001.
- [29] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings," *IEEE Transactions on Software Engineering*, vol. 34, no. 4, pp. 485–496, Jul. 2008.
- [30] Y. Kamei, S. Matsumoto, A. Monden, K.-i. Matsumoto, B. Adams, and A. E. Hassan, "Revisiting common bug prediction findings using effort-aware models," in *2010 IEEE International Conference on Software Maintenance*. IEEE, Sep. 2010, pp. 1–10.
- [31] Y. Hu, J. Huang, J. Chen, M. Liu, K. Xie, and S. Yat-sen, "Software Project Risk Management Modeling with Neural Network and Support Vector Machine Approaches," *Natural Computation, 2007. ICNC 2007. Third International Conference on*, vol. 3, no. 70572053, pp. 358–362, 2007.
- [32] D. Neumann, "An enhanced neural network technique for software risk analysis," *IEEE Transactions on Software Engineering*, vol. 28, no. 9, pp. 904–912, Sep. 2002.
- [33] Q. Wang, J. Zhu, and B. Yu, "Combining Classifiers in Software Quality Prediction : A Neural Network Approach," pp. 921–926, 2005.
- [34] J. R. Quinlan, "C4.5: programs for machine learning," Mar. 1993.
- [35] R. Conforti, M. de Leoni, M. La Rosa, W. M. van der Aalst, and A. H. ter Hofstede, "A recommendation system for predicting risks across multiple business process instances," *Decision Support Systems*, vol. 69, pp. 1–19, Jan. 2015.
- [36] W. M. Ibrahim, N. Bettenburg, E. Shihab, B. Adams, and A. E. Hassan, "Should I contribute to this discussion?" *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*, pp. 181–190, May 2010.
- [37] J. Wolfson, S. Bandyopadhyay, M. Elidrissi, G. Vazquez-Benitez, D. Musgrove, G. Adomavicius, P. Johnson, and P. O'Connor, "A Naive Bayes machine learning approach to risk prediction using censored, time-to-event data," p. 21, Apr. 2014.
- [38] W. Abdelmoez, M. Kholief, and F. M. Elsalmy, "Bug Fix-Time Prediction Model Using Naïve Bayes Classifier," in *2012 22nd International Conference on Computer Theory and Applications (ICCTA)*, no. October, 2012, pp. 13–15.
- [39] R. Kohavi, "Scaling Up the Accuracy of Naive-Bayes Classifiers: A Decision-Tree Hybrid." *KDD*, 1996.
- [40] B. Zadrozny and C. Elkan, "Transforming classifier scores into accurate multiclass probability estimates," in *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2002, pp. 694–699.

- [41] Y. L. John Denker, "Transforming Neural-Net Output Levels to Probability Distributions." [Online]. Available: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.32.7096>
- [42] M. D. Richard and R. P. Lippmann, "Neural Network Classifiers Estimate Bayesian a posteriori Probabilities," *Neural Computation*, vol. 3, no. 4, pp. 461–483, Dec. 1991. [Online]. Available: <http://www.mitpressjournals.org/doi/abs/10.1162/neco.1991.3.4.461>\#.VN61hS5GTSF
- [43] N. Chawla and D. Cieslak, "Evaluating probability estimates from decision trees," *American Association for Artificial Intelligence*, 2006.
- [44] A. Garg and D. Roth, *Understanding Probabilistic Classifiers*, ser. Lecture Notes in Computer Science, L. De Raedt and P. Flach, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, Aug. 2001, vol. 2167. [Online]. Available: <http://link.springer.com/10.1007/3-540-44795-4>
- [45] L.-M. Wang, X.-L. Li, C.-H. Cao, and S.-M. Yuan, "Combining decision tree and Naive Bayes for classification," *Knowledge-Based Systems*, vol. 19, no. 7, pp. 511–515, Nov. 2006.
- [46] B. Boehm, *Software risk management*. Springer, 1989.
- [47] S. Baccianella, a. Esuli, and F. Sebastiani, "Evaluation Measures for Ordinal Regression," *2009 Ninth International Conference on Intelligent Systems Design and Applications*, 2009.
- [48] X. Qin, M. Salter-Townshend, and P. Cunningham, "Exploring the Relationship between Membership Turnover and Productivity in Online Communities," *CoRR*, 2014.
- [49] Xu Ruzhi, Q. leqiu, and Jing Xinhai, "CMM-based software risk control optimization," in *Proceedings Fifth IEEE Workshop on Mobile Computing Systems and Applications*. IEEE, 2003, pp. 499–503.
- [50] E. Letier, D. Stefan, and E. T. Barr, "Uncertainty, risk, and information value in software requirements and architecture," in *Proceedings of the 36th International Conference on Software Engineering - ICSE 2014*. New York, New York, USA: ACM Press, May 2014, pp. 883–894. [Online]. Available: <http://dl.acm.org/citation.cfm?id=2568225.2568239>
- [51] C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller, "How long will it take to fix this bug?" in *Proceedings of the Fourth International Workshop on Mining Software Repositories*, ser. MSR '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 1–.
- [52] E. Giger, M. Pinzger, and H. Gall, "Predicting the fix time of bugs," in *Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering - RSSE '10*. ACM Press, May 2010, pp. 52–56. [Online]. Available: <http://dl.acm.org/citation.cfm?id=1808920.1808933>
- [53] L. Marks, Y. Zou, and A. E. Hassan, "Studying the fix-time for bugs in large open source projects," in *Proceedings of the 7th International Conference on Predictive Models in Software Engineering*, ser. Promise '11. New York, NY, USA: ACM, 2011, pp. 11:1–11:8.
- [54] A. Lamkanfi, S. Demeyer, E. Giger, and B. Goethals, "Predicting the severity of a reported bug," in *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*. IEEE, May 2010, pp. 1–10. [Online]. Available: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5463284>
- [55] T. Menzies and A. Marcus, "Automated severity assessment of software defect reports," in *Software Maintenance, 2008. ICSM 2008. IEEE International Conference on*, September 2008, pp. 346–355.
- [56] D. A. da Costa, S. L. Abebe, S. McIntosh, U. Kulesza, and A. E. Hassan, "An empirical study of delays in the integration of addressed issues," in *Proc. of the 30th Int'l Conf. on Software Maintenance and Evolution (ICSME)*, 2014, pp. 281–290.
- [57] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?" in *Proceeding of the 28th international conference on Software engineering - ICSE '06*. New York, New York, USA: ACM Press, May 2006, p. 361.
- [58] J. Anvik and G. C. Murphy, "Reducing the effort of bug report triage," *ACM Transactions on Software Engineering and Methodology*, vol. 20, no. 3, pp. 1–35, Aug. 2011.
- [59] M. M. Rahman, G. Ruhe, and T. Zimmermann, "Optimized assignment of developers for fixing bugs an initial evaluation for eclipse projects," in *2009 3rd International Symposium on Empirical Software Engineering and Measurement*. IEEE, Oct. 2009, pp. 439–442.
- [60] G. Murphy and D. Čubranić, "Automatic bug triage using text categorization," in *Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering*, 2004.
- [61] P. Runeson, M. Alexandersson, and O. Nyholm, "Detection of Duplicate Defect Reports Using Natural Language Processing," in *29th International Conference on Software Engineering (ICSE'07)*. IEEE, May 2007, pp. 499–510.
- [62] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun, "An approach to detecting duplicate bug reports using natural language and execution information," in *Proceedings of the 30th international conference on Software engineering*, 2008, pp. 461–470.
- [63] N. Bettenburg, R. Premraj, and T. Zimmermann, "Duplicate bug reports considered harmful ... really?" in *2008 IEEE International Conference on Software Maintenance*. IEEE, 2008, pp. 337–345.
- [64] C. Sun, D. Lo, S.-C. Khoo, and J. Jiang, "Towards more accurate retrieval of duplicate bug reports," *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*, pp. 253–262, Nov. 2011.
- [65] N. Jalbert and W. Weimer, "Automated duplicate detection for bug tracking systems," in *2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN)*. IEEE, 2008, pp. 52–61.